Moneris™

BE PAYMENT READY

.NET - North American API - Integration Guide

Version: 1.0.0

Security and Compliance

Your solution may be required to demonstrate compliance with the card associations' PCI/CISP/PABP requirements. For more information on how to make your application PCI-DSS compliant, contact the Moneris Sales Center and visit https://developer.moneris.com to download the PCI_DSS Implementation Guide.

All Merchants and Service Providers that store, process, or transmit cardholder data must comply with PCI DSS and the Card Association Compliance Programs. However, certification requirements vary by business and are contingent upon your "Merchant Level" or "Service Provider Level".

The card association has some data security standards that define specific requirements for all organizations that store, process, or transmit cardholder data. As a Moneris client or partner using this method of integration, your solution must demonstrate compliance to the Payment Card Industry Data Security Standard (PCI DSS) and/or the Payment Application Data Security Standard (PA DSS). These standards are designed to help the cardholders and merchants in such ways as they ensure credit card numbers are encrypted when transmitted/stored in a database and that merchants have strong access control measures.

Non-compliant solutions may prevent merchant boarding with Moneris. A non-compliant merchant can also be subject to fines, fees, assessments or termination of processing services.

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

Confidentiality

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.

Table of Contents

List of Tables

1 About This Documentation

1.1 Purpose

This document describes the transaction information for using the .NET API for sending credit card transactions. In particular, it describes the format for sending transactions and the corresponding responses you will receive.

This document contains information about the following features:

- Basic transactions
- MPI
- Convenience fee
- IOP (INTERAC® Online Payment)
- ACH (Automated Clearing House)
- Vault
- MSR (Magnetic Swipe Reader) and Encrypted MSR
- Contactless

1.2 Who Is This Guide For?

The North American API - Integration Guide is intended for developers integrating with Moneris Payment Gateway.

This guide assumes that the system you are trying to integrate meets the requirements outlined below and that you have some familiarity with the .NET programming language.

System Requirements

- Java 1.6 or above
- Port 443 open for bi-directional communication
- Web server with a SSL certificate

2 Testing a Solution

2.1 Merchant Resource Centre

The Merchant Resource Center is the user interface for Moneris Payment Gateway services. There is also a QA version of the Merchant Resource Centre site specifically allocated for you and other developers to use to test your API integrations with the gateway.

You can access the Merchant Resource Center in the test environment at:

https://esqa.moneris.com/mpg (Canada)

https://esplusqa.moneris.com/usmpg (United States)

The test environment is generally available 24×7, but 100% availability is not guaranteed. Also, please be aware that other merchants are using the test environment in the Merchant Resource Center. Therefore, you may see transactions and user IDs that you did not create. As a courtesy to others who are testing, we ask that you use only the transactions/users that you created. This applies to processing Refund transactions, changing passwords or trying other functions.

2.2 Testing INTERAC® Online Payment Solutions

Acxsys has two websites where merchants can post transactions for testing the fund guarantee porting of INTERAC® Online Payment transactions. The test `IDEBIT_MERCHNUM` value is provided by Moneris after registering in the test environment.

After registering, the following two links become accessible:

- Merchant Test Tool
- Certification Test Tool

Merchant Test Tool

https://merchant-test.interacidebit.ca/gateway/merchant_test_processor.do

This URL is used to simulate the transaction response process, to validate response variables, and to properly integrate your checkout process.

When testing INTERAC® Online Payment transactions, you are forwarded to the INTERAC® Online Payment Merchant Testing Tool. A screen appears where certain fields need to be completed.

For an approved response, do not alter any of the fields except for the ones listed here.

IDEBIT_TRACK2
To form a track2 when testing with the Moneris Gateway, use one of these three numbers:

3728024906540591206=01121122334455000

5268051119993326=01121122334455000000

453781122255=011211223344550000000000

IDEBIT_ISSNAME
RBC

IDEBIT_ISSCONF
123456

For a declined response, provide any other value as the IDEBIT_TRACK2. Click **Post to Merchant**.

Whether the transaction is approved or declined, do **not** click **Validate Data**. This will return validation errors.

Certification Test Tool

https://merchant-test.interacidebit.ca/gateway/merchant_certification_processor.do

This URL is used to complete the required INTERAC® Online Payment Merchant Front-End Certification test cases, which are outlined in Appendix K (page 309) and Appendix L (page 313).

To confirm the fund that was guaranteed above, an INTERAC® Online Payment Purchase (see page 87) must be sent to the Moneris Payment Gateway QAusing the following test store information:

Host: esqa.moneris.com

Store ID: store3

API Token: yesguy

You can always log into the Merchant Resource Center to check the results using the following information:

URL: **https://esqa.moneris.com/mpg**

Store ID: store3

Note that all response variables that are posted back from the IOP gateway in step 4 of 7.3 must be validated for length of field, permitted characters and invalid characters.

2.3 Testing MPI Solutions

When testing your implementation of the Moneris MPI, you can use the VISA/MasterCard/Amex PIT (production integration testing) environment. The testing process is slightly different than a production environment in that when the inLine window is generated, it does not contain any input boxes. Instead, it contains a window of data and a **Submit** button. Clicking **Submit** loads the response in the testing window. The response will not be displayed in production.

Note	MasterCard SecureCode may not be directly tested within our current test environment. However, the process and behavior tested with the Visa test cards will be the same for MCSC.

When testing you may use the following test card numbers with any future expiry date. Use the appropriate test card information from the tables below: Visa and Mastercard use the same test card information, while Amex uses unique information.

Table 1: MPI test card numbers (Visa and Mastercard only)

Card Number	VERes	PARes	Action
4012001037141112	Y	true	TXN – Call function to create inLine window. ACS – Send CAVV to Moneris Payment Gateway using either the Cavv Purchase or the Cavv Pre-Authorization transaction.
4012001038488884	U	NA	Send transaction to Moneris Payment Gateway using either the basic Purchase or the basic Pre-Authorization transaction. Set crypt_type = 7.
4012001038443335	N	NA	Send transaction to Moneris Payment Gateway using either the basic Purchase or the basic Pre-Authorization transaction. Set crypt_type = 6.
4242424242424242	Y	true	TXN – call function to create inLine window. ACS – Send CAVV to Moneris Payment Gateway using either the Cavv Purchase or the Cavv Pre-authorization transaction.
4012001037461114	Y	false	Card failed to authenticate. Merchant may chose to send transaction or decline transaction. If transaction is sent, use crypt type = 7.

Table 2: MPI test card numbers (Amex only)

Card Number	VERes	PARes	Action
375987000000062			Set crypt_type = 7.
375987000000021			Set crypt_type = 7.
375987000000013			Set crypt_type = 6.
374500261001009			Set crypt_type = 5.

VERes
The result U, Y or N is obtained by using getMessage().

PARes
The result “true” or “false” is obtained by using getSuccess().

To access the Merchant Resource Centre in the test environment go to https://esqa.moneris.com/mpg (Canada) or https://esplusqa.moneris.com/usmpg (USA).

Transactions in the test environment should not exceed $11.00.

2.4 Test Credentials

When testing, use the test credentials provided in the following tables with the corresponding lines of code, as in the examples below.

For Canada:

Table 3: Test Server Credentials - Canada

store_id	api_token	Username	Password	Other Information
store1	yesguy	demouser	password	
store2	yesguy	demouser	password	
store3	yesguy	demouser	password	
store4	yesguy	demouser	password	
store5	yesguy	demouser	password	
monca00392	yesguy	demouser	password	Use this store to test Convenience Fee transactions

For US:

Table 4: Test Server Credentials - USA

store_id	api_token	Username	Password	Other Information
monusqa002	qatoken	demouser	abc1234	
monusqa003	qatoken	demouser	abc1234	
monusqa004	qatoken	demouser	abc1234	
monusqa005	qatoken	demouser	abc1234	
monusqa006	qatoken	demouser	abc1234	
monusqa024	qatoken	demouser	abc1234	For testing ACH transactions only
monusqa025	qatoken	demouser	abc1234	For testing both ACH and Credit Card transactions
monusqsa138	qatoken	demouser	abc1234	For testing Convenience Fee transactions

2.5 Test Cards

Because of security and compliance reasons, the use of live credit and debit card numbers for testing is strictly prohibited. Only test credit and debit card numbers are to be used.

To test general transactions, use the following test card numbers:

Table 5: General test card numbers

Card Plan	Card Number
MasterCard	5454545454545454
Visa	4242424242424242
Amex	373599005095005
JCB	3566007770015365
Diners	36462462742008
Track2	5258968987035454=06061015454001060101?

To test Level 2/3 transactions, use the following test card numbers:

Table 6: Level 2/3 test card numbers

Card Plan	Card Number
MasterCard	5454545442424242
Visa	4242424254545454
Amex	373269005095005
Diners	36462462742008

To test ACH transactions (US only), use the following account details:

Financial institution: FEDERAL RESERVE BANK

Routing Number: 011000015

Account number: Any number between 5 and 22 digits

Check number: Any number

2.6 Simulator Host

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

The test environment approves and declines transactions based on the penny value of the amount sent. For example, a transaction made for the amount of $9.00 or $1.00 is approved because of the .00 penny value.

Transactions in the test environment must not exceed $11.00.

For a list of all current test environment responses for various penny values, please see the Test Environment Penny Response Table available at https://developer.moneris.com.

Remember

These responses may change without notice. Check the Moneris Developer Portal (https://developer.moneris.com) regularly to access the latest documentation and downloads.

3 Moving to Production

3.1 Activating a Store

The steps below outline how to activate your production account so that you can process production transactions.

1. Obtain your activation letter/fax from Moneris.
2. Go to https://www3.moneris.com/connect/en/activate/index.php(Canada) or https://esplus.-moneris.com/usmpg/activate (United States) as instructed in the letter/fax.
3. Input your store ID and merchant ID from the letter/fax and click **Activate**.
4. Follow the on-screen instructions to create an administrator account. This account will grant you access to the Merchant Resource Center.
5. Log into the Merchant Resource Center at https://www3.moneris.com/mpg (Canada) or https://esplus.moneris.com/usmpg (US) using the user credentials created in step 4.
6. Proceed to **ADMIN** and then **STORE SETTINGS**.
7. Locate the API token at the top of the page. Use this API Token along with the store ID that you received in your letter/fax and to send any production transactions through the API.

For more information about how to use the Merchant Resource Center, see the Moneris Payment Gateway Merchant Resource Center User's Guide, which is available at https://developer.moneris.com.

3.2 Configuring a Store for Production

After you have completed your testing, you are ready to point your store to the production host.

To configure a store for production:

1. Change the test mode setting from true to false.
2. Change the Store ID to reflect your production store ID
3. Change the API token to the production token that you received during activation.

Sample credentials for each set method are included in the table below.

Set method	Production	Development
	`"US"` or `"CA"`	`"US"` or `"CA"`
	`""`	`""`
		(Canada) (US)

Set method	Production	Development
		(Canada) (US)

(where X is an alphanumeric character)

3.2.1 Configuring an INTERAC® Online Payment Store for Production

Before you can process INTERAC® Online Payment transactions through your web site, you need to complete the certification registration process with Moneris, as described below. The production IDEBIT_MERCHNUM value is provided by Moneris after you have successfully completed the certification.

Acxsys' production INTERAC® Online PaymentGateway URL is https://gateway.interaconline.com/merchant_processor.do.

To access the Moneris Moneris Payment Gateway production gateway URL, use the following:

Store ID: Provided by Moneris

API Token: Generated during your store activation process.

Processing country code: CA

The **production** Merchant Resource Center URL is https://www3.moneris.com/mpg/

3.2.1.1 Completing the Certification Registration - Merchants

To complete the certification registration, fax or email the information below to our Integration Support helpdesk:

- Merchant logo to be displayed on the INTERAC® Online Payment Gateway page
  - In both French and English
  - 120 × 30 pixels
  - Only PNG format is supported.
- Merchant business name
  - In both English and French
  - Maximum 30 characters.
- List of all referrer URLs. That is, URLs from which the customer may be redirected to the INTERAC® Online Payment gateway.
- List of all URLs that may appear in the IDEBIT_FUNDEDURL field of the https form POST to the INTERAC® Online Payment Gateway.
- List of all URLs that may appear in the IDEBIT_NOTFUNDEDURL field of the https form POST to the INTERAC® Online Payment Gateway.

3.2.1.2 Third-Party Service/Shopping Cart Provider

In your product documentation, instruct your clients to provide the information below to the Moneris Payment Gateway Integration Support helpdesk for certification registration:

- Merchant logo to be displayed on the INTERAC® Online Payment Gateway page
  - In both French and English
  - 120 × 30 pixels
  - Only PNG format is supported.

- Merchant business name
  - In both English and French
  - Maximum 30 characters.

- List of all referrer URLs. That is, URLs from which the customer may be redirected to the INTERAC® Online Payment gateway.
- List of all URLs that may appear in the IDEBIT_FUNDEDURL field of the https form POST to the INTERAC® Online Payment Gateway.
- List of all URLs that may appear in the IDEBIT_NOTFUNDEDURL field of the https form POST to the INTERAC® Online Payment Gateway.

See 7.2.3, page 84 for additional client requirements.

3.3 Receipt Requirements

Visa and MasterCard expect certain details to be provided to the cardholder and on the receipt when a transaction is approved.

Receipts must comply with the standards outlined within the Integration Receipts Requirements. Forall the receipt requirements covering all transaction scenarios, visit the Moneris Developer Portal at https://developer.moneris.com.

Production of the receipt must begin when the appropriate response to the transaction request is received by the application. The transaction may be any of the following:

- **Sale** (Purchase)
- **Authorization** (PreAuth, Pre-Authorization)
- **Authorization Completion** (Completion, Capture)
- **Offline Sale** (Force Post)
- **Sale Void** (Purchase Correction, Void)
- **Refund**.

The boldface terms listed above are the names for transactions as they are to be displayed on receipts. Other terms used for the transaction are indicated in brackets.

3.3.1 Certification Requirements

Card-present transaction receipts are required to complete certification.

Card-not-present integration
Certification is optional but highly recommended.

Card-present integration

After 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.

Contact a Client Integration Specialist for the Certification Test checklist that must be completed and returned for verification. (See "Getting Help" below for contact details.) Be sure to include the application version of your product. Any further changes to the product after certification requires re-certification.

After the certification requirements are met, Moneris will provide you with an official certification letter.

3.4 Getting Help

Help is available to Moneris merchants at no cost. Ensure that you have your merchant number or store ID handy.

Getting Started

If you are just getting started, a client integration specialist can help with integration and certification.

Contact

- ClientIntegrations@moneris.com
- Monday-Friday: 8:30 am - 8 pm EST.

Development Assistance

If you are already working with an integration specialist and need development assistance, our eProducts technical consultants offer development and technical support.

Contact

- 1-866-562-4354
- eproducts@moneris.com
- Monday-Friday: 8 am - 8 pm EST

Production Support

Already have a live application and need production support? Our Customer Service specialists provide financial and technical support to merchants.

Contact

1-866-319-7450 (24 hours/day, 7 days/week)

eselectplus@moneris.com

4 Processing a Transaction

4.1 Overview

There are some common steps for every transaction that is processed.

1. Instantiate the transaction object (such as Purchase), and update it with object definitions that refer to the individual transaction.
2. Instantiate the HttpsPostRequest connection object and update it with connection information, host information and the transaction object that you created in step 1.

   Section 4.2 (page 25) provides the HttpsPostRequest connection object definition. This object and its variables apply to **every** transaction request.
3. Invoke the HttpsPostRequest object's `send()` method.
4. Instantiate the Receipt object, by invoking the HttpsPostRequest object's get Receipt method. Use this object to retrieve the applicable response details.

Some transactions may require steps in addition to the ones listed here. For example, ACH transactions require the use of an ACHinfo object. Below is a sample Purchase transaction with each major step outlined. For extensive code samples of other transaction types, refer to the API ZIP file.

NOTE	For illustrative purposes, the order in which lines of code appear below may differ slightly from the same sample code presented elsewhere in this document.

```
using System;
using System.Collections.Generic;
using System.Text;
using Moneris;
```

Include all necessary classes.

```
namespace CanadaPurchaseConsoleTest
{
class CanadaPurchaseTest
{
public static void Main(string[] args)
{
```

```
string order_id = "Test" + DateTime.Now.ToString("yyyyMMddhhmmss");
string amount = "5.00";
string pan = "4242424242424242";
string expdate = "1901"; //YYMM format
string crypt = "7";
string processing_country_code = "CA";
```

Define all mandatory values for the transaction object properties.

```
string store_id = "store5";
string api_token = "yesguy";
```

Define all mandatory values for the connection object properties.

```
Purchase purchase = new Purchase();
purchase.SetOrderId(order_id);
purchase.SetAmount(amount);
purchase.SetPan(pan);
purchase.SetExpdate(expdate);
purchase.SetCryptType(crypt);
purchase.SetDynamicDescriptor("2134565");
```

Instantiate the transaction object and assign values to properties.

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production
    transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(purchase);
mpgReq.SetStatusCheck(status_check);
```

Instantiate connection object and assign values to properties, including the transaction object you just created.

```
mpgReq.Send();
```

Invoke the connection object's `send()` method.

```
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("IsVisaDebit = " + receipt.GetIsVisaDebit());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Instantiate the Receipt object and use its get methods to retrieve the desired response data.

4.2 HttpsPostRequest Object

The transaction object that you instantiate becomes a property of this object when you call its set Transaction method.

HttpsPostRequest Object Definition

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

After instantiating the HttpsPostRequest object, update its mandatory values as outlined in Table 7

Table 7: **HttpsPostRequest object mandatory values**

Value	Type	Limits	Set method
	Description		
Processing country code	String	2-character alphabetic	`mpgReq.setProcCountryCode(processing_country_code);`
	`CA` for Canada, `US` for USA.		
Test mode	Boolean	true/false	`mpgReq.setTestMode(true);`
	Set to `true` when in test mode. Set to `false` (or comment out entire line) when in production mode.		
Store ID	String	10-character alphanumeric	`mpgReq.setStoreId(store_id);`
	Unique identifier provided by Moneris upon merchant account set up. See Testing Credentials (2.1, page 14) for test environment details.		
API Token	String	20-character alphanumeric	`mpgReq.setApiToken(api_token);`
	Unique alphanumeric string assigned upon merchant account activation. To locate your production API token, refer to the Merchant Resource Centre Admin Store Settings. See Testing Credentials (2.1, page 14) for test environment details.		
Transaction	Object	Not applicable	`mpgReq.setTransaction(transaction);`
	This argument is one of the numerous transaction types discussed in the rest of this manual. (Such as Purchase, Refund and so on.) This object is instantiated in step 1 on page 1.		

Table 1: **HttpsPostRequest object optional values**

Value	Type	Limits	Set method
	Description		
Status Check	Boolean	true/false	`mpgReq.setStatusCheck (status_check);`
	See "Definition of Request Fields" on page 258. Note that while this value belongs to the HttpsPostRequest object, it is only supported by some transactions. Check the individual transaction definition to find out whether Status Check can be used.		

4.3 Receipt Object

After you send a transaction using the HttpsPostRequest object's send method, you can instantiate a receipt object.

Receipt Object Definition

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

For an in-depth explanation of Receipt object methods and properties, See"Definition of Response Fields" on page 266.

5 Basic Transaction Set

5.1 Basic Transaction Type Definitions

The following is a list of basic transactions that are supported by the .NET API.

Purchase

Verifies funds on the customer's card, removes the funds and prepares them for deposit into the merchant's account.

Pre-Authorization

Verifies and locks funds on the customer's credit card. The funds are locked for a specified amount of time based on the card issuer.

To retrieve the funds that have been locked by a Pre-Authorization transaction so that they may be settled in the merchant's account, a Completion transaction must be performed. A Pre-Authorization transaction may only be "completed" once.

Completion

Retrieves funds that have been locked (by either a Pre-Authorization or a Re-Authorization transaction), and prepares them for settlement into the merchant's account.

Re-Authorization

If a Pre-Authorization transaction has already taken place, and not all the locked funds were released by a Completion transaction, a Re-Authorization allows you to lock the remaining funds so that they can be released by another Completion transaction in the future.

Re-Authorization is necessary because funds that have been locked by a Pre-Authorization transaction can only be released by a Completion transaction **one** time. If the Completion amount is less than the Pre-Authorization amount, the remaining money cannot be "completed".

Force Post

Retrieves the locked funds and prepares them for settlement into the merchant's account.

This is used when a merchant obtains the authorization number directly from the issuer by a third-party authorization method (such as by phone).

Purchase Correction

Restores the **full** amount of a previous Purchase, Completion or Force Post transaction to the cardholder's card, and removes any record of it from the cardholder's statement.

This transaction is sometimes referred to as "void".

This transaction can be used against a Purchase or Completion transaction that occurred same day provided that the batch containing the original transaction remains open. When using the automated closing feature, Batch Close occurs daily between 10 and 11pm Eastern Time.

Refund

Restores all or part of the funds from a Purchase, Completion or Force Post transaction to the cardholder's card. Unlike a Purchase Correction, there is a record of both the initial charge and the refund on the cardholder's statement.

Independent Refund

Credits a specified amount to the cardholder’s credit card. The credit card number and expiry date are mandatory.

It is not necessary for the transaction that you are refunding to have been processed via the Moneris Payment Gateway

Card Verification

Verifies the validity of the credit card, expiry date and any additional details (such as the Card Verification Digits or Address Verification details). It does not verify the available amount or lock any funds on the credit card.

Recur Update

Alters characteristics of a previously registered Recurring Billing transaction.

This transaction is commonly used to update a customer’s credit card information and the number of recurs to the account.

Recurring billing is explained in more detail in Appendix G (page 297). The Recur Update transaction is specifically discussed in G.2 (page 300).

Batch Close

Takes the funds from all Purchase, Completion, Refund and Force Post transactions so that they will be deposited or debited the following business day.

For funds to be deposited the following business day, the batch must close before 11pm Eastern Time.

Open Totals

Returns the details about the currently open batch.

This transaction is similar to the Batch Close. The difference is that it does not close the batch for settlement.

5.2 Purchase

Purchase transaction object definition

```
Purchase purchase = new Purchase();
```

HttpsPostRequest object for Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(purchase);
```

Purchase transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 8: Purchase transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`purchase..SetOrderId(order_id);`
Amount	String	9-character decimal	`purchase..SetAmount(amount);`
Credit card number	String	20-character alpha-numeric	`purchase.SetPan(pan);`
Expiry date	String	4-character alpha-numeric (YYMM format)	`purchase..SetExpdate(expdate);`
E-commerce indicator	String	1-character alpha-numeric[1]	`purchase..SetCryptType(crypt);`
Commcard invoice[2]	String	17-character alpha-numeric	`preauth..SetCommcardInvoice (commcard_invoice);`
Commcard tax amount[3]	String	9-character decimal Must contain at least 3 digits, two of which must be penny values.	`preauth..SetCommcardTaxAmount (commcard_tax_amount);`

[1]Full explanation on page 259

[2]Available to US integrations only.

[3]Available to US integrations only.

Table 8: Purchase transaction object mandatory values

Value	Type	Limits	Set method
Customer inform- ation	Object	Not applicable. See Section Appendix D (page 282).	`preauth..SetCustInfo(customer);`
AVS	Object	Not applicable. See Appendix E (page 288).	`purchase..SetAvsInfo(avsCheck);`
CVD	Object	Not applicable. See Appendix F (page 294).	`purchase..SetCvdInfo(cvdCheck);`
Convenience fee[1]	Object	Not applicable. See Appendix H (page 304).	`purchase.`
Recurring billing	Object	Not applicable. See Section Appendix G (page 297).	`purchase..SetRecur(recurring_cycle);`

Table 9: Purchase transaction object optional values

Value	Type	Limits	Set method
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`
Dynamic descriptor	String	20-character alpha-numeric[3]	`purchase.SetDynamicDescriptor(dynamic_descriptor);`

Sample Purchase - CA

```
using System;
using System.Collections.Generic;
using System.Text;
using Moneris;
namespace CanadaPurchaseConsoleTest
{
class CanadaPurchaseTest
{
public static void Main(string[] args)
{
string order_id = "Test" +
```

Sample Purchase - US

```
namespace Moneris
{
using System;
public class TestUSAPurchase
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
```

[1]Available to US integrations only.

[2]For more information, see Appendix C (page 280).

[3]See "Definition of Request Fields" (page 258) for proper length definition.

Sample Purchase - CA

```
    DateTime.Now.ToString("yyyyMMddhhmmss");
string store_id = "store5";
string api_token = "yesguy";
string amount = "5.00";
string pan = "4242424242424242";
string expdate = "1901"; //YYMM format
string crypt = "7";
string processing_country_code = "CA";
bool status_check = false;
Purchase purchase = new Purchase();
purchase.SetOrderId(order_id);
purchase.SetAmount(amount);
purchase.SetPan(pan);
purchase.SetExpdate(expdate);
purchase.SetCryptType(crypt);
purchase.SetDynamicDescriptor("2134565");
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(purchase);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
```

Sample Purchase - US

```
string amount = "5.00";
string pan = "4242424242424242";
string expdate = "1602"; //YYMM format
string crypt = "7";
string commcard_invoice = "INVC090";
string commcard_tax_amount = "1.00";
string processing_country_code = "US";
bool status_check = false;
Purchase purchase = new Purchase();
purchase.SetOrderId(order_id);
purchase.SetAmount(amount);
purchase.SetPan(pan);
purchase.SetExpdate(expdate);
purchase.SetCryptType(crypt);
purchase.SetCommcardInvoice(commcard_invoice);
purchase.SetCommcardTaxAmount(commcard_tax_
    amount);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(purchase);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
```

Sample Purchase - CA

```
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("IsVisaDebit = " +
    receipt.GetIsVisaDebit());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Sample Purchase - US

```
//Console.WriteLine("CardLevelResult = " +
    receipt.GetCardLevelResult());
//Console.WriteLine("StatusCode = " +
    receipt.GetStatusCode());
//Console.WriteLine("StatusMessage = " +
    receipt.GetStatusMessage());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

5.3 Pre-Authorization

Things to consider:

- If a Pre-Authorization transaction is not followed by a Completion transaction, it must be reversed via a Completion transaction for 0.00. See "Completion" on page 36
- A Pre-Authorization transaction may only be "completed" once . If the Completion transaction is for less than the original amount, a Re-Authorization transaction is required to collect the remaining funds by another Completion transaction. See "Re-Authorization" (page 39).
- For a process flow, see "Process Flow for Basic PreAuth, ReAuth and Completion Transactions" on page 308

Pre-Authorization transaction object definition

```
PreAuth preauth = new PreAuth();
```

HttpsPostRequest object for Pre-Authorization transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(preauth);
```

Pre-Authorization transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 10: Pre-Authorization object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`preauth..SetOrderId(order_id);`
Amount	String	9-character decimal	`preauth..SetAmount(amount);`
Credit card number	String	20-character numeric	`preauth.SetPan(pan);`

Table 10: Pre-Authorization object mandatory values (continued)

Value	Type	Limits	Set method
Expiry date	String	4-character numeric	`preauth..SetExpdate (expdate);`
E-Commerce indicator	String	1-character alphanumeric[1]	`preauth..SetCryptType (crypt);`

Table 1: Pre-Authorization object optional values

Value	Type	Limits	Set method
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck(status_ check);`
Dynamic descriptor	String	20-character alphanumeric[3]	`preauth..SetDynamicDescriptor (dynamic_descriptor);`
Customer information	Object	Not applicable. See Section Appendix D (page 282).	`preauth..SetCustInfo(customer);`
AVS	Object	Not applicable. See Appendix E (page 288).	`preauth..SetAvsInfo(avsCheck);`
CVD	Object	Not applicable. See Appendix F (page 294).	`preauth..SetCvdInfo(cvdCheck);`
Customer ID	String	50-character alphanumeric	`preauth.SetCustId(cust_id);`

Sample Pre-Authorization - CA

```
using System;
using System.Collections.Generic;
using System.Text;
using Moneris;
namespace CanadaPurchaseConsoleTest
{
class CanadaPreauthTest
{
public static void Main(string[] args)
{
string store_id = "store5";
string api_token = "yesguy";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string amount = "5.00";
string pan = "4242424242424242";
string expdate = "0412";
```

Sample Pre-Authorization - US

```
namespace Moneris
{
using System;
public class USAPreAuthTest
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string amount = "10.00";
string pan = "4242424242424242";
string expdate = "1902"; //YYMM format
string crypt = "7";
string processing_country_code = "US";
bool status_check = false;
```

[1]Full explanation on page 259

[2]For more information, see Appendix C (page 280).

[3]See "Definition of Request Fields" (page 258) for proper length definition

Sample Pre-Authorization - CA

```
string crypt = "7";
string processing_country_code = "CA";
bool status_check = false;
PreAuth preauth = new PreAuth();
preauth.SetOrderId(order_id);
preauth.SetAmount(amount);
preauth.SetPan(pan);
preauth.SetExpdate(expdate);
preauth.SetCryptType(crypt);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(preauth);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("IsVisaDebit = " +
    receipt.GetIsVisaDebit());
//Console.WriteLine("StatusCode = " +
    receipt.GetStatusCode());
```

Sample Pre-Authorization - US

```
PreAuth preauth = new PreAuth();
preauth.SetOrderId(order_id);
preauth.SetAmount(amount);
preauth.SetPan(pan);
preauth.SetExpdate(expdate);
preauth.SetCryptType(crypt);
preauth.SetCommcardInvoice("123456");
preauth.SetCommcardTaxAmount("1.00");
preauth.SetDynamicDescriptor("2134565");
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(preauth);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
//Console.WriteLine("CardLevelResult = " +
    receipt.GetCardLevelResult());
//Console.WriteLine("StatusCode = " +
    receipt.GetStatusCode());
//Console.WriteLine("StatusMessage = " +
    receipt.GetStatusMessage());
```

Sample Pre-Authorization - CA	Sample Pre-Authorization - US

```
//Console.WriteLine("StatusMessage = " +
    receipt.GetStatusMessage());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```
	```
	Console.ReadLine();
	}
	catch (Exception e)
	{
	Console.WriteLine(e);
	}
	}
	}
	}
	```

5.4 Completion

Things to consider:

- Completion is also known as "capture" or "pre-authorization completion".
- A Pre-Authorization or Re-Authorization transaction can only be completed once. Refer to the Re-Authorization transaction (page 39 for more information on how to perform multiple Completion transactions.
- To reverse the full amount of a Pre-Authorization transaction, use the Completion transaction with the amount set to `0.00`.
- For a process flow, see "Process Flow for Basic PreAuth, ReAuth and Completion Transactions" on page 308

Completion transaction object

```
Completion completion = new Completion();
```

HttpsPostRequest object for Completion transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(completion);
```

Completion transaction values

To process this transaction, you need the order ID and transaction number from the original Pre-Authorization transaction.

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 11: Completion transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`completion..SetOrderId (order_id);`
Completion Amount	String	9-character decimal	`completion..SetCompAmount (amount);`

Table 11: Completion transaction object mandatory values

Value	Type	Limits	Set method
Transaction number	String	255-character alphanumeric	`completion..SetTxnNumber (txn_number);`
E-Commerce indicator	String	1-character alphanumeric[1]	`completion..SetCryptType (crypt);`

Table 12: Completion transaction optional values

Value	Type	Limits	Set method
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`
Customer ID[3]	String	50-character alpha-numeric	`completion.SetCustId(cust_id);`
Dynamic descriptor	String	20-character alpha-numeric[4]	`completion..SetDynamicDescriptor (dynamic_descriptor);`
Commcard invoice[5]	String	17-character alpha-numeric	`completion..SetCommcardInvoice(commcard_ invoice);`
Commcard tax amount[6]	String	9-character decimal Must contain at least 3 digits, two of which must be penny val-ues.	`completion..SetCommcardTaxAmount(com-mcard_tax_amount);`

Sample Basic Completion - CA	Sample Basic Completion - US

`namespace Moneris`
`{`
`using System;`
`public class TestCanadaCompletion`
`{`
`public static void Main(string[] args)`
`{`
`string store_id = "store5";`
`string api_token = "yesguy";`
	`namespace Moneris`
	`{`
	`using System;`
	`public class TestUSACompletion`
	`{`
	`public static void Main(string[] args)`
	`{`
	`string store_id = "monusqa002";`
	`string api_token = "qatoken";`

[1]Full explanation on page 259

[2]For more information, see Appendix C (page 280).

[3]Available to Canadian integrations only.

[4]See "Definition of Request Fields" (page 258) for proper length definition

[5]Available to US integrations only.

[6]Available to US integrations only.

Sample Basic Completion - CA

```
string order_id = "Test20150625111153";
string amount = "1.00";
string txn_number = "113117-0_10";
string crypt = "7";
string cust_id = "my customer id";
string dynamic_descriptor = "my descriptor";
string processing_country_code = "CA";
bool status_check = false;
Completion completion = new Completion();
completion.SetOrderId(order_id);
completion.SetCompAmount(amount);
completion.SetTxnNumber(txn_number);
completion.SetCryptType(crypt);
completion.SetCustId(cust_id);
completion.SetDynamicDescriptor(dynamic_
    descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(completion);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
```

Sample Basic Completion - US

```
string order_id = "Test20150723033036";
string amount = "1.00";
string txn_number = "856503-0_25";
string crypt = "7";
string cust_id = "my customer id";
string dynamic_descriptor = "my descriptor";
string processing_country_code = "US";
bool status_check = false;
Completion completion = new Completion();
completion.SetOrderId(order_id);
completion.SetCompAmount(amount);
completion.SetTxnNumber(txn_number);
completion.SetCryptType(crypt);
completion.SetCustId(cust_id);
completion.SetDynamicDescriptor(dynamic_
    descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(completion);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
```

Sample Basic Completion - CA	Sample Basic Completion - US

`Console.WriteLine("Ticket = " +`
`    receipt.GetTicket());`
`Console.WriteLine("TimedOut = " +`
`    receipt.GetTimedOut());`
`Console.WriteLine("IsVisaDebit = " +`
`    receipt.GetIsVisaDebit());`
`Console.ReadLine();`
`}`
`catch (Exception e)`
`{`
`Console.WriteLine(e);`
`}`
`}`
`}`
`}`
	`Console.WriteLine("Ticket = " +`
	`    receipt.GetTicket());`
	`Console.WriteLine("TimedOut = " +`
	`    receipt.GetTimedOut());`
	`Console.WriteLine("IsVisaDebit = " +`
	`    receipt.GetIsVisaDebit());`
	`Console.ReadLine();`
	`}`
	`catch (Exception e)`
	`{`
	`Console.WriteLine(e);`
	`}`
	`}`
	`}`
	`}`

5.5 Re-Authorization

For a process flow, "Process Flow for Basic PreAuth, ReAuth and Completion Transactions" (page 308).

Re-Authorization transaction object definition

```
ReAuth reauth = new ReAuth();
```

HttpsPostRequest object for Re-Authorization transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(reauth);
```

Re-Authorization transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 13: Re-Authorization transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`reauth..SetOrderId(order_id);`
Original order ID	String	50-character alphanumeric	`reauth..SetOrigOrderId(orig_order_id);`
Amount	String	9-character decimal	`reauth..SetAmount(amount);`
Transaction number	String	255-character variable character	`reauth..SetTxnNumber(txn_number);`
E-Commerce indicator	String	1-character alphanumeric[1]	`reauth..SetCryptType(crypt);`

[1]Full explanation on page 259

Table 1: Re-Authorization transaction optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alphanumeric	`reauth.SetCustId(cust_id);`
Status check	Boolean	true/false	`mpgReq.SetStatusCheck (status_check);`
Dynamic descriptor[1]	String	20-character alphanumeric[2]	`reauth..SetDynamicDescriptor (dynamic_descriptor);`
Customer information	Object	Not applicable. See Section Appendix D (page 282).	`reauth..SetCustInfo (customer);`
AVS	Object	Not applicable. See Appendix E (page 288).	`reauth..SetAvsInfo (avsCheck);`
CVD	Object	Not applicable. See Appendix F (page 294).	`reauth..SetCvdInfo (cvdCheck);`

Sample Re-Authorization - CA

```
namespace Moneris
{
using System;
public class TestCanadaReauth
{
public static void Main(string[] args)
{
string store_id = "store5";
string api_token = "yesguy";
string order_id =
    "mvt2713557ss83ss9ssdfsdfsdf";
string orig_order_id = "mvt3525350028";
string amount = "1.00";
string txn_number = "113457-0_10";
string crypt = "8";
string dynamic_descriptor = "123456";
string cust_id = "my customer id";
string processing_country_code = "CA";
bool status_check = false;
ReAuth reauth = new ReAuth();
reauth.SetOrderId(order_id);
reauth.SetCustId(cust_id);
reauth.SetOrigOrderId(orig_order_id);
reauth.SetTxnNumber(txn_number);
reauth.SetAmount(amount);
reauth.SetCryptType(crypt);
reauth.SetDynamicDescriptor(dynamic_
    descriptor);
HttpsPostRequest mpgReq = new
    HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_
```

Sample Re-Authorization - US

```
namespace Moneris
{
using System;
public class TestUSAReAuth
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string orig_order_id = "Test20150723033036";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string txn_number = "856503-0_25";
string amount = "1.00";
string crypt = "7";
string descriptor = "my descriptor";
string cust_id = "my customer id";
string processing_country_code = "US";
bool status_check = false;
ReAuth reauth = new ReAuth();
reauth.SetOrderId(order_id);
reauth.SetCustId(cust_id);
reauth.SetOrigOrderId(orig_order_id);
reauth.SetTxnNumber(txn_number);
reauth.SetAmount(amount);
reauth.SetCryptType(crypt);
reauth.SetDynamicDescriptor(descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
```

[1]Available for Canadian integrations only.

[2]See "Definition of Request Fields" (page 258) for proper length definition

Sample Re-Authorization - CA

```
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(reauth);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("IsVisaDebit = " +
    receipt.GetIsVisaDebit());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Sample Re-Authorization - US

```
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(reauth);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

5.6 Force Post

It is not required for the transaction that you are submitting to have been processed via the .NET Moneris Payment Gateway. However, a credit card number, expiry date and original authorization number are required.

Things to consider:

- This transaction is an independent completion where the original Pre-Authorization transaction was not processed via the same Moneris Payment Gateway merchant account.

ForcePost transaction object definition

```
ForcePost forcepost = new ForcePost();
```

HttpsPostRequest object for ForcePost transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(forcepost);
```

Force Post transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 14: ForcePost transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`forcepost..SetOrderId(order_id);`
Amount	String	9-character decimal	`forcepost..SetAmount(amount);`
Credit card number	String	20-character numeric	`forcepost.SetPan(pan);`
Expiry date	String	4-character numeric	`forcepost..SetOrderId(order_id);`
Authorization code	String	8-character alphanumeric	`forcepost..SetAuthCode(auth_code);`
E-Commerce indicator	String	1-character alphanumeric[1]	`forcepost..SetCryptType(crypt);`

Table 15: Force Post transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`forcepost.SetCustId(cust_id);`

[1]Full explanation on page 259

Table 15: Force Post transaction optional values (continued)

Value	Type	Limits	Set method
Dynamic descriptor	String	20-character alphanumeric[1]	`forcepost..SetDynamicDescriptor (dynamic_descriptor);`
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck(status_ check);`

Sample Basic Force Post - CA

```
using System;
namespace Moneris
{
public class TestCanadaForcePost
{
public static void Main(string[] args)
{
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "my customer id";
string store_id = "moneris";
string api_token = "hurgle";
string amount = "59.00";
string pan = "4242424242424242";
string expdate = "1901"; //YYMM format
string auth_code = "88864";
string crypt = "7";
string dynamic_descriptor = "my descriptor";
string processing_country_code = "CA";
bool status_check = false;
ForcePost forcepost = new ForcePost();
forcepost.SetOrderId(order_id);
forcepost.SetCustId(cust_id);
forcepost.SetAmount(amount);
forcepost.SetPan(pan);
forcepost.SetExpdate(expdate);
forcepost.SetAuthCode(auth_code);
forcepost.SetCryptType(crypt);
forcepost.SetDynamicDescriptor(dynamic_
    descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(forcepost);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
```

Sample Basic Force Post - US

```
namespace Moneris
{
using System;
public class TestUSAForcePost
{
public static void Main(string[] args)
{
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string store_id = "monusqa002";
string api_token = "qatoken";
string cust_id = "customer1";
string amount = "10.00";
string pan = "4242424242424242";
string expdate = "1602"; //YYMM format
string auth_code = "AU4R6";
string crypt = "1";
string processing_country_code = "US";
bool status_check = false;
string dynamic_descriptor = "my descriptor";
ForcePost forcepost = new ForcePost();
forcepost.SetOrderId(order_id);
forcepost.SetCustId(cust_id);
forcepost.SetAmount(amount);
forcepost.SetPan(pan);
forcepost.SetExpdate(expdate);
forcepost.SetAuthCode(auth_code);
forcepost.SetCryptType(crypt);
forcepost.SetDynamicDescriptor(dynamic_
    descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(forcepost);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
```

[1]See "Definition of Request Fields" (page 258) for proper length definition

[2]For more information, see Appendix C (page 280).

Sample Basic Force Post - CA

```
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("CorporateCard = " +
    receipt.GetCorporateCard());
//Console.WriteLine("MessageId = " +
    receipt.GetMessageId());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Sample Basic Force Post - US

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

Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
//Console.WriteLine("CardLevelResult = " +
    receipt.GetCardLevelResult());
//Console.WriteLine("StatusCode = " +
    receipt.GetStatusCode());
//Console.WriteLine("StatusMessage = " +
    receipt.GetStatusMessage());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

5.7 Purchase Correction

Things to consider:

- Purchase correction is also known as "void" or "correction".

Purchase Correction transaction object definition

```
PurchaseCorrection purchasecorrection = new PurchaseCorrection();
```

HttpsPostRequest object for Purchase Correction transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(purchasecorrection);
```

Purchase Correction transaction object values

To process this transaction, you need the order ID and the transaction number from the original Completion, Purchase or Force Post transaction.

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 16: Purchase Correction transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`purchasecorrection..SetOrderId(order_id);`
Transaction number	String	255-character variable character	`purchasecorrection..SetTxnNumber(txn_number);`
E-Commerce indicator	String	1-character alphanumeric[1]	`purchasecorrection..SetCryptType(crypt);`

Table 17: Purchase Correction transaction optional values

Value	Type	Limits	Set method
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`
Customer ID	String	50-character alpha-numeric	`purchasecorrection.SetCustId(cust_id);`
Dynamic descriptor[3]	String	20-character alpha-numeric[4]	`purchasecorrection..SetDynamicDescriptor(dynamic_descriptor);`

Sample Purchase Correction - CA	Sample Purchase Correction - US
`namespace Moneris`	`namespace Moneris`

[1]Full explanation on page 259

[2]For more information, see Appendix C (page 280).

[3]Available for Canadian integrations only.

[4]See "Definition of Request Fields" (page 258) for proper length definition

Sample Purchase Correction - CA

```
{
using System;
public class TestCanadaPurchaseCorrection
{
public static void Main(string[] args)
{
string store_id = "store5";
string api_token = "yesguy";
string order_id = "Test20150723031154";
string txn_number = "165745-0_10";
string crypt = "8";
string dynamic_descriptor = "123456";
string processing_country_code = "CA";
bool status_check = false;
PurchaseCorrection purchasecorrection = new
    PurchaseCorrection();
purchasecorrection.SetOrderId(order_id);
purchasecorrection.SetTxnNumber(txn_number);
purchasecorrection.SetCryptType(crypt);
purchasecorrection.SetDynamicDescriptor
    (dynamic_descriptor);
purchasecorrection.SetCustId("my customer
    id");
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(purchasecorrection);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
```

Sample Purchase Correction - US

```
{
using System;
public class TestUSAPurchaseCorrection
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string order_id = "Test20150723030805";
string txn_number = "856500-0_25";
string crypt = "7";
string dynamic_descriptor = "123456";
string custid = "mycustomerid";
string processing_country_code = "US";
bool status_check = false;
PurchaseCorrection purchasecorrection = new
    PurchaseCorrection();
purchasecorrection.SetOrderId(order_id);
purchasecorrection.SetTxnNumber(txn_number);
purchasecorrection.SetCryptType(crypt);
purchasecorrection.SetCustId(custid);
purchasecorrection.SetDynamicDescriptor
    (dynamic_descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(purchasecorrection);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
```

Sample Purchase Correction - CA

```
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("IsVisaDebit = " +
    receipt.GetIsVisaDebit());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Sample Purchase Correction - US

```
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
//Console.WriteLine("StatusCode = " +
    receipt.GetStatusCode());
//Console.WriteLine("StatusMessage = " +
    receipt.GetStatusMessage());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

5.8 Refund

To process this transaction, you need the order ID and transaction number from the original Completion,Purchase or Force Post transaction.

Refund transaction object definition

```
Refund refund = new Refund();
```

HttpsPostRequest object for Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(refund);
```

Refund transaction object values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 18: Refund transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`refund..SetOrderId(order_id);`
Amount	String	9-character decimal	`refund..SetAmount(amount);`

Table 18: Refund transaction object mandatory values (continued)

Value	Type	Limits	Set method
Transaction number	String	255-character variable character	`refund..SetTxnNumber(txn_number);`
E-Commerce indicator	String	1-character alphanumeric[1]	`refund..SetCryptType(crypt);`

Table 19: Refund transaction optional values

Value	Type	Limits	Set method
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Sample Refund - CA

```
namespace Moneris
{
using System;
public class TestCanadaRefund
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string amount = "1.00";
string crypt = "7";
string dynamic_descriptor = "123456";
string custid = "mycust9";
string order_id = "mvt3230836758";
string txn_number = "21964-0_10";
string processing_country_code = "CA";
bool status_check = false;
Refund refund = new Refund();
refund.SetTxnNumber(txn_number);
refund.SetOrderId(order_id);
refund.SetAmount(amount);
refund.SetCryptType(crypt);
refund.SetCustId(custid);
refund.SetDynamicDescriptor(dynamic_
    descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
```

Sample Refund - US

```
namespace Moneris
{
using System;
public class TestUSARefund
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string amount = "1.00";
string crypt = "7";
string dynamic_descriptor = "123456";
string custid = "mycustomerid";
string order_id = "Test20150723034412";
string txn_number = "856506-0_25";
string processing_country_code = "US";
bool status_check = false;
Refund refund = new Refund();
refund.SetOrderId(order_id);
refund.SetTxnNumber(txn_number);
refund.SetAmount(amount);
refund.SetCryptType(crypt);
refund.SetCustId(custid);
refund.SetDynamicDescriptor(dynamic_
    descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
```

[1]Full explanation on page 259

[2]For more information, see Appendix C (page 280).

Sample Refund - CA

```
mpgReq.SetTransaction(refund);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Sample Refund - US

```
mpgReq.SetTransaction(refund);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
//Console.WriteLine("StatusCode = " +
    receipt.GetStatusCode());
//Console.WriteLine("StatusMessage = " +
    receipt.GetStatusMessage());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

5.9 Independent Refund

Things to consider:

- Because of the potential for fraud, permission for this transaction is not granted to all accounts by default. If it is required for your business, it must be requested via your account manager.

Independent Refund transaction object definition

```
IndependentRefund indrefund = new IndependentRefund();
```

HttpsPostRequest object for Independent Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(indrefund);
```

Independent Refund transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 20: Independent Refund transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`indrefund..SetOrderId(order_id);`
Amount	String	9-character decimal	`indrefund..SetAmount(amount);`
Credit card number	String	20-character alphanumeric	`indrefund.SetPan(pan);`
Expiry date	String	4-character alphanumeric (YYMM format)	`indrefund..SetExpdate(expdate);`
E-Commerce indicator	String	1-character alphanumeric[1]	`indrefund..SetCryptType(crypt);`

Table 21: Independent Refund transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`indrefund.SetCustId(cust_id);`
Dynamic descriptor	String	20-character alphanumeric[2]	`indrefund..SetDynamicDescriptor (dynamic_descriptor);`
Status Check[3]	Boolean	true/false	`mpgReq.SetStatusCheck(status_ check);`
Commcard invoice[4]	String	17-character alphanumeric	`indrefund..SetCommcardInvoice (commcard_invoice);`

[1]Full explanation on page 259

[2]See "Definition of Request Fields" (page 258) for proper length definition

[3]For more information, see Appendix C (page 280).

[4]Available to US integrations only.

Table 21: Independent Refund transaction optional values (continued)

Value	Type	Limits	Set method
Commcard tax amount[1]	String	9-character decimal Must contain at least 3 digits, two of which must be penny values.	`indrefund..SetCommcardTaxAmount (commcard_tax_amount);`

Sample Independent Refund - CA	Sample Independent Refund - US

Sample Independent Refund - CA

```
namespace Moneris
{
using System;
public class TestCanadaIndependentRefund
{
public static void Main(string[] args)
{
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string store_id = "store5";
string api_token = "yesguy";
string cust_id = "my customer id";
string amount = "20.00";
string pan = "4242424242424242";
string expdate = "1901"; //YYMM
string crypt = "7";
string processing_country_code = "CA";
bool status_check = false;
IndependentRefund indrefund = new
    IndependentRefund();
indrefund.SetOrderId(order_id);
indrefund.SetCustId(cust_id);
indrefund.SetAmount(amount);
indrefund.SetPan(pan);
indrefund.SetExpdate(expdate);
indrefund.SetCryptType(crypt);
indrefund.SetDynamicDescriptor("123456");
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(indrefund);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
```

Sample Independent Refund - US

```
namespace Moneris
{
using System;
public class TestUSAIndependentRefund
{
public static void Main(string[] args)
{
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string store_id = "monusqa002";
string api_token = "qatoken";
string cust_id = "my customer id";
string amount = "20.00";
string pan = "4242424242424242";
string expdate = "1602"; //YYMM format
string crypt = "7";
string commcard_invoice = "INVC090";
string commcard_tax_amount = "1.00";
string processing_country_code = "US";
bool status_check = false;
IndependentRefund indrefund = new
    IndependentRefund();
indrefund.SetOrderId(order_id);
indrefund.SetCustId(cust_id);
indrefund.SetAmount(amount);
indrefund.SetPan(pan);
indrefund.SetExpdate(expdate);
indrefund.SetCryptType(crypt);
indrefund.SetCommcardInvoice(commcard_
    invoice);
indrefund.SetCommcardTaxAmount(commcard_tax_
    amount);
indrefund.SetDynamicDescriptor("123456");
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(indrefund);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
```

[1]Available to US integrations only.

Sample Independent Refund - CA

```
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("IsVisaDebit = " +
    receipt.GetIsVisaDebit());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Sample Independent Refund - US

```
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("IsVisaDebit = " +
    receipt.GetIsVisaDebit());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

5.10 Card Verification

Things to consider:

- This transaction type only applies to Visa and MasterCard transactions.
- This transaction is also known as an "account status inquiry".

Card Verification object definition

```
CardVerification cardVerification = new CardVerification();
```

HttpsPostRequest object for Card Verification transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(cardVerification);
```

Card Verification transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Note	AVD and CVD values are mandatory for US integrations only

Table 22: Card Verification transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`cardVerification..SetOrderId(order_id);`
Credit card number	String	20-character alphanumeric	`cardVerification.SetPan(pan);`
Expiry date	String	4-character alphanumeric (YYMM format)	`cardVerification..SetExpdate(expdate);`
E-commerce indicator	String	1-character alphanumeric[1]	`cardVerification..SetCryptType(crypt);`
AVS	Object	Not applicable. See Appendix E (page 288).	`cardVerification..SetAvsInfo(avsCheck);`
CVD	Object	Not applicable. See Appendix F (page 294).	`cardVerification..SetCvdInfo(cvdCheck);`

Sample Card Verification - CA

```
namespace Moneris
{
using System;
public class TestCanadaCardVerficiation
{
public static void Main(string[] args)
{
string store_id = "store5";
string api_token = "yesguy";
string order_id = "Test" +
```

Sample Card Verification - US

```
namespace Moneris
{
using System;
public class TestCardVerification
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string order_id = "Test" +
```

[1]Full explanation on page 259

Sample Card Verification - CA	Sample Card Verification - US

```
    DateTime.Now.ToString("yyyyMMddhhmmss");
string pan = "4242424242424242";
string expdate = "1901"; //YYMM format
string crypt = "7";
string processing_country_code = "CA";
bool status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
CvdInfo cvdCheck = new CvdInfo();
cvdCheck.SetCvdIndicator("1");
cvdCheck.SetCvdValue("099");
CardVerification cardVerification = new
    CardVerification();
cardVerification.SetOrderId(order_id);
cardVerification.SetPan(pan);
cardVerification.SetExpdate(expdate);
cardVerification.SetCryptType(crypt);
cardVerification.SetAvsInfo(avsCheck);
cardVerification.SetCvdInfo(cvdCheck);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(cardVerification);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
```
	```
	    DateTime.Now.ToString("yyyyMMddhhmmss");
	string cust_id = "customer1";
	string pan = "4242424242424242";
	string expiry_date = "1901"; //YYMM format
	string processing_country_code = "US";
	bool status_check = false;
	AvsInfo avsCheck = new AvsInfo();
	avsCheck.SetAvsStreetNumber("212");
	avsCheck.SetAvsStreetName("Payton Street");
	avsCheck.SetAvsZipCode("M1M1M1");
	CvdInfo cvdCheck = new CvdInfo();
	cvdCheck.SetCvdIndicator("1");
	cvdCheck.SetCvdValue("099");
	CardVerification cardVerification = new
	    CardVerification();
	cardVerification.SetOrderId(order_id);
	cardVerification.SetCustId(cust_id);
	cardVerification.SetPan(pan);
	cardVerification.SetExpdate(expiry_date);
	cardVerification.SetAvsInfo(avsCheck);
	cardVerification.SetCvdInfo(cvdCheck);
	HttpsPostRequest mpgReq = new HttpsPostRequest
	    ();
	mpgReq.SetProcCountryCode(processing_country_
	    code);
	mpgReq.SetTestMode(true); //false or comment
	    out this line for production transactions
	mpgReq.SetStoreId(store_id);
	mpgReq.SetApiToken(api_token);
	mpgReq.SetTransaction(cardVerification);
	mpgReq.SetStatusCheck(status_check);
	mpgReq.Send();
	try
	{
	Receipt receipt = mpgReq.GetReceipt();
	Console.WriteLine("CardType = " +
	    receipt.GetCardType());
	Console.WriteLine("TransAmount = " +
	    receipt.GetTransAmount());
	Console.WriteLine("TxnNumber = " +
	    receipt.GetTxnNumber());
	Console.WriteLine("ReceiptId = " +
	    receipt.GetReceiptId());
	Console.WriteLine("TransType = " +
	    receipt.GetTransType());
	Console.WriteLine("ReferenceNum = " +
	    receipt.GetReferenceNum());
	Console.WriteLine("ResponseCode = " +
	    receipt.GetResponseCode());
	Console.WriteLine("Message = " +
	    receipt.GetMessage());
	Console.WriteLine("AuthCode = " +
	    receipt.GetAuthCode());
	Console.WriteLine("Complete = " +
	    receipt.GetComplete());
	Console.WriteLine("TransDate = " +
	    receipt.GetTransDate());
	Console.WriteLine("TransTime = " +
	```

Sample Card Verification - CA	Sample Card Verification - US

```
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("IsVisaDebit = " +
    receipt.GetIsVisaDebit());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```
	```
	    receipt.GetTransTime());
	Console.WriteLine("Ticket = " +
	    receipt.GetTicket());
	Console.WriteLine("TimedOut = " +
	    receipt.GetTimedOut());
	//Console.WriteLine("CardLevelResult = " +
	    receipt.GetCardLevelResult());
	Console.ReadLine();
	}
	catch (Exception e)
	{
	Console.WriteLine(e);
	}
	}
	}
	}
	```

5.11 Batch Close

BatchClose transaction object definition

```
BatchClose batchclose = new BatchClose();
```

HttpsPostRequest object for Batch Close transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(batchclose);
```

Batch Close transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 23: BatchClose transaction object mandatory values

Value	Type	Limits	Set method
ECR (electronic cash register) number	String	No limit (value provided by Moneris)	`batchclose..SetEcrno(ecr_no);`

Sample Batch Close - CA	Sample Batch Close - US

```
namespace Moneris
{
using System;
public class TestCanadaBatchClose
{
```
	```
	namespace Moneris
	{
	using System;
	public class TestUSABatchClose
	{
	```

Sample Batch Close - CA

```
public static void Main(string[] args)
{
string store_id = "store5";
string api_token = "yesguy";
string ecr_no = "66013455"; //ecr within
    store
string processing_country_code = "CA";
bool status_check = false;
BatchClose batchclose = new BatchClose();
batchclose.SetEcrno(ecr_no);
HttpsPostRequest mpgReq = new
    HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(batchclose);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
if ((receipt.GetReceiptId()).Equals("Global
    Error Receipt"))
{
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = null");
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
```

Sample Batch Close - US

```
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string ecr_no = "64000003";//ecr within store
    "64000001"
string processing_country_code = "US";
bool status_check = false;
BatchClose batchclose = new BatchClose();
batchclose.SetEcrno(ecr_no);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(batchclose);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
if ((receipt.GetReceiptId()).Equals("Global
    Error Receipt"))
{
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("BankTotals = null");
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
}
```

Sample Batch Close - CA	Sample Batch Close - US

`}`
`else`
`{`
`foreach (string ecr in receipt.GetTerminalIDs`
`    ())`
`{`
`Console.WriteLine("ECR: " + ecr);`
`foreach (string cardType in`
`    receipt.GetCreditCards(ecr))`
`{`
`Console.WriteLine("\tCard Type: " +`
`    cardType);`
`Console.WriteLine("\t\tPurchase: Count = "`
`+ receipt.GetPurchaseCount(ecr, cardType)`
`+ " Amount = "`
`+ receipt.GetPurchaseAmount(ecr,`
`cardType));`
`Console.WriteLine("\t\tRefund: Count = "`
`+ receipt.GetRefundCount(ecr, cardType)`
`+ " Amount = "`
`+ receipt.GetRefundAmount(ecr, cardType));`
`Console.WriteLine("\t\tCorrection: Count = "`
`+ receipt.GetCorrectionCount(ecr, cardType)`
`+ " Amount = "`
`+ receipt.GetCorrectionAmount(ecr,`
`cardType));`
`}`
`}`
`}`
`Console.ReadLine();`
`}`
`catch (Exception e)`
`{`
`Console.WriteLine(e);`
`}`
`}`
`}`
`}`
	`else`
	`{`
	`foreach (string ecr in receipt.GetTerminalIDs`
	`    ())`
	`{`
	`Console.WriteLine("ECR: " + ecr);`
	`foreach (string cardType in`
	`    receipt.GetCreditCards(ecr))`
	`{`
	`Console.WriteLine("\tCard Type: " + cardType);`
	`Console.WriteLine("\t\tPurchase: Count = "`
	`+ receipt.GetPurchaseCount(ecr, cardType)`
	`+ " Amount = "`
	`+ receipt.GetPurchaseAmount(ecr,`
	`cardType));`
	`Console.WriteLine("\t\tRefund: Count = "`
	`+ receipt.GetRefundCount(ecr, cardType)`
	`+ " Amount = "`
	`+ receipt.GetRefundAmount(ecr, cardType));`
	`Console.WriteLine("\t\tCorrection: Count = "`
	`+ receipt.GetCorrectionCount(ecr, cardType)`
	`+ " Amount = "`
	`+ receipt.GetCorrectionAmount(ecr,`
	`cardType));`
	`}`
	`}`
	`}`
	`Console.ReadLine();`
	`}`
	`catch (Exception e)`
	`{`
	`Console.WriteLine(e);`
	`}`
	`}`
	`}`
	`}`

5.12 Open Totals

OpenTotals transaction object definition

```
OpenTotals opentotals = new OpenTotals();
```

HttpsPostRequest object for Open Totals transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(opentotals);
```

Open Totals transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 24: Open Totals transaction object mandatory values

Value	Type	Limits	Set method
ECR (electronic cash register) number	String	No limit (value provided by Moneris)	`opentotals..SetEcrno(ecr_no);`

Open Totals transaction optional values: None.

Sample Open Totals - CA	Sample Open Totals - US
(see below)	(see below)

Sample Open Totals - CA

```
namespace Moneris
{
using System;
public class TestCanadaOpenTotals
{
public static void Main(string[] args)
{
string store_id = "store5";
string api_token = "yesguy";
string ecr_no = "66013455";
//string ecr_no = "66013455";
string processing_country_code = "CA";
OpenTotals opentotals = new OpenTotals();
opentotals.SetEcrno(ecr_no);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(opentotals);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
if ((receipt.GetReceiptId()).Equals("Global
    Error Receipt") ||
receipt.GetReceiptId().Equals("") ||
receipt.GetReceiptId().Equals("null"))
{
Console.WriteLine("CardType = null ");
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = null");
```

Sample Open Totals - US

```
namespace Moneris
{
using System;
public class TestUSAOpenTotals
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string ecr_no = "64000003";
string processing_country_code = "US";
OpenTotals opentotals = new OpenTotals();
opentotals.SetEcrno(ecr_no);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(opentotals);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
if ((receipt.GetReceiptId()).Equals("Global
    Error Receipt"))
{
Console.WriteLine("CardType = " +
    receipt.GetCreditCards(ecr_no));
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = null");
Console.WriteLine("Message = " +
```

Sample Open Totals - CA

```
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
}
else
{
foreach (string ecr in receipt.GetTerminalIDs
    ())
{
Console.WriteLine("ECR: " + ecr);
foreach (string cardType in
    receipt.GetCreditCards(ecr))
{
Console.WriteLine("\tCard Type: " + cardType);
Console.WriteLine("\t\tPurchase: Count = "
+ receipt.GetPurchaseCount(ecr, cardType)
+ " Amount = "
+ receipt.GetPurchaseAmount(ecr,
cardType));
Console.WriteLine("\t\tRefund: Count = "
+ receipt.GetRefundCount(ecr, cardType)
+ " Amount = "
+ receipt.GetRefundAmount(ecr, cardType));
Console.WriteLine("\t\tCorrection: Count = "
+ receipt.GetCorrectionCount(ecr, cardType)
+ " Amount = "
+ receipt.GetCorrectionAmount(ecr,
cardType));
}
}
}
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Sample Open Totals - US

```
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());

}
else
{
foreach (string ecr in receipt.GetTerminalIDs
    ())
{
Console.WriteLine("ECR: " + ecr);
foreach (string cardType in
    receipt.GetCreditCards(ecr))
{
Console.WriteLine("\tCard Type: " + cardType);
Console.WriteLine("\t\tPurchase: Count = "
+ receipt.GetPurchaseCount(ecr, cardType)
+ " Amount = "
+ receipt.GetPurchaseAmount(ecr,
cardType));
Console.WriteLine("\t\tRefund: Count = "
+ receipt.GetRefundCount(ecr, cardType)
+ " Amount = "
+ receipt.GetRefundAmount(ecr, cardType));
Console.WriteLine("\t\tCorrection: Count = "
+ receipt.GetCorrectionCount(ecr, cardType)
+ " Amount = "
+ receipt.GetCorrectionAmount(ecr,
cardType));
}
}
}
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

6 MPI

The MonerisMPI accepts requests for Verified by Visa (VbV) and MasterCard Secure Code (MCSC). VBV and MCSC are programs based on the 3-D Secure Protocol to improve the security of online transactions. These programs involve authentication of the cardholder during an online e-commerce transaction. Authentication is based on the issuer’s selected method of authentication.

The following are examples of authentication methods:

- Risk-based authentication
- Dynamic passwords
- Static passwords.

Some of the benefits of these programs are reduced risk of fraudulent transactions and protection against chargebacks for certain fraudulent transactions. Enrollment is required to participate in the VbV and Secure Code programs. Merchants must contact the Moneris Sales/Support Helpdesk to enroll into these programs.

Any of the transaction objects that are defined in this section can be passed to the HttpsPostRequest connection object defined in Section 4 (page 24).

Additional eFraud features

To further decrease fraudulent activity, Moneris also recommends implementing the following features:

- AVS: Address Verification Service (page 288)
- CVD: Card Validation Digits (page 294).

6.1 Transaction Flow

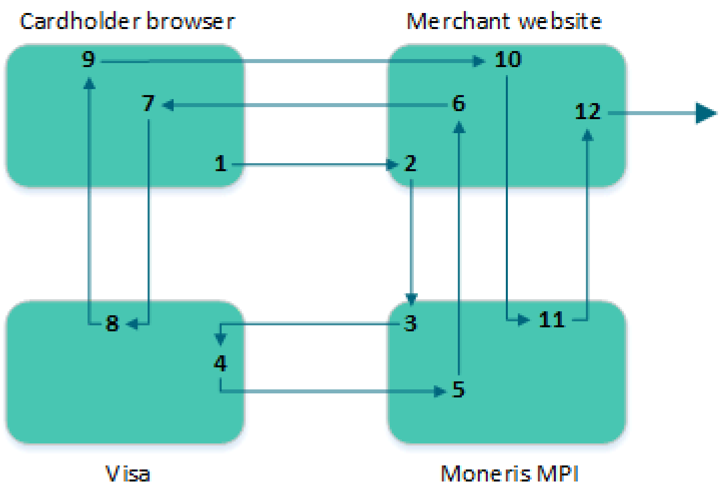


Figure 1: Transaction flow diagram

1. Cardholder enters the credit card number and submits the transaction information to the merchant.
2. Upon receiving the transaction request, the merchant calls the MonerisMPI API and passes a TXN type request. For sample code please refer to section 6.a(XREF TBD).
3. The Moneris MPI receives the request, authenticates the merchant and sends the transaction information to Visa or MasterCard.
4. Visa/MasterCard verifies that the card is enrolled and returns the issuer URL.
5. Moneris MPI receives the response from Visa or MasterCard and forwards the information to the merchant.
6. The MonerisMPI API installed at the merchant receives the response from the Moneris MPI.

   If the response is "Y" for enrolled, the merchant makes a call to the API, which opens a popup/inline window in the cardholder browser.

   If the response is “N” for not enrolled, a transaction could be sent to the processor identifying it as VBV/MCSC attempted with an ECI value of 6.

   If the response is “U” for unable to authenticate or the response times out, the transaction can be sent to the processor with an ECI value of 7. The merchant can then choose to continue with the transaction and be liable for a chargeback, or the merchant can choose to end the transaction.
7. The cardholder browser uses the URL that was returned from Visa/MasterCard via the merchant to communicate directly to the bank. The contents of the popup are loaded and the cardholder enters the PIN.
8. The information is submitted to the bank and authenticated. A response is then returned to the client browser.
9. The client browser receives the response from the bank, and forwards it to the merchant.
10. The merchant receives the response information from the cardholder browser, and passes an ACS request type to the Moneris MPI API.
11. Moneris MPI receives the ACS request and authenticates the information. The Moneris MPI then provides a CAVV value (getCavv()) to the merchant.

    If the getSuccess() of the response is “true”, the merchant may proceed with the cavv purchase or cavv preauth.

    If the getSuccess() of the response is “false” **and** the getMessage() is “N”, the transaction must be cancelled because the cardholder failed to authenticate.

    If the getSuccess() of the response is “false” **and** the getMessage is “U”, the transaction can be processed as a normal purchase or PreAuth; however in this case the merchant assumes liability of a chargeback.

    If the response times out, the transaction can be processed as a normal purchase or PreAuth; however in this case the merchant assumes liability of a chargeback.

    (The boolean logic was getting a bit complicated in the last option, so I broke thigns up a bit more. Let me know if I understood all the outcomes correctly.)
12. The merchant retrieves the CAVV value, and formats a cavv purchase or a cavv preauth request using the method that is normally used. As part of this transaction method, the merchant must pass the CAVV value.

    For more information on sending cavv-purchase and cavv-preauth, refer to the main API available from the MonerisDeveloper Portal (https://developer.moneris.com).

6.2 MPI Transactions

TXN
Sends the initial transaction data to the Moneris MPI to verify whether the card is enrolled.

The browser returns a PARes as well as a success field.

ACS
Passes the PARes (received in the response to the TXN transaction) to the Moneris MPI API.

Cavv Purchase
After receiving confirmation from the ACS transaction, this verifies funds on the customer's card, removes the funds and prepares them for deposit into the merchant's account.

Cavv Pre-Authorization
After receiving confirmation from the ACS transaction, this verifies and locks funds on the customer's credit card. The funds are locked for a specified amount of time based on the card issuer.

To retrieve the funds that have been locked by a Pre-Authorization transaction so that they may be settled in the merchant's account, a basic Completion transaction (page 36) must be performed. A PreAuthorization transaction may only be "completed" once.

6.2.1 VbV and MCSC Responses

For each transaction, a crypt type is sent to identify whether it is a VbV- or MCSC-authenticated transaction. Below are the tables defining the possible crypt types as well as the possible VARes and PARes responses.

Table 25: Crypt type definitions

Crypt type	Visa definition	MasterCard definition
5	• Fully authenticated • There is a liability shift, and the merchant is protected from chargebacks	• Fully authenticated • There is a liability shift, and the merchant is protected from chargebacks.
6	• VbV has been attempted • There is a liability shift, and the merchant is protected from certain chargebacks on fraudulent transactions	• MCSC has been attempted • There is a liability shift, and the merchant is protected from certain chargebacks on fraudulent transactions
7	• Non-VbV transaction • No liability shift • Merchant is not protected from chargebacks	• Non-MCSC transaction • No liability shift • Merchant is not protected from chargebacks

Table 26: VERes response definitions

VERes Response	Response Definition
N	The card/issuer is not enrolled. Sent as a normal Purchase/PreAuth transaction with a crypt type of 6.
U	The card type is not participating in VbV or MCSC. It could be corporate card or another card plan that Visa or MasterCard excludes. Proceed with a regular transaction with a crypt type of 7 or cancel the transaction.
Y	The card is enrolled. Proceed to create the VbV/MCSC inline window for cardholder authentication. Proceed to PARes for crypt type.

Table 27: PARes response definitions

PARes response	Response definition
A	Attempted to verify PIN, and will receive a CAVV. Send as a cavv_purchase/cavv_preAuth, which returns a crypt type of 6.
Y	Fully authenticated, and will receive a CAVV. Send as a cavv_purchase/cavv_preAuth which will return a crypt type of 5.
N	Failed to authenticate. No CAVV is returned. Cancel transaction. Merchant may proceed with a crypt type of 7 although this is mstrongly discouraged.

Table 28: CAVV transaction handling

Step 1: VERes Cardholder/issuer enrolled?	Step 2: PARes VbV/MCSC InLine window response	Step 3: Transaction Are you protected?
Y	Y	Send a CAVV transaction
Y	N	Cancel transaction. Authentication failed or high-risk transaction.
Y	A	Send a CAVV transaction
U	n/a	Send a regular transaction with a crypt type of 7
N	n/a	Send a regular transaction with a crypt type of 6

6.3 MpiTxn Request Transaction

MpiTxn transaction object definition

```
MpiTxn mpiTxn = new MpiTxn();
```

HttpsPostRequest object for MpiTxn transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(mpiTxn);
```

MpiTxn transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 29: MpiTxn transaction object mandatory values

Value	Type	Limits	Set method
XID	String	20-character alphanumeric	`mpiTxn..SetXid(xid);`
Credit card number	String	20-character numeric	`mpiTxn.SetPan(pan);`
Expiry date	String	4-character alphanumeric (YYMM format)	`mpiTxn..SetExpdate(expdate);`
Amount	String	9-character decimal Must contain at least 3 digits including two penny values.	`mpiTxn..SetAmount(amount);`
MD	String	1024-character alphanumeric	`mpiTxn..SetMD(MD);`
Merchant URL	String	TBD	`mpiTxn..SetMerchantUrl (merchantUrl);`
Accept	String	TBD	`mpiTxn..SetAccept(accept);`
User Agent	String	TBD	`mpiTxn..SetUserAgent(userAgent);`

Sample MpiTXN Request - CA	Sample MpiTXN Request - US

```
namespace Moneris
{
using System;
using System.Text;
public class TestCanadaMpiTxn
{
public static void Main(string[] args)
{
string store_id = "moneris";
string api_token = "hurgle";
string amount = "1.00";
Random r = new Random();
```
	```
	namespace Moneris
	{
	using System;
	using System.Text;
	public class TestUSAMpiTxn
	{
	public static void Main
	    (string[] args)
	{
	string store_id =
	    "monusqa002";
	```

Sample MpiTXN Request - CA

```
StringBuilder sb = new StringBuilder();
for(int i=0; i< 20; i++)
{
sb.Append(r.Next(0,9));
}
string xid = sb.ToString();
//string MD = xid + "mycardinfo" + amount;
string MD =
    "xid=99999999999999992464&amp;pan=4242424242424242&amp;expi
    ry=1511&amp;amount=1.00";
string merchantUrl = "https://YOUR_MPI_RESPONSE_URL";
string accept =
    "text/html,application/xhtml+xml,application/xml;q=0.9,imag
    e/webp,*/*;q=0.8";
string userAgent = "Mozilla/5.0 (Windows NT 6.1; WOW64)
    AppleWebKit/537.36 (KHTML, like Gecko) Chrome/43.0.2357.130
    Safari/537.36";
string processing_country_code = "CA";
string pan = "4242424242424242";
string expdate = "1511";
bool status_check = false;
MpiTxn mpiTxn = new MpiTxn();
mpiTxn.SetXid(xid);
mpiTxn.SetPan(pan);
mpiTxn.SetExpDate(expdate);
mpiTxn.SetAmount(amount);
mpiTxn.SetMD(MD);
mpiTxn.SetMerchantUrl(merchantUrl);
mpiTxn.SetAccept(accept);
mpiTxn.SetUserAgent(userAgent);
//************************OPTIONAL
    VARIABLES***************************
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for
    production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(mpiTxn);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
/********************** REQUEST ************************/
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("MpiMessage = " + receipt.GetMpiMessage());
Console.WriteLine("MpiSuccess = " + receipt.GetMpiSuccess());
if (receipt.GetMpiSuccess() == "true")
{
Console.WriteLine(receipt.GetInLineForm());
}
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
} // end TestResMpiTxn
```

Sample MpiTXN Request - US

```
string api_token =
    "qatoken";
string amount = "1.00";
Random r = new Random();
StringBuilder sb = new
    StringBuilder();
for(int i=0; i< 20; i++)
{
sb.Append(r.Next(0,9));
}
string xid = sb.ToString();
string MD = xid +
    "mycardinfo" + amount;
string merchantUrl =
    "www.mystoreurl.com";
string accept = "true";
string userAgent =
    "Mozilla";
string processing_country_
    code = "US";
string pan =
    "4242424242424242";
string expdate = "1905";
bool status_check = false;
MpiTxn mpiTxn = new MpiTxn
    ();
mpiTxn.SetXid(xid);
mpiTxn.SetPan(pan);
mpiTxn.SetExpDate(expdate);
mpiTxn.SetAmount(amount);
mpiTxn.SetMD(MD);
mpiTxn.SetMerchantUrl
    (merchantUrl);
mpiTxn.SetAccept(accept);
mpiTxn.SetUserAgent
    (userAgent);
//************************OP
    TIONAL
    VARIABLE
    S**********************
    ****
HttpsPostRequest mpgReq =
    new HttpsPostRequest();
mpgReq.SetProcCountryCode
    (processing_country_
    code);
mpgReq.SetTestMode(true);
    //false or comment out
    this line for production
    transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_
    token);
mpgReq.SetTransaction
    (mpiTxn);
mpgReq.SetStatusCheck
    (status_check);
mpgReq.Send();
```

Sample MpiTXN Request - CA	Sample MpiTXN Request - US

`}`	`/********************** REQUEST ************************ /`
	`try`
	`{`
	`Receipt receipt = mpgReq.GetReceipt();`
	`Console.WriteLine ("MpiMessage = " + receipt.GetMpiMessage ());`
	`Console.WriteLine ("MpiSuccess = " + receipt.GetMpiSuccess ());`
	`if (receipt.GetMpiSuccess() == "true")`
	`{`
	`Console.WriteLine (receipt.GetInLineForm ());`
	`}`
	`}`
	`catch (Exception e)`
	`{`
	`Console.WriteLine(e);`
	`}`
	`}`
	`} // end TestResMpiTxn`
	`}`

6.3.1 TXN Response and Creating the Popup

The TXN request returns a response with one of several possible values. The get Message method of the response object returns “Y”, “U”, or “N”.

N
Purchase or Pre-Authorization can be sent as a crypt type of 6 (attempted authentication).

Y
A call to the API to create the VBV form is made.

U
(Returned for non-participating cards such as corporate cards)

Merchant can send the transaction with crypt_type 7. However, the merchant is liable for chargebacks.

6.4 ResMpiTxn

ResMpiTxn transaction object definition

```
ResMpiTxn resMpiTxn = new ResMpiTxn();
```

HttpsPostRequest object for ResIndRefundCC transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetTransaction(resMpiTxn);
```

ResMpiTxn transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 30: ResMpiTxn transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alpha-numeric	`resMpiTxn..SetData(data_key);`
XID	String	TBD	`resMpiTxn..SetXid(xid);`
Amount	String	9-character decimal	`resMpiTxn..SetAmount(amount);`
MD	String		`resMpiTxn..SetMD(MD);`
Merchant URL	String		`resMpiTxn..SetMerchantUrl(merchantUrl);`
Accept	String		`resMpiTxn..SetAccept(accept);`
User Agent	String		`resMpiTxn..SetUserAgent(userAgent);`
Expiry date	String	4-character alphanumeric (YYMM format)	`resMpiTxn..SetExpdate(expdate);`

Table 31: ResMpiTxn transaction optional values

Value	Type	Limits	Set method
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Sample ResMpiTxn - CA	Sample ResMpiTxn - US

`namespace Moneris`
`{`
`using System;`
`using System.Text;`
`using System.Collections;`
`public class TestCanadaResMpiTxn`
`{`
`public static void Main(string[] args)`	

[1]For more information, see Appendix C (page 280).

Sample ResMpiTxn - CA	Sample ResMpiTxn - US
```
{
string store_id = "store5";
string api_token = "yesguy";
string data_key = "SzSrdoy0bt8UFXOtgS88wFAy7";
string amount = "1.00";
Random r = new Random();
StringBuilder sb = new StringBuilder();
for(int i=0; i< 20; i++)
{
sb.Append(r.Next(0,9));
}
string xid = sb.ToString();
string MD = xid + "mycardinfo" + amount;
string merchantUrl = "www.mystoreurl.com";
string accept = "true";
string userAgent = "Mozilla";
string processing_country_code = "CA";
bool status_check = false;
ResMpiTxn resMpiTxn = new ResMpiTxn();
resMpiTxn.SetData(data_key);
resMpiTxn.SetXid(xid);
resMpiTxn.SetAmount(amount);
resMpiTxn.SetMD(MD);
resMpiTxn.SetMerchantUrl(merchantUrl);
resMpiTxn.SetAccept(accept);
resMpiTxn.SetUserAgent(userAgent);
//**********************OPTIONAL
    VARIABLES*************************
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resMpiTxn);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
/********************** REQUEST
    ************************/
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("MpiMessage = " +
    receipt.GetMpiMessage());
Console.WriteLine("MpiSuccess = " +
    receipt.GetMpiSuccess());
if (receipt.GetMpiSuccess() == "true")
{
Console.WriteLine(receipt.GetInLineForm());
}
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
```

Sample ResMpiTxn - CA	Sample ResMpiTxn - US

`} // end TestResMpiTxn`
`}`	

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

6.5 MpiAcs Request Transaction

MpiAcs transaction object definition

```
MpiAcs resMpiAcs = new MpiAcs();
```

HttpsPostRequest object for MpiAcs transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resMpiAcs);
```

MpiAcs transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 32: MpiACS transaction object mandatory values

Value	Type	Limits	Set method
XID	String	20-character alphanumeric	`.SetXid(xid);`
Amount	String	9-character decimal Must contain at least 3 digits including two penny values.	`.SetAmount(amount);`
MD	String	1024-character alphanumeric	`.SetMD(MD);`
PARes	String	TBD	`resMpiAcs..SetPaRes(PaRes);`

Sample MpiACS Request - CA	Sample MpiACS Request - US

`namespace Moneris`
`{`
`using System;`
`public class TestCanadaMpiAcs`
`{`
`public static void Main(string[] args)`
`{`
`string store_id = "moneris";`
`string api_token = "hurgle";`
	`namespace Moneris`
	`{`
	`using System;`
	`public class TestUSAMpiAcs`
	`{`
	`public static void Main(string[] args)`
	`{`
	`string store_id = "monusqa006";`
	`string api_token = "qatoken";`

Sample MpiACS Request - CA

```
string amount = "1.00";
string xid = "12345678910111214005";
string MD = xid + "mycardinfo" + amount;
string PaRes = "PaRes Info";
string processing_country_code = "CA";
bool status_check = false;
MpiAcs resMpiAcs = new MpiAcs();
resMpiAcs.SetPaRes(PaRes);
resMpiAcs.SetMD(MD);
//************************OPTIONAL
    VARIABLES***************************
HttpsPostRequest mpgReq = new
    HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resMpiAcs);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
/********************** REQUEST
    ************************/
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("MpiMessage = " +
    receipt.GetMpiMessage());
Console.WriteLine("MpiSuccess = " +
    receipt.GetMpiSuccess());
if (receipt.GetMpiSuccess() == "true")
{
Console.WriteLine("Cavv = " +
    receipt.GetMpiCavv());
}
else
{
Console.WriteLine("Message = " +
    receipt.GetMessage());
}
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Sample MpiACS Request - US

```
string amount = "1.00";
string xid = "12345678910111214005";
string MD = xid + "mycardinfo" + amount;
String PaRes = "PaRes Info";
string processing_country_code = "US";
bool status_check = false;
MpiAcs resMpiAcs = new MpiAcs();
resMpiAcs.SetPaRes(PaRes);
resMpiAcs.SetMD(MD);
//************************OPTIONAL
    VARIABLES***************************
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resMpiAcs);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
/********************** REQUEST
    ************************/
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("MpiMessage = " +
    receipt.GetMpiMessage());
Console.WriteLine("MpiSuccess = " +
    receipt.GetMpiSuccess());
if (receipt.GetMpiSuccess() == "true")
{
Console.WriteLine("Cavv = " +
    receipt.GetMpiCavv());
}
else
{
Console.WriteLine("Message = " +
    receipt.GetMessage());
}
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

6.5.1 ACS Response and Forming a Transaction

The ACS response contains the CAVV value. This value is to be passed to the transaction engine using the cavv Purchase or cavv Pre-Authorization request. Please see the documentation provided by your payment solution.

Outlined below is how to send a transaction to Moneris Payment Gateway.

```
if ( mpiRes.getSuccess().equals("true") )
    {
    //Send transaction to host using CAVV purchase or CAVV preauth, refer to sample
    //code for Moneris Payment Gateway. Call mpiRes.getCavv() to obtain the CAVV value.
    //If you are using preauth/capture model, be sure to call getMessage() so the
    //value can be stored and used in the capture transaction after on to protect
    //your chargeback liability. (e.g. getMPIMessage()= A = crypt type of 6 for
    //follow on transaction and getMPIMessage() = Y = crypt type of 5 for follow on
    //transaction.
    }
else
    {
        if (mpiRes.getMessage().equals("N"))
        {
        //Do not send transaction as the cardholder failed authentication.
        }
        else
        {
        //Optional to send transaction using the mpg API. In this case merchant
        //assumes liability.
        }
    }
```

6.6 Cavv Purchase

CavvPurchase transaction object definition

```
CavvPurchase cavvPurchase = new CavvPurchase();
```

HttpsPostRequest object for Cavv Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(cavvPurchase);
```

Cavv Purchase transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 33: CavvPurchase transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`cavvPurchase..SetOrderId(order_id);`
Amount	String	9-character decimal	`cavvPurchase..SetAmount(amount);`
Credit card number	String	20-character alpha-numeric	`cavvPurchase.SetPan(pan);`
Expiry date	String	4-character alpha-numeric (YYMM format)	`cavvPurchase..SetExpdate(expdate);`
CAVV	String	50-character alpha-numeric	`cavvPurchase..SetCavv(cavv);`

Table 1: CavvPurchase transaction object optional values

Value	Type	Limits	Set Method
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`
Customer ID	String	50-character alphanumeric	`cavvPurchase.SetCustId(cust_id);`
Dynamic descriptor	String	20-character alphanumeric[2]	`cavvPurchase..SetDynamicDescriptor (dynamic_descriptor);`
Commercial card invoice[3]	String	17-character alphanumeric	`cavvPurchase..SetCommcardInvoice (commcard_invoice);`
Commercial card tax amount[4]	String	9-character decimal Must contain at least 3 digits, two of which must be penny values.	`cavvPurchase..SetCommcardTaxAmount (commcard_tax_amount);`
Customer information	Object	Not applicable. See Appendix E (page 288)	`cavvPurchase..SetCustInfo(cus-tomer);`
AVS[5]	Object	Not applicable. See Appendix E (page 288)	`cavvPurchase..SetAvsInfo (avsCheck);`
CVD[6]	Object	Not applicable. See Appendix F (page 294) .	`cavvPurchase..SetCvdInfo (cvdCheck);`
Convenience fee[7]	Object	Not applicable. See Appendix H (page 304).	`cavvPurchase.`

Sample CavvPurchase - CA	Sample CavvPurchase - US

`namespace Moneris`
`{`
`using System;`
`using System.Collections;`
`public class TestCanadaCavvPurchase`
`{`
`public static void Main(string[] args)`
`{`
`string store_id = "store5";`
	`namespace Moneris`
	`{`
	`using System;`
	`using System.Collections;`
	`public class TestUSACavvPurchase`
	`{`
	`public static void Main(string[] args)`
	`{`
	`string store_id = "monusqa002";`

[1]For more information, see Appendix C (page 280).
[2]See "Definition of Request Fields" (page 258) for proper length definition.
[3]Available to US integrations only.
[4]Available to US integrations only.
[5]Available to US integrations only.
[6]Available to US integrations only.
[7]Available to US integrations only.

Sample CavvPurchase - CA

```
string api_token = "yesguy";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "CUS887H67";
string amount = "10.42";
string pan = "4242424242424242";
string expdate = "1901"; //YYMM
string cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA=";
string dynamic_descriptor = "123456";
string processing_country_code = "CA";
bool status_check = false;
CavvPurchase cavvPurchase = new CavvPurchase
    ();
cavvPurchase.SetOrderId(order_id);
cavvPurchase.SetCustId(cust_id);
cavvPurchase.SetAmount(amount);
cavvPurchase.SetPan(pan);
cavvPurchase.SetExpdate(expdate);
cavvPurchase.SetCavv(cavv);
cavvPurchase.SetDynamicDescriptor(dynamic_
    descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(cavvPurchase);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
```

Sample CavvPurchase - US

```
string api_token = "qatoken";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "B_Urlac_54";
string amount = "10.42";
string pan = "4005554444444403";
string expdate = "1901"; //YYMM format
string cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA";
string commcard_invoice = "COINV982";
string commcard_tax_amount = "1.00";
string dynamic_descriptor = "my descriptor";
string processing_country_code = "US";
bool status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
CvdInfo cvdCheck = new CvdInfo();
cvdCheck.SetCvdIndicator("1");
cvdCheck.SetCvdValue("099");
CavvPurchase cavvPurchase = new CavvPurchase
    ();
cavvPurchase.SetOrderId(order_id);
cavvPurchase.SetCustId(cust_id);
cavvPurchase.SetAmount(amount);
cavvPurchase.SetPan(pan);
cavvPurchase.SetExpdate(expdate);
cavvPurchase.SetCavv(cavv);
cavvPurchase.SetDynamicDescriptor(dynamic_
    descriptor);
cavvPurchase.SetCommcardInvoice(commcard_
    invoice);
cavvPurchase.SetCommcardTaxAmount(commcard_
    tax_amount);
cavvPurchase.SetAvsInfo(avsCheck);
cavvPurchase.SetCvdInfo(cvdCheck);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(cavvPurchase);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
```

Sample CavvPurchase - CA	Sample CavvPurchase - US

```
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("CavvResultCode = " +
    receipt.GetCavvResultCode());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```
	```
	Console.WriteLine("TransType = " +
	    receipt.GetTransType());
	Console.WriteLine("ReferenceNum = " +
	    receipt.GetReferenceNum());
	Console.WriteLine("ResponseCode = " +
	    receipt.GetResponseCode());
	Console.WriteLine("ISO = " + receipt.GetISO
	    ());
	Console.WriteLine("BankTotals = " +
	    receipt.GetBankTotals());
	Console.WriteLine("Message = " +
	    receipt.GetMessage());
	Console.WriteLine("AuthCode = " +
	    receipt.GetAuthCode());
	Console.WriteLine("Complete = " +
	    receipt.GetComplete());
	Console.WriteLine("TransDate = " +
	    receipt.GetTransDate());
	Console.WriteLine("TransTime = " +
	    receipt.GetTransTime());
	Console.WriteLine("Ticket = " +
	    receipt.GetTicket());
	Console.WriteLine("TimedOut = " +
	    receipt.GetTimedOut());
	Console.WriteLine("Avs Response = " +
	    receipt.GetAvsResultCode());
	Console.WriteLine("Cvd Response = " +
	    receipt.GetCvdResultCode());
	//Console.WriteLine("CardLevelResult = " +
	    receipt.GetCardLevelResult());
	Console.WriteLine("CavvResultCode = " +
	    receipt.GetCavvResultCode());
	//Console.WriteLine("StatusCode = " +
	    receipt.GetStatusCode());
	//Console.WriteLine("StatusMessage = " +
	    receipt.GetStatusMessage());
	Console.ReadLine();
	}
	catch (Exception e)
	{
	Console.WriteLine(e);
	}
	}
	}
	}
	```

6.7 Cavv Pre-Authorization

CavvPre-Authorization transaction object definition

```
CavvPreAuth cavvPreauth = new CavvPreAuth();
```

HttpsPostRequest object for Cavv Pre-Authorization transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetTransaction(cavvPreauth);
```

Cavv Pre-Authorization transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 34: CavvPre-Authorization object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`cavvPreauth..SetOrderId(order_id);`
Amount	String	9-character decimal	`cavvPreauth..SetAmount(amount);`
Credit card number	String	20-character numeric	`cavvPreauth.SetPan(pan);`
Cardholder Authentic-ation Verification Value (CAVV)	String	50-character alphanumeric	`cavvPreauth..SetCavv(cavv);`
Expiry date	String	4-character numeric	`cavvPreauth..SetExpdate(expdate);`

Table 1: Cavv Pre-Authorization object optional values

Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`
Customer ID	String	50-character alphanumeric	`cavvPreauth.SetCustId(cust_id);`
Dynamic descriptor	String	20-character alphanumeric[2]	`cavvPreauth..SetDynamicDescriptor(dynamic_descriptor);`
AVS[3]	Object	Not applicable. See Appendix E (page 288).	`cavvPreauth..SetAvsInfo(avsCheck);`
CVD[4]	Object	Not applicable. See Appendix F (page 294) .	`cavvPreauth..SetCvdInfo(cvdCheck);`

Sample Cavv Pre-Authorization - CA	Sample Cavv Pre-Authorization - US

`namespace Moneris`
`{`
`using System;`
`using System.Collections;`
`public class TestCanadaCavvPreauth`
	`namespace Moneris`
	`{`
	`using System;`
	`using System.Collections;`
	`public class TestUSACavvPreauth`

[1]For more information, see Appendix C (page 280).

[2]See "Definition of Request Fields" (page 258) for proper length definition

[3]Available to US integrations only.

[4]Available to US integrations only.

Sample Cavv Pre-Authorization - CA

```
{
public static void Main(string[] args)
{
string store_id = "store5";
string api_token = "yesguy";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "CUS887H67";
string amount = "10.42";
string pan = "4242424242424242";
string expdate = "1911"; //YYMM format
string cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA=";
string dynamic_descriptor = "123456";
string processing_country_code = "CA";
bool status_check = false;
CavvPreAuth cavvPreauth = new CavvPreAuth();
cavvPreauth.SetOrderId(order_id);
cavvPreauth.SetCustId(cust_id);
cavvPreauth.SetAmount(amount);
cavvPreauth.SetPan(pan);
cavvPreauth.SetExpdate(expdate);
cavvPreauth.SetCavv(cavv);
cavvPreauth.SetDynamicDescriptor(dynamic_
    descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(cavvPreauth);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
```

Sample Cavv Pre-Authorization - US

```
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "B_Urlac_54";
string amount = "10.42";
string pan = "4242424242424242";
string expdate = "1902"; //YYMM format
string cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA";
string dynamic_descriptor = "123456";
string processing_country_code = "US";
bool status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
CvdInfo cvdCheck = new CvdInfo();
cvdCheck.SetCvdIndicator("1");
cvdCheck.SetCvdValue("099");
CavvPreAuth cavvPreauth = new CavvPreAuth();
cavvPreauth.SetOrderId(order_id);
cavvPreauth.SetCustId(cust_id);
cavvPreauth.SetAmount(amount);
cavvPreauth.SetPan(pan);
cavvPreauth.SetExpdate(expdate);
cavvPreauth.SetCavv(cavv);
cavvPreauth.SetDynamicDescriptor(dynamic_
    descriptor);
cavvPreauth.SetAvsInfo(avsCheck);
cavvPreauth.SetCvdInfo(cvdCheck);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(cavvPreauth);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
```

Sample Cavv Pre-Authorization - CA	Sample Cavv Pre-Authorization - US

`Console.WriteLine("AuthCode = " +`
`    receipt.GetAuthCode());`
`Console.WriteLine("Complete = " +`
`    receipt.GetComplete());`
`Console.WriteLine("TransDate = " +`
`    receipt.GetTransDate());`
`Console.WriteLine("TransTime = " +`
`    receipt.GetTransTime());`
`Console.WriteLine("Ticket = " +`
`    receipt.GetTicket());`
`Console.WriteLine("TimedOut = " +`
`    receipt.GetTimedOut());`
`Console.WriteLine("CavvResultCode = " +`
`    receipt.GetCavvResultCode());`
`Console.ReadLine();`
`}`
`catch (Exception e)`
`{`
`Console.WriteLine(e);`
`}`
`}`
`}`
`}`
	`    receipt.GetReferenceNum());`
	`Console.WriteLine("ResponseCode = " +`
	`    receipt.GetResponseCode());`
	`Console.WriteLine("ISO = " + receipt.GetISO`
	`    ());`
	`Console.WriteLine("BankTotals = " +`
	`    receipt.GetBankTotals());`
	`Console.WriteLine("Message = " +`
	`    receipt.GetMessage());`
	`Console.WriteLine("AuthCode = " +`
	`    receipt.GetAuthCode());`
	`Console.WriteLine("Complete = " +`
	`    receipt.GetComplete());`
	`Console.WriteLine("TransDate = " +`
	`    receipt.GetTransDate());`
	`Console.WriteLine("TransTime = " +`
	`    receipt.GetTransTime());`
	`Console.WriteLine("Ticket = " +`
	`    receipt.GetTicket());`
	`Console.WriteLine("TimedOut = " +`
	`    receipt.GetTimedOut());`
	`Console.WriteLine("Avs Response = " +`
	`    receipt.GetAvsResultCode());`
	`Console.WriteLine("Cvd Response = " +`
	`    receipt.GetCvdResultCode());`
	`//Console.WriteLine("CardLevelResult = " +`
	`    receipt.GetCardLevelResult());`
	`Console.WriteLine("CavvResultCode = " +`
	`    receipt.GetCavvResultCode());`
	`Console.ReadLine();`
	`}`
	`catch (Exception e)`
	`{`
	`Console.WriteLine(e);`
	`}`
	`}`
	`}`
	`}`

6.8 Cavv Result Codes

Table 35: CAVV result codes

Code	Message	Significance
0	CAVV authentication results invalid.	For this transaction, you may not receive protection from chargebacks as a result of using VBV because the CAVV was considered invalid at the time the financial transaction was processed. Check that you are following the VBV process correctly and passing the correct data in our transactions.
1	CAVV failed validation; authentication	Provided that you have implemented the VBV process correctly, the liability for this transaction should remain with the Issuer for chargeback reason codes covered by Verified by Visa.
2	CAVV passed validation; authentication	The CAVV was confirmed as part of the financial transaction. This transaction is a fully authenticated VBV transaction (ECI 5)
3	CAVV passed validation; attempt	The CAVV was confirmed as part of the financial transaction. This transaction is an attempted VBV transaction (ECI 6)
4	CAVV failed validation; attempt	Provided that you have implemented the VBV process correctly the liability for this transaction should remain with the Issuer for chargeback reason codes covered by Verified by Visa.
7	CAVV failed validation; attempt (US issued cards only)	Please check that you are following the VBV process correctly and passing the correct data in your transactions. Provided that you have implemented the VBV process correctly the liability for this transaction should be the same as an attempted transaction (ECI 6)
8	CAVV passed validation; attempt (US issued cards only	The CAVV was confirmed as part of the financial transaction. This transaction is an attempted VBV transaction (ECI 6)
9	= CAVV failed validation; attempt (US issued cards only)	Please check that you are following the VBV process correctly and passing the correct data in our transactions. Provided that you have implemented the VBV process correctly the liability for this transaction should be the same as an attempted transaction (ECI 6)
A	CAVV passed validation; attempt (US issued cards only)	The CAVV was confirmed as part of the financial transaction. This transaction is an attempted VBV transaction (ECI 6)
B	CAVV passed validation; information only, no liability shift	The CAVV was confirmed as part of the financial transaction. However, this transaction does not qualify for the liability shift. Treat this transaction the same as an ECI 7.

6.9 Vault Cavv Purchase

Vault Cavv Purchase transaction object definition

```
ResCavvPurchaseCC resCavvPurchaseCC = new ResCavvPurchaseCC();
```

HttpsPostRequest object for Vault Cavv Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resCavvPurchaseCC);
```

Vault Cavv Purchase transaction details

Table 36: Vault CavvPurchase transaction object mandatory values

Value	Type	Limits	Set method
Data Key	String	25-character alpha-numeric	`resCavvPurchaseCC.SetData(data_key);`
Order ID	String	50-character alpha-numeric	`resCavvPurchaseCC.SetOrderId(order_id);`
Amount	String	9-character decimal	`resCavvPurchaseCC..SetAmount(amount);`
Cardholder Authentication Verification Value (CAVV)	String	50-character alpha-numeric	`resCavvPurchaseCC..SetCavv(cavv);`

Table 37: Vault CavvPurchase transaction object optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`resCavvPurchaseCC.SetCustId(cust_id);`
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`
Expiry date	String	4-character alpha-numeric (YYMM format)	`resCavvPurchaseCC..SetExpdate(expdate);`

[1]For more information, see Appendix C (page 280).

6.10 Vault Cavv Pre-authorization

Vault Cavv Pre-authorization transaction object definition

```
ResCavvPreauthCC resCavvPreauthCC = new ResCavvPreauthCC();
```

HttpsPostRequest object for Vault Cavv Pre-authorization

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resCavvPreauthCC);
```

Vault Cavv Pre-authorization transaction details

Table 38: Vault Cavv Pre-Authorization object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`resCavvPreauthCC.SetOrderId (order_id);`
Amount	String	9-character decimal	`resCavvPreauthCC.SetAmount (amount);`
Credit card number	String	20-character numeric	`resCavvPreauthCC.SetPan (pan);`
CAVV	String	50-character alphanumeric	`resCavvPreauthCC.SetCavv (cavv);`
Expiry date	String	4-character numeric	`resCavvPreauthCC.SetExpdate (expdate);`

Table 39: Vault Cavv Pre-Authorization object optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`resCavvPreauthCC.SetCustId (cust_id);`
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck (status_check);`
Dynamic descriptor	String	20-character alphanumeric[2]	`resCavvPreauthCC.SetDy-namicDescriptor(dynamic_descriptor);`

[1]For more information, see Appendix C (page 280).

[2]See "Definition of Request Fields" (page 258) for proper length definition

Table 39: Vault Cavv Pre-Authorization object optional values

Value	Type	Limits	Set method
AVS[1]	Object	Not applicable. See Appendix E (page 288).	`resCavvPreauthCC.SetAvsInfo (avsCheck);`
CVD[2]	Object	Not applicable. See Appendix F (page 294) .	`resCavvPreauthCC.SetCvdInfo (cvdCheck);`

[1]Available to US integrations only.

[2]Available to US integrations only.

7 INTERAC® Online Payment

The INTERAC® Online Payment (IOP) method offers cardholders the ability to pay using online banking. This payment method can be combined with the Moneris Payment Gateway API solution to allow online payments using credit and debit cards.

INTERAC® Online Payment transactions via the API require two steps:

1. The cardholder guarantees the funds for the purchase amount using their online banking process.
2. The merchant confirms the payment by sending an INTERAC® Online Payment purchase request to Moneris using the API.

Any of the transaction objects that are defined in this section can be passed to the HttpsPostRequest connection object defined in Section 4 (page 24).

INTERAC® Online Paymenttransactions are available to **Canadian integrations** only.

7.1 Other Documents and References

INTERAC® Online Payment is offered by Acxsys Corporation, which is also a licensed user of the *Interac* logo. Refer to the following documentation and websites for additional details.

INTERAC® Online PaymentMerchant Guideline

Visit the Moneris Developer Portal (https://developer.moneris.com) to access the latest documentation and downloads.

This details the requirements for each page consumers visit on a typical INTERAC® Online Payment merchant website. It also details the requirements that can be displayed on any page (that is, requirements that are not page-specific).

Logos

Visit the Moneris Developer Portal (https://developer.moneris.com) to access the logos and downloads.

7.2 Website and Certification Requirements

7.2.1 Things to provide to Moneris

Refer to the Merchant Guidelines referenced in Section 7.1 for instructions on proper use of logos and the term "INTERAC® Online Payment". You need to provide Moneris with the following registration information:

- Merchant logo to be displayed on the INTERAC® Online Payment Gateway page
  - In both French and English
  - 120 × 30 pixels
  - Only PNG format is supported.
- Merchant business name
  - In both English and French
  - Maximum 30 characters.
- List of all referrer URLs. That is, URLs from which the customer may be redirected to the INTERAC® Online Payment gateway.
- List of all URLs that may appear in the IDEBIT_FUNDEDURL field of the https form POST to the INTERAC® Online Payment Gateway.
- List of all URLs that may appear in the IDEBIT_NOTFUNDEDURL field of the https form POST to the INTERAC® Online Payment Gateway.

Note that if your test and production environments are different, provide the above information for both environments.

7.2.2 Certification process

Test cases

All independent merchants and third-party service/shopping cart providers must pass the certification process by conducting all the test cases outlined in Appendix K (page 309) and "Third-Party Service Provider Checklists for INTERAC® Online Payment Certification Testing" on page 313 respectively. This is required after you have completed all of your testing.

Any major changes to your website after certification (with respect to the INTERAC® Online Payment functionality) require the site to be re-certified by completing the test cases again.

Appendix N (page 321) is the Certification Test Case Detail showing all the information and requirements for each test case.

Screenshots

You must provide Moneris with screenshots of your check-out process showing examples of approved and declined transactions using the INTERAC® Online Payment service.

Checklists

To consistently portray the INTERAC Online service as a secure payment option, you must complete the respective Merchant Requirement checklist inAppendix K (page 309) or Appendix L (page 313)accordingly. The detailed descriptions of the requirements in these checklists can be found in the INTERAC® Online Payment Merchant Guidelines document referred to in 7.1 (page 82). If any item does not apply, mark it as "N/A".

After completion, fax or email the results to the Moneris Integration Support help desk for review before implementing the change into the production environment.

7.2.3 Client Requirements

Checklists

As a merchant using an INTERAC® Online Payment-certified third-party solution, your clients must complete the Merchant Checklists for INTERAC® Online Payment Certification form (Appendix M, page 318). They will **not** be required to complete any of the test cases.

Your clients must also complete the Merchant Requirement checklist (Appendix M, page 318). Ensure that your product documentation properly instructs your clients to fax or email the results to the Moneris Integration Support helpdesk for registration purposes.

Screenshots

Your clients must provide Moneris with screenshots of their check-out process that show examples of approved and declined transactions using INTERAC® Online Payment.

7.2.4 Delays

Note that merchants that fall under the following category codes listed in Table 40 may experience delays in the certification or registration process of up to 7 days.

Table 40: Category codes that might introduce certification/registration delays

Category code	Merchant type/name
4812	Telecommunication equipment including telephone sales
4829	Money transfer—merchant
5045	Computers, computer peripheral equipment, software
5732	Electronic sales
6012	Financial institution—merchandise and services
6051	Quasi cash—merchant
6530	Remote stored value load—merchant
6531	Payment service provider—money transfer for a purchase
6533	Payment service provider—merchant—payment transaction

7.3 Transaction Flow

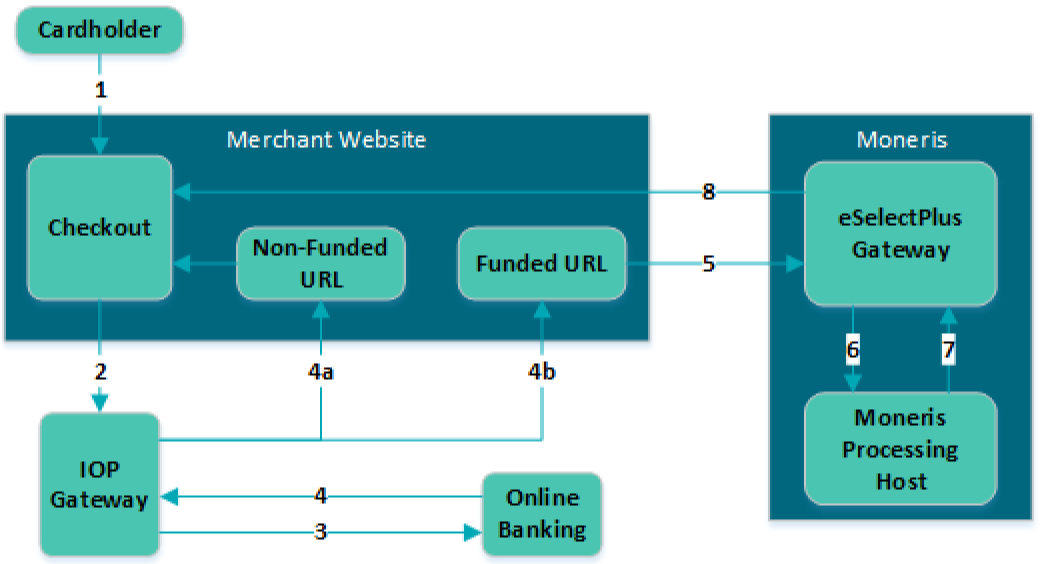


Figure 2: INTERAC® Online Payment transaction flow diagram

1. Customer selects the INTERAC® Online Payment option on the merchant's web store.
2. Merchant redirects the customer to the IOP gateway to select a financial institution (issuer) of choice. This step involves form-posting the following required variables over the HTTPS protocol:
   - IDEBIT_MERCHNUM
   - IDEBIT_AMOUNT[1]
   - IDEBIT_CURRENCY
   - IDEBIT_FUNDEDURL
   - IDEBIT_NOTFUNDEDURL
   - IDEBIT_MERCHLANG
   - IDEBIT_VERSIONIDEBIT_TERMID - optional
   - IDEBIT_INVOICE - optional
   - IDEBIT_MERCHDATA - optional
3. Customer selects an issuer, and is directed to the online banking site. Customer completes the online banking process and guarantees the funds for the purchase.
4. Depending on the results of step 3, the issuer re-directs the customer through the IOP Gateway to either the merchant's non-funded URL (4a) or funded URL (4b). Both URLs can appear on the same page. The funded/non-funded URLs must validate the variables posted back according to 7.7 (page 91) before continuing.

   Table 41 shows the variables that are posted back in the re-direction.

   If the customer is directed to the non-funded URL, return to step 2 and ask for another means of payment.

   If the customer is directed to the funded URL, continue to the next step.

[1]This value is expressed in cents. Therefore, $1 is input as 100

5. Merchant sends an INTERAC® Online Payment purchase request to Moneris Payment Gateway while displaying the "Please wait...." message to the customer. This should be done within 30 minutes of receiving the response in step 4.
6. Moneris' processing host sends a request for payment confirmation to the issuer.
7. The issuer sends a response (either approved or declined) to Moneris host.
8. Moneris Payment Gateway relays the response back to the merchant. If the payment was approved, the merchant fulfills the order.

Table 41: Funded and non-funded URL variables

To funded URL only	To funded and non-funded URL
IDEBIT_TRACK2	IDEBIT_VERSION
IDEBIT_ISSCONF	IDEBIT_ISSLANG
IDEBIT_ISSNAME	IDEBIT_TERMID (optional)
	IDEBIT_INVOICE (optional)
	IDEBIT_MERCHDATA (optional)

7.4 Sending an INTERAC® Online Payment Purchase Transaction

7.4.1 Fund-Guarantee Request

After choosing to pay by INTERAC® Online Payment, the customer is redirected using an HTML form post to the INTERAC® Online PaymentGateway page. Below is a sample code that is used to post the request to the Gateway.

```
<form action='from Section 9' method='post'>
<input type='text' name='IDEBIT_INVOICE' value=’your unique invoice number'>
    <input type='text' name='IDEBIT_AMOUNT' value='100'> <!— ($1.00) use cent values instead of
        dollar.cent format ->
<input type='text' name='IDEBIT_MERCHNUM' value='from Moneris Solutions’>
<input type='text' name='IDEBIT_CURRENCY' value='CA'>
<input type='text' name='IDEBIT_FUNDEDURL' value='your funded url'>
<input type='text' name='IDEBIT_NOTFUNDEDURL' value='your not funded url'>
<input type='text' name='IDEBIT_ISSLANG' value='en'>
<input type='text' name='IDEBIT_VERSION' value='1'>
<input type="submit" name="Submit" value="Submit to Gateway">
</form>
```

7.4.2 Online Banking Response and Fund-Confirmation Request

The response variables are posted back in an HTML form to either the funded or non-funded URL that was provided to INTERAC®.

The following variables must be validated (7.7, page 91):

- IDEBIT_TRACK2
- IDEBIT_ISSCONF
- IDEBIT_ISSNAME
- IDEBIT_VERSION

- IDEBIT_ISSLANG
- IDEBIT_INVOICE

Note that IDEBIT_ISSCONF and IDEBIT_ISSNAME must be displayed on the client's receipt that is generated by the merchant.

After validation, IDEBIT_TRACK2 is used to form an IDebitPurchase transaction that is sent to Moneris Payment Gateway to confirm the fund.

If the validation fails, redirect the client to the main page and ask for a different means of payment.

If the validation passes, an IDebitPurchase transaction can be sent to Moneris Payment Gateway.

7.5 INTERAC® Online Payment Purchase

IDebitPurchase transaction object definition

```
IDebitPurchase IOP_Txn = new IDebitPurchase();
```

HttpsPostRequest object for INTERAC® Online Payment Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(IOP_Txn);
```

INTERAC® Online Payment Purchase transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 42: IDebitPurchase transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`IOP_Txn.SetOrderId(order_id);`
Amount	String	9-character decimal	`IOP_Txn.SetAmount(amount);`
Track2 data	String	40-character alphanumeric	`IOP_Txn.SetTrack2(track2);`

Table 43: INTERAC® Online Payment Purchase transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`IOP_Txn.SetCustId(cust_id);`
Dynamic descriptor	String	20-character alphanumeric[1]	`IOP_Txn.SetDynamicDescriptor (dynamic_descriptor);`
Customer information	Object	Not applicable. See Section Appendix D (page 282).	`IOP_Txn.SetCustInfo(customer);`

[1]See "Definition of Request Fields" (page 258) for proper length definition

Sample IDebitPurchase - CA

```
namespace Moneris
{
using System;
public class TestCanadaIDebitPurchase
{
public static void Main(string[] args)
{
string store_id = "store5";
string api_token = "yesguy";
string order_id = "Test" + DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "Lance_Briggs_55";
string amount = "5.00";
string track2 = "5268051119993326=0609AAAAAAAAAAAAA000";
string processing_country_code = "CA";
bool status_check = false;
/********************* Billing/Shipping Variables ***************************/
string first_name = "Bob";
string last_name = "Smith";
string company_name = "ProLine Inc.";
string address = "623 Bears Ave";
string city = "Chicago";
string province = "Illinois";
string postal_code = "M1M2M1";
string country = "Canada";
string phone = "777-999-7777";
string fax = "777-999-7778";
string tax1 = "10.00";
string tax2 = "5.78";
string tax3 = "4.56";
string shipping_cost = "10.00";
/********************* Order Line Item Variables ****************************/
string[] item_description = new string[] { "Chicago Bears Helmet", "Soldier Field Poster" };
string[] item_quantity = new string[] { "1", "1" };
string[] item_product_code = new string[] { "CB3450", "SF998S" };
string[] item_extended_amount = new string[] { "150.00", "19.79" };
/********************** Customer Information Object *************************/
CustInfo customer = new CustInfo();
/********************** Set Customer Billing Information **********************/
customer.SetBilling(first_name, last_name, company_name, address, city,
province, postal_code, country, phone, fax, tax1, tax2,
tax3, shipping_cost);
/******************** Set Customer Shipping Information **********************/
customer.SetShipping(first_name, last_name, company_name, address, city,
province, postal_code, country, phone, fax, tax1, tax2,
tax3, shipping_cost);
/***************************** Order Line Items *****************************/
customer.SetItem(item_description[0], item_quantity[0],
item_product_code[0], item_extended_amount[0]);
customer.SetItem(item_description[1], item_quantity[1],
item_product_code[1], item_extended_amount[1]);
/************************* Request ************************/
IDebitPurchase IOP_Txn = new IDebitPurchase();
IOP_Txn.SetOrderId(order_id);
IOP_Txn.SetCustId(cust_id);
IOP_Txn.SetAmount(amount);
IOP_Txn.SetIdebitTrack2(track2);
IOP_Txn.SetCustInfo(customer);
//IOP_Txn.SetDynamicDescriptor("dynamicdescriptor1");
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

Sample IDebitPurchase - CA

```
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(IOP_Txn);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

7.6 INTERAC® Online Payment Refund

To process this transaction, you need the order ID and transaction number from the original INTERAC® Online Payment Purchase transaction.

IDebitRefund transaction object definition

```
IDebitRefund refund = new IDebitRefund();
```

HttpsPostRequest object for Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.SetTransaction(refund);
```

Refund transaction object values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 44: IDebitRefund transaction object mandatory variables

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`refund..SetOrderId(order_id);`
Amount	String	9-character decimal	`refund..SetAmount(amount);`
Transaction number	String	255-character varchar	`refund..SetTxnNumber(txn_number);`

Table 45: INTERAC® Online Payment Refund transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`refund.SetCustId(cust_id);`
Status Check	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Sample code

Sample IDebitRefund - CA

```
namespace Moneris
{
using System;
public class TestCanadaIDebitRefund
{
public static void Main(string[] args)
{
string store_id = "store5";
string api_token = "yesguy";
string order_id = "Test20150625014816";
string amount = "5.00";
string txn_number = "113524-0_10";
string processing_country_code = "CA";
bool status_check = false;
IDebitRefund refund = new IDebitRefund();
refund.SetOrderId(order_id);
refund.SetAmount(amount);
refund.SetTxnNumber(txn_number);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(refund);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("TransType = " + receipt.GetTransType());
```

Sample IDebitRefund - CA

```
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

7.7 INTERAC® Online Payment Field Definitions

Table 46: Field Definitions

Value	Size[1]	Limits
	Description	
IDEBIT_ MERCHNUM	5-14	Numbers and uppercase letters
	This field is provided by Moneris. For example, 0003MONMPGXXXX.	
IDEBIT_TERMID	8	Numbers and uppercase letters
	Optional field	
IDEBIT_ AMOUNT	1-12	Numbers
	Amount expressed in cents (for example, 1245 for $12.45) to charge to the card.	
IDEBIT_ CURRENCY	3	"CAD" or "USD"
	National currency of the transaction.	

[1]Expressed in characters

Table 46: Field Definitions (continued)

Value	Size[1]	Limits
	Description	
IDEBIT_INVOICE	1-20	ISO-8859-1 encoded characters restricted to: • Uppercase and lowercase • Numbers • À Á Â Ä È É Ê Ë Î Ï Ô Ù Û Ü Ç à á â ä è é ê ë î ï ô ù û ü ÿ ç • Spaces • # $. , - / = ? @ '
	Optional field Can be the Order ID when used with Moneris Payment Gateway fund confirmation transactions.	
IDEBIT_ MERCHDATA	1024	ISO-8859-1 restricted to single-byte codes, hex 20 to 7E (consistent with US-ASCII and ISO-8859-1 Latin-1). Note that the following character combinations may not be accepted in the IDEBIT_MERCHDATA field: • "/..", "/%2E.", "/.%2E", "/%2E%2E", "\\%2E%2E", "\\%2E.", "\\.%2E", "\\%2E%2E", "&#", "<", "%3C", ">", "%3E"
	Free form data provided by the merchant that will be passed back unchanged to the merchant once the payment has been guaranteed in online banking. This may be used to identify the customer, session or both.	
IDEBIT_ FUNDEDURL	1024	ISO-8859-1 restricted to single-byte codes, restricted to: • Uppercase and lowercase letters • Numbers • ; / ? : @ & = + $, - _ . ! ~ * ' ( ) %
	Https address to which the issuer will redirect cardholders after guaranteeing the fund through online banking.	
IDEBIT_ NOTFUNDEDURL	1024	ISO-8859-1, restricted to single-byte codes, restricted to: • Uppercase and lowercase letters • Numbers • ; / ? : @ & = + $, - _ . ! ~ * ' ( ) %
	Https address to which the issuer redirects cardholders after failing or canceling the online banking process.	

[1]Expressed in characters

Table 46: Field Definitions (continued)

Value	Size[1]	Limits
	Description	
IDEBIT_ MERCHLANG	2	“en” or “fr”
	Customer's current language at merchant.	
IDEBIT_VERSION	3	Numbers
	Initially, the value is 1.	
IDEBIT_ISSLANG	2	“en” or “fr”
	Customer’s current language at issuer.	
IDEBIT_TRACK2	37	ISO-8859-1 (restricted to single-byte codes), hex 20 to 7E (consistent with US-ASCII and ISO-8859-1 Latin-1)
	Value returned by the issuer. It includes the PAN, expiry date, and transaction ID.	
IDEBIT_ISSCONF	15	ISO-8859-1 encoded characters restricted to: • Uppercase and lowercase letters • Numbers • À Á Â Ä È É Ê Ë Î Ï Ô Ù Û Ü Ç à á â ä è é ê ë î ï ô ù û ü ÿ ç • Spaces • # $. , - / = ? @ '
	Confirmation number returned from the issuer to be displayed on the merchant’s confirmation page and on the receipt.	
IDEBIT_ ISSNAME	30	ISO-8859-1 encoded characters restricted to: • Uppercase and lowercase letters • Numbers • À Á Â Ä È É Ê Ë Î Ï Ô Ù Û Ü Ç à á â ä è é ê ë î ï ô ù û ü ÿ ç • Spaces • # $. , - / = ? @ • '
	Issuer name to be displayed on the merchant’s confirmation page and on the receipt.	

[1]Expressed in characters

8 ACH Transaction Set

Automated Clearing House (ACH) is a flexible low-cost way to automatically collect payments and fees directly from a customer's bank account. ACH transactions allow the customer to submit bank account information to/from which funds can be credited/debited.

Any of the transaction objects that are defined in this section can be passed to the HttpsPostRequest connection object defined in Section 4 (page 24).

ACH transactions are available to **US integrations** only.

8.1 ACH Transaction Definitions

ACH Debit
Verifies and collects the customer's bank account information, removes the funds directly from the bank account and prepares them for deposit into the merchant's account.

ACH Reversal
Refunds the **full** amount of an ACH Debit transaction.

This transaction can only be performed against an ACH Debit transaction that was performed within the last 3 months.

ACH Credit
Verifies and collects the customer's bank account information, and transfers merchant funds directly to the customer.

ACH Financial Inquiry (FI)
Verifies which financial institution a routing number belongs to.

Can also be used to verify whether the routing number is valid before submitting an ACH Debit transaction or an ACH Credit transaction.

8.2 ACHInfo Object

The ACHDebit and ACHCredit transaction objects have the ACHInfo object as a property. Therefore, before invoking the connection object's setTransaction method, you need to pass the ACHInfo object to the ACH transaction object by using its setAchInfo method.

ACH Info object definition

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

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

Note	AchInfo fields are **not** used for any type of address verification or fraud check.

Table 47: ACHInfo object mandatory arguments

Value	Type	Limits	Sample Code Variable Name
	Description (if any)		
Sec code	String	3-character alphanumeric	
	See " ACH SEC Codes and Process Flow" on the facing page.		
Customer's first name	String	50-character alphanumeric	
Customer's last name	String	50-character alphanumeric	
Customer's address 1	String	50-character alphanumeric	
Customer's address 2	String	50-character alphanumeric	
Customer's city	String	50-character alphanumeric	
Customer's state	String	2-character alphanumeric	
Customer's zip code	String	15-character alphanumeric	
Check routing number	String	9-character numeric	
	First number in the MICR line at the bottom of a check. It always begins with 0, 1, 2 or 3.		
Account number	String	50-character numeric	
	May appear before or after the check number in the MICR line at the bottom of the check.		
Check number	String	16-character numeric	
	Sequential number that appears in both the MICR line at the bottom of the check and in the upper right corner.		
Account type	String	savings/checking	
	Identifies the type of bank account. This field is case-sensitive.		

Sample ACHInfo object definition (using ACHDebit as the transaction)

```
//Declaration and initialization of variables removed for space.

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


ACHDebit achdebit = new ACHDebit();
achdebit.setAchInfo(achinfo);

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setTransaction(achdebit);
mpgReq.send();
```

8.2.1 ACH SEC Codes and Process Flow

Table 48: ACH SEC codes

Check	Code	Description
Not present	PPD*	**Pre-arranged payment and deposit** Debit (sale): Consumer grants the merchant the right to initiate either a one-time or recurring charge(s) to an account as bills become due. Credit (refund): Transfers funds into a consumer's bank account. The funds being deposited can represent a variety of financial transactions, such as payroll, interest, pension and so on.
	CCD*	**Cash concentration or disbursement** Debit (sale): Client grants the merchant the right to initiate a one-time or recurring charge(s) to a business bank account. Credit (Refund): Transfers funds to a client's business bank account.
	WEB	**Internet-initiated entry** Debit (Sale): A debit entry to a consumer's bank account initiated by a merchant. The consumer's authorization is obtained via the Internet. Credit (Refund): N/A.

* Only PPD and CCD apply to ACH Credit transactions.

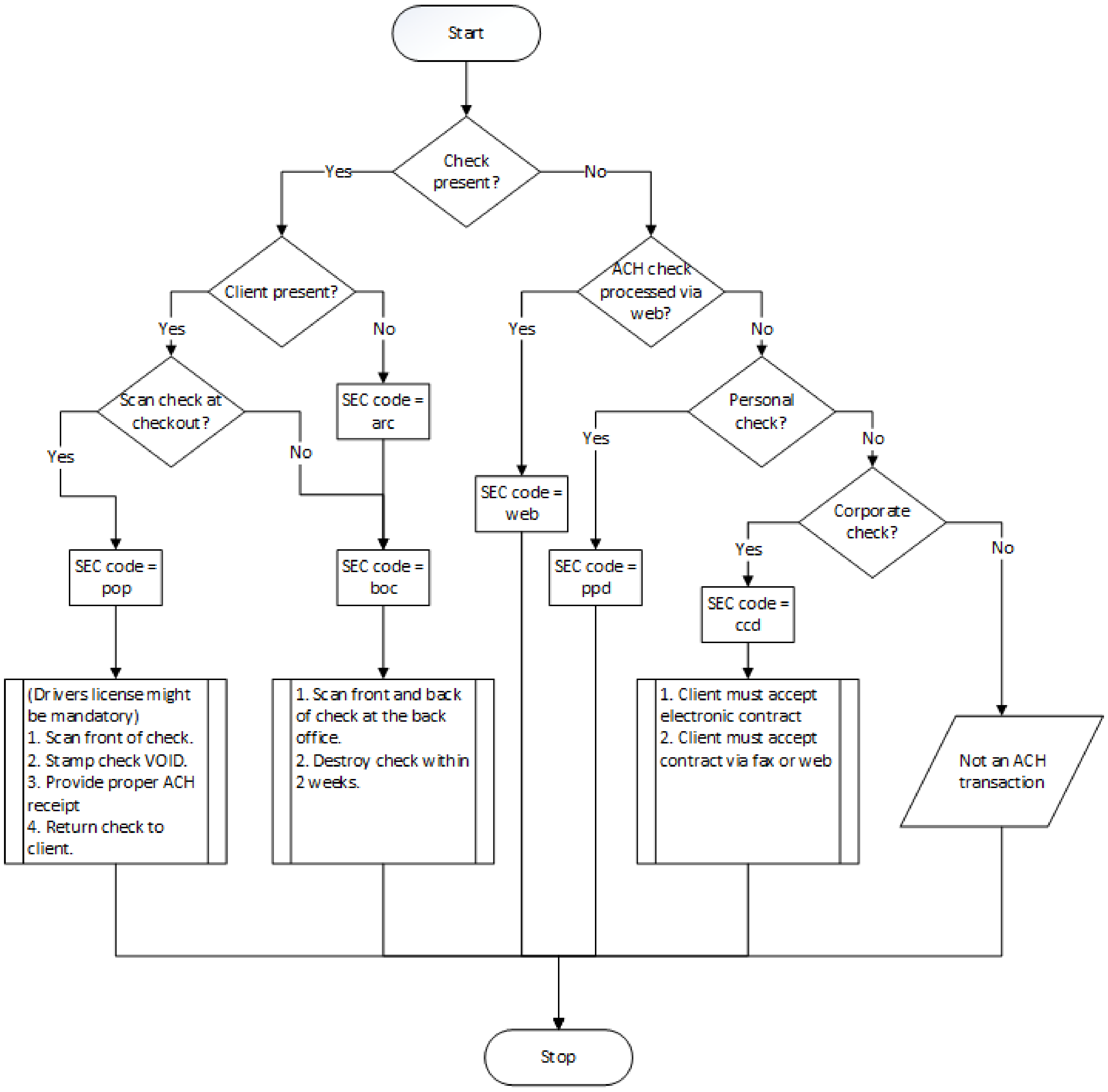

Start
Check present?
Yes
No
Client present?
ACH check processed via web?
Yes
No
Yes
No
Scan check at checkout?
SEC code = arc
Personal check?
Yes
No
Yes
No
SEC code = web
Corporate check?
Yes
No
SEC code = pop
SEC code = boc
SEC code = ppd
SEC code = ccd
(Drivers license might be mandatory)
1. Scan front of check.
2. Stamp check VOID.
3. Provide proper ACH receipt
4. Return check to client.
1. Scan front and back of check at the back office.
2. Destroy check within 2 weeks.
1. Client must accept electronic contract
2. Client must accept contract via fax or web
Not an ACH transaction
Stop

Figure 3: Process flow for ACH transactions

8.3 ACH Debit

ACH Debit transaction object definition

```
ACHDebit achdebit = new ACHDebit();
```

HttpsPostRequest object for ACH Debit transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(achdebit);
```

ACHDebit transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 49: ACH Debit transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`achdebit.SetOrderId(order_id);`
Amount	String	9-character decimal	`achdebit.SetAmount(amount);`
ACH Info	Object	See ACH info object tables below for a list of variables	`achdebit.SetAchInfo(achinfo);`

Table 50: ACH Debit transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`achdebit`
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck (status_check);`
Customer information	Object	Not applicable. See Section Appendix D (page 282).	`achdebit.SetCustInfo(cus-tomer);`
Convenience fee	Object	Not applicable. See Appendix H (page 304).	`achdebitCODE TO COME`
Recurring billing[2]	Object	Not applicable. See Section Appendix G (page 297).	`achdebit.SetRecur(recur-ring_cycle);`

[1]For more information, see Appendix C (page 280).

[2]Recurring Billing fields are only available to SEC codes `ppd`, `ccd` and `web`.

Table 1: ACH Info object mandatory values

Value	Type	Limits	Variable
SEC code	String	ppd/ccd/web	`sec`
Routing Number	String	9-character numeric	`routing_num`
Account Number	String	15-character alphanumeric	`account_num`
Account Type	String	savings/checking	`account_type`

Table 2: ACH Info object optional values

Value	Type	Limits	Variable
Customer First Name	String	50-character alphanumeric	`cust_first_name`
Customer Last Name	String	50-character alphanumeric	`cust_last_name`
Customer Address 1	String	50-character alphanumeric	`cust_address1`
Customer Address 2	String	50-character alphanumeric	`cust_address2`
Customer City	String	50-character alphanumeric	`cust_city`
Customer State	String	2-character alphanumeric	`cust_state`
Customer Zip Code	String	10-character numeric	`cust_zip`
Check Number	String	16-character numeric	`check_num`

Sample ACH Debit - US

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAACHDebit
{
public static void Main(string[] args)
{
string order_id = "Test" + DateTime.Now.ToString("yyyyMMddhhmmss");
string store_id = "monusqa002";
string api_token = "qatoken";
//string status = "true";
string amount = "1.00";
//ACHInfo Variables
string sec = "ppd";
string cust_first_name = "Christian";
string cust_last_name = "M";
string cust_address1 = "3300 Bloor St W";
string cust_address2 = "4th floor west tower";
string cust_city = "Toronto";
string cust_state = "ON";
string cust_zip = "M1M1M1";
string routing_num = "490000018";
```

Sample ACH Debit - US

```
string account_num = "222222";
string check_num = "11";
string account_type = "checking";
string micr = "t071000013t742941347o128";
string dl_num = "CO-12312312";
string magstripe = "no";
string image_front = "";
string image_back = "";
string processing_country_code = "US";
bool status_check = false;
ACHInfo achinfo = new ACHInfo(sec, cust_first_name, cust_last_name,
cust_address1, cust_address2, cust_city, cust_state, cust_zip,
routing_num, account_num, check_num, account_type, micr);
achinfo.SetImgFront(image_front);
achinfo.SetImgBack(image_back);
achinfo.SetDlNum(dl_num);
achinfo.SetMagstripe(magstripe);
ACHDebit achdebit = new ACHDebit();
achdebit.SetOrderId(order_id);
achdebit.SetAmount(amount);
achdebit.SetAchInfo(achinfo);
//************************OPTIONAL VARIABLES***************************
//Cust_id Variable
string cust_id = "customer1";
achdebit.SetCustId(cust_id);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(achdebit);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
/*Status Check Example
ACHHttpsPostRequest mpgReq = new ACHHttpsPostRequest(host, store_id, api_token, status, achdebit);
*/
/********************** REQUEST ************************/
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
//Console.WriteLine("StatusCode = " + receipt.GetStatusCode());
//Console.WriteLine("StatusMessage = " + receipt.GetStatusMessage());
Console.ReadLine();
}
catch (Exception e)
{
```

Sample ACH Debit - US

```
Console.WriteLine(e);
}
}
}
}
```

8.4 ACH Reversal

ACH Reversal transaction object definition

```
ACHReversal achreversal = new ACHReversal();
```

HttpsPostRequest object for ACH Reversal transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetTransaction(achreversal);
```

ACH Reversal transaction values

The ACH Reversal transaction requires the order ID and the transaction number from the corresponding ACH Debit transaction.

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 51: ACH Reversal transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`achreversal.SetOrderId(order_id);`
Transaction number	String	255-character variable	`achreversal.SetTxnNumber(txn_number);`

Table 52: ACH Reversal transaction optional values

Value	Type	Limits	Set method
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Sample ACH Reversal - US

```
namespace Moneris
{
```

[1]For more information, see Appendix C (page 280).

Sample ACH Reversal - US

```
using System;
public class TestUSAACHReversal
{
public static void Main(string[] args)
{
string order_id = "639178169517904319808133";
string txn_number = "42636-0_25";
string store_id = "monusqa002";
string api_token = "qatoken";
string processing_country_code = "US";
bool status_check = false;
ACHReversal achreversal = new ACHReversal();
achreversal.SetOrderId(order_id);
achreversal.SetTxnNumber(txn_number);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(achreversal);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
//Console.WriteLine("StatusCode = " + receipt.GetStatusCode());
//Console.WriteLine("StatusMessage = " + receipt.GetStatusMessage());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

8.5 ACH Credit

ACH Credit transaction object definition

```
ACHCredit achcredit = new ACHCredit();
```

HttpsPostRequest object for ACH Credit transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.SetTransaction(achcredit);
```

ACH Credit transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 53: ACH Credit transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`achcredit.SetOrderId(order_id);`
Amount	String	9-character decimal	`achcredit.SetAmount(amount);`
ACH Info[1]	Object	See ACH info object tables below for a list of variables	`achcredit.SetAchInfo(achinfo);`

Table 54: ACH Credit transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`achcredit`
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Table 1: ACH Info mandatory values

Value	Type	Limits	Set method
SEC code	String	ppd/ccd/web	`sec`
Routing Number	String	9-character numeric	`routing_num`
Account Number	String	15-character alphanumeric	`account_num`
Account Type	String	savings/checking	`account_type`

Table 2: ACH Info object optional values

Value	Type	Limits	Set method
Customer First Name	String	50-character alphanumeric	`cust_first_name`
Customer Last Name	String	50-character alphanumeric	`cust_last_name`
Customer Address 1	String	50-character alphanumeric	`cust_address1`
Customer Address 2	String	50-character alphanumeric	`cust_address2`

[1]The ACHCredit transaction may only be submitted with an SEC code of `ppd` or `ccd`.

[2]For more information, see Appendix C (page 280).

Value	Type	Limits	Set method
Customer City	String	50-character alphanumeric	`cust_city`
Customer State	String	2-character alphanumeric	`cust_state`
Customer Zip Code	String	10-character numeric	`cust_zip`
Check Number	String	16-character numeric	`check_num`

Sample code

```
namespace Moneris
{
using System;
public class TestUSAACHCredit
{
public static void Main(string[] args)
{
string order_id = "Test" + DateTime.Now.ToString("yyyyMMddhhmmss");
string store_id = "monusqa002";
string api_token = "qatoken";
string amount = "1.00";
//ACHInfo Variables
string sec = "ppd";
string cust_first_name = "Christian";
string cust_last_name = "M";
string cust_address1 = "3300 Bloor St W";
string cust_address2 = "4th floor west tower";
string cust_city = "Toronto";
string cust_state = "ON";
string cust_zip = "M1M1M1";
string routing_num = "490000018";
string account_num = "222222";
string check_num = "11";
string account_type = "checking";
string micr = "t071000013t742941347o129";
string dl_num = "CO-12312312";
string magstripe = "no";
string image_front =
    "SUkqAG4AAABUMDcxMDAwMDEzVDc0Mjk0MTM0N0E5OTkwAE1hZ1RlaywgSW5jLgAATUlDUkltYWdlIFJTMjMyIChBMDM3Nkw0KQBWZXJzaW9uIE
    CsIiob3nZ+q89K1DxuoYhhbt4VehBpLunkrcJBOiYhg/SaSROfwdWgRol/JcGEgfVBCsQyDIJLh7QUM7v4ToJgxPT10EEFEWxbC2v6dgh0quQQw
    y77emqBmsZGC/WyH8zsKEwgwmahNula++ZA9NNBp/8UqzEHGT6nJxTf0lioQPNCJyPAlNy/pBtEoYvwkqXhBhO0tJuE8INjVOgkLp37/6/C6C5d
    nIZCIiDIdeyS6gMIswkDhBXBAgQUKCBAkd1o7V9AwQIIEzyoIIIEEJnS0A8yAgQr+EGIIh16BQgiv4eQ8jsECK+AmMsxODBXcMEbzJTXRB18Idd
    iEvaXQIhCy4ILiKnQggkl9BJYIFtRC2mgwgwQKrCBbCnU3EREIEudlCBESQ8NZVTCCrsIFgwVRBJXEqAv0R0O59lJ6FwIiGXMuZcyOReLhTecyj
    mXiOy4Jg0VcJYREh6VAgR2HrxDBD5Q4QXighwrBXoFhAiGf0Eh8GJcf4qVA4cIH4IJQk36kIFCIV2COOW8KhuOynEw5xws74wl2/pAgR2B5XVY8
    a3RKQlhoLp0kp0INhJ0m0ggWgQQQX2oILYggtKGkEChhIJcMGEggtEdBBB0mMEFQyv2CCtWCQLBGWua8TPCC6DQIEWenCIzZwtIlRgghiCWohBT
    y49/4SMgaEslrKWD5mfeatBDlPHuePemECCXxf/NkCIPBuI55HBKGEH5gjBZe+wgmwQQlS8GVJf2qMO5bmfB9Z+gljB7iE/O8IEECEmx0JHxj8f
    LFCZoNUIiIiwjNsINONBBBBEHplUQjCkaQiIiIiIiEEdgWGGIno8sh9MhojplbHO51xMIQkQeSzBAjCZXHVxwRH3FigQQbGhERH4l8QgScMnRLb
    kXzBFEQZEHmvNxr9REREREhkgqjmBn0JDmsijnkoIxxERERERESGSGwchlKrRXMzEREREt6ZPHZTnM3GGTswzebzmcIjmR8wBguGoYjnFIRERER
    l/hSrO+VhQ6TMMuyOsRKgRFCxBAhBEfhxEqBCI5xTEER/qI2IRHP1ERERERFhCIie2EIqIj6+uvr6gAwAYAMAA==";
string image_back = "SUkqAGoAAAA/Pz8wPz8/Pz8/MD8/Pz8/Pz8/Pz80Pz8/MD8ATWFnVGVrLCBJbmMuAABNSUNSSW1hZ2UgUlMyMzIgKCBBMD
    V0ZHRHyOKR8jjBZTk4NB3BQQhAhL6jeEEbzDI4QEIkEaSzMlVkcyOZHF4nHQz1R0CwUQgiOiOBfCE7TSTIJlyI4L444g2J/MMjt7I4NRHMRJsh5
    R0iVv/UUigky8hdxC7/CCHCD71WkkPJD2E0LeTauLg3e4dBfv0qC587Rz8t1UzGRxSOiOGmtBjbfggtA6hQRpvwrZWMbLcPI4OXZgGmMhrkdY E
    lUSL0hbsTdDzGfWqiMIjrzNIIjoTCP8Js88/tbJj5Mf985AeJhH 3uRR4hBBL9iE8fJTDCr/Dij4sPpw0Puw1DNfWwwgzsh/sVhYpzrj9NMQgwh
    mqtmKKy4sU4aEREWE3GFZjN5jxI6e7EfqzThQnTOX8MPHQtBoRg6Dx6Edazr7diCBCOR9fBAiOvdNhBhCOsPBFPx/WtzHZ5Aor4JpuHER1 6sxn
    P4uHHxUfFREfceIS7u8xBDrx57CTaH/a2ml1f9hb/bu7QYWGmZycGEBYIFW1r7MP3g3H8isGcwY4QhhAwhEQwhEQYQk8IiIiIiIiMy4FiGTKg1v
    OXMcfJTGAMEfN6JNF3p4kgZs9uHaucze05ezFeSAzAj2TsXULg3mKDPFrr6CN5u1VP 7yPGAawxhBDCMJVI/0E8j6BQRHpVPt oTj/t9iDI4ai7
    3dg/CX8V2lf/yKbGSBF1xq/1IYu /7IU2Wozsz/q613b/8MPnZe 6 u//JxLBZA/uXRodycOpdVv/wemR8gt4 6eQ m199fSDcGjoZB LkDI4dd
    dqWRSWoIuoIE5oEU1u7/QoH8EXxI1mhHm/g5nLe1sM0jxnsuYJhPjwrEOMKCEINDDbO4eTYWzhEcMsH7CyolI6svHlKHOPDBEcw4SP W60MvkcM
    HCYlOg21CH8JoRESyXnXznxEILusLHHDJtTEdFuLZzI R0XM4gQIdldLiPlwYMRiCcdjjyuJhCOGYRzMGIiLrcrnDWLsxiIjoyUs2GuYYcSo9mX
string processing_country_code = "US";
bool status_check = false;
ACHInfo achinfo = new ACHInfo(sec, cust_first_name, cust_last_name,
cust_address1, cust_address2, cust_city, cust_state, cust_zip,
routing_num, account_num, check_num, account_type, micr);
achinfo.SetImgFront(image_front);
achinfo.SetImgBack(image_back);
achinfo.SetDlNum(dl_num);
achinfo.SetMagstripe(magstripe);
ACHCredit achcredit = new ACHCredit();
achcredit.SetOrderId(order_id);
achcredit.SetAmount(amount);
achcredit.SetAchInfo(achinfo);
//************************OPTIONAL VARIABLES**************************
//Cust_id Variable
string cust_id = "customer1";
achcredit.SetCustId(cust_id);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(achcredit);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
/*Status Check Example
ACHHttpsPostRequest mpgReq = new ACHHttpsPostRequest(host, store_id, api_token, status, achcredit);
*/
/********************** REQUEST ************************/
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
//Console.WriteLine("StatusCode = " + receipt.GetStatusCode());
//Console.WriteLine("StatusMessage = " + receipt.GetStatusMessage());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
```

```
}
}
```

8.6 ACH Fi Inquiry

ACHFiInquiry transaction object definition

```
ACHFiInquiry achfiinquiry = new ACHFiInquiry();
```

HttpsPostRequest object for ACH Fi Inquiry transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(achfiinquiry);
```

ACH Fi Inquiry transaction object mandatory arguments

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 55: ACH Fi Inquiry transaction object mandatory values

Value	Type	Limits	Set method
Routing number	String	9-character numeric	`achfiinquiry.`**`SetRoutingNum(routing_num)`**`;`

ACH Fi Inquiry transaction optional values: None.

Sample ACH Fi Inquiry - US

```
namespace Moneris
{
using System;
public class TestUSAACHFiInquiry
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string routing_num = "071000013";
string processing_country_code = "US";
bool status_check = false;
ACHFiInquiry achfiinquiry = new ACHFiInquiry();
achfiinquiry.SetRoutingNum(routing_num);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(achfiinquiry);
mpgReq.SetStatusCheck(status_check);
```

Sample ACH Fi Inquiry - US

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

9 Vault Transaction Set

The Vault feature allows merchants to create customer profiles, edit those profiles, and use them to process transactions without having to enter financial information each time. Customer profiles store customer data essential to processing transactions, including credit, signature debit and ACH payment details.

The Vault is a complement to the recurring payment module. It securely stores customer account information on Moneris secure servers. This allows merchants to bill customers for routine products or services when an invoice is due.

Any of the transaction objects that are defined in this section can be passed to the HttpsPostRequest connection object defined in Section 4 (page 24).

9.1 Vault Transaction Types

The Vault API supports both administrative and financial transactions.

9.1.1 Administrative Vault Transaction types

ResAddCC

Creates a new credit card profile, and generates a unique data key which can be obtained from the Receipt object.

This data key is the profile identifier that all future financial Vault transactions will use to associate with the saved information (see 9.2.1.1, page 113).

EncResAddCC

Creates a new credit card profile, but requires the card data to be either swiped or manually keyed in via a Moneris-provided encrypted mag swipe reader.

ResAddACH

Creates a new ACH profile. A data key is generated and returned to the merchant in the response.

For more information about the data key, see "Data Key" on page 113.

ResTempAdd

TBD

ResUpdateCC

Updates a Vault profile (based on the data key) to contain credit card information.

All information contained within a credit card profile is updated as indicated by the submitted fields. The fields are explained in more detail in "Administrative Transactions" on page 111.

EncResUpdateCC

Updates a profile (based on the data key) to contain credit card information. The encrypted version of this transaction requires the card data to either be swiped or manually keyed in via a Moneris-provided encrypted mag swipe reader.

ResUpdateACH
Updates a Vault profile (based on the unique data key) to contain ACH information.

ResDelete
Deletes an existing Vault profile of any type using the unique data key that was assigned when the profile was added.

It is important to note that after a profile is deleted, the information which was saved within can no longer be retrieved.

ResLookupFull
Verifies what is currently saved under the Vault profile associated with the given data key. The response to this transaction returns the latest active data for that profile.

Unlike ResLookupMasked (which returns the masked credit card number), this transaction returns both the masked and the unmasked credit card numbers.

ResLookupMasked
Verifies what is currently saved under the Vault profile associated with the given data key. The response to this transaction returns the latest active data for that profile.

Unlike ResLookupFull (which only returns both the masked and the unmasked credit card numbers), this transaction only returns the masked credit card number.

ResGetExpiring
Verifies which profiles have credit cards that are expiring during the current and next calendar month. For example, if you are processing this transaction on September 30, then it will return all cards that expire(d) in September and October of this year.

When generating a list of profiles with expiring credit cards, only the **masked** credit card numbers are returned.

This transaction can be performed no more than 2 times on any given calendar day, and it only applies to credit card profiles.

ResIscorporatecard
Determines whether a profile has a corporate card registered within it.

After sending the transaction, the response field to the Receipt object's getCorporateCard method is either `true` or `false` depending on whether the associated card is a corporate card.

ResAddToken
Converts a Hosted Tokenization temporary token to a permanent Vault token.

A temporary token is valid for 15 minutes after it is created.

ResTokenizeCC
Creates a new credit card profile using the credit card number, expiry date and e-commerce indicator that were submitted in a previous financial transaction. A transaction that was previously done in Moneris Payment Gateway is taken, and the card date from that transaction is stored in the Moneris Vault.

As with ResAddCC, a unique data key is generated and returned to the merchant via the Receipt object. This is the profile identifier that all future financial Vault transactions will use to associate with the saved information.

For more information about the data key, see "Data Key" on page 113.

ResTempTokenize
TBD

9.1.2 Financial Vault Transaction types

ResPurchaseCC
Uses the data key to identify a previously registered credit card profile. The details saved within the profile are then submitted to perform a Purchase transaction.

ResPurchaseACH
This transaction is processed as an ACHDebit. The ACHInfo registered for this profile will be used. The details submitted within ACHInfo object are returned in the response within ResolveData.

ResPreauthCC
Uses the data key to identify a previously registered credit card profile. The details within the profile are submitted to perform a Pre-Authorization transaction.

ResIndRefundCC
Uses the unique data key to identify a previously registered credit card profile, and credits a specified amount to that credit card.

ResIndRefundACH
Uses the unique data key to identify a previously registered ACH profile, and credits a specified amount to that credit card. This is processed as an ACHCredit.

ResMpiTxn
Uses the data key (as opposed to a credit card number) in a VBV/SecureCode Txn MPI transaction. The merchant uses the data key with ResMpiTxn request, and then reads the response fields to verify whether the card is enrolled in Verified by Visa or MasterCard SecureCode. Retrieves the vault transaction value to pass on to Visa or Mastercard.

After it has been validated that the data key is is enrolled in 3ds, a window appears in which the customer can enter the 3ds password. The merchant may initiate the forming of the validation form `(getMpiInLineForm()`.

For more information on integrating with MonerisMPI, refer to the MPISection in this guide

9.1.3 Charging a Temporary Token

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

The following financial transactions can charge a temporary token:

- ResPurchaseCC (page 147)
- ResPreauthCC (page 152)
- ResIndRefundCC (page 156).

A temporary token can be made permanent by using the ResAddTokenCC transaction (page 142).

9.2 Administrative Transactions

Administrative transactions allow you to perform such tasks as creating new Vault profiles, deleting existing Vault profiles and updating profile information.

9.2.1 Vault Add Credit Card- ResAddCC

ResAddCC transaction object definition

```
ResAddCC resaddcc = new ResAddCC();
```

HttpsPostRequest object for ResAddCC transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resaddcc);
```

ResAddCC transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 56: ResAddCC transaction object mandatory values

Value	Type	Limits	Set method
Credit card number	String	20-character alphanumeric	`resaddcc.SetPan(pan);`
Expiry date	String	4-character alphanumeric (YYMM format)	`resaddcc..SetExpdate (expdate);`
E-commerce indicator	String	1-character alphanumeric[1]	`resaddcc..SetCryptType (crypt);`

Table 57: Purchase transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`resaddcc.SetCustId(cust_id);`
AVS information	Object	Not applicable. See Appendix E (page 288).	`resaddcc..SetAvsInfo(avsCheck);`
Email address	String	30-character alphanumeric	`resaddcc..SetEmail(email);`
Phone number	String	30-character alphanumeric	`resaddcc..SetPhone(phone);`
Note	String	30-character alphanumeric	`resaddcc..SetNote(note);`

[1]Full explanation on page 259

Sample ResAddCC - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaResAddCC
{
public static void Main(string[] args)
{
string store_id = "store5";
string api_token = "yesguy";
string pan = "4242424242424242";
string expdate = "1912";
string phone = "0000000000";
string email = "bob@smith.com";
string note = "my note";
string cust_id = "customer1";
string crypt_type = "7";
string processing_country_code = "CA";
bool status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
ResAddCC resaddcc = new ResAddCC();
resaddcc.SetPan(pan);
resaddcc.SetExpdate(expdate);
resaddcc.SetCryptType(crypt_type);
resaddcc.SetCustId(cust_id);
resaddcc.SetPhone(phone);
resaddcc.SetEmail(email);
resaddcc.SetNote(note);
resaddcc.SetAvsInfo(avsCheck);
resaddcc.SetGetCardType("true");
//resaddcc.SetDataKeyFormat("0"); //1=F6L4 w/
    Length preserve, 2=F6L4 w/o Length
    preserve
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resaddcc);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
```

Sample ResAddCC - US

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAResAddCC
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string pan = "5454545454545454";
string expdate = "1602"; //YYMM format
string phone = "0000000000";
string email = "bob@smith.com";
string note = "my note";
string cust_id = "customer1";
string crypt_type = "7";
string processing_country_code = "US";
bool status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
ResAddCC resaddcc = new ResAddCC();
resaddcc.SetPan(pan);
resaddcc.SetExpdate(expdate);
resaddcc.SetCryptType(crypt_type);
resaddcc.SetCustId(cust_id);
resaddcc.SetPhone(phone);
resaddcc.SetEmail(email);
resaddcc.SetNote(note);
resaddcc.SetAvsInfo(avsCheck);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resaddcc);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
```

Sample ResAddCC - CA

```
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
Console.WriteLine("Cust ID = " +
    receipt.GetResDataCustId());
Console.WriteLine("Phone = " +
    receipt.GetResDataPhone());
Console.WriteLine("Email = " +
    receipt.GetResDataEmail());
Console.WriteLine("Note = " +
    receipt.GetResDataNote());
Console.WriteLine("MaskedPan = " +
    receipt.GetResDataMaskedPan());
Console.WriteLine("Exp Date = " +
    receipt.GetResDataExpdate());
Console.WriteLine("Crypt Type = " +
    receipt.GetResDataCryptType());
Console.WriteLine("Avs Street Number = " +
    receipt.GetResDataAvsStreetNumber());
Console.WriteLine("Avs Street Name = " +
    receipt.GetResDataAvsStreetName());
Console.WriteLine("Avs Zipcode = " +
    receipt.GetResDataAvsZipcode());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Sample ResAddCC - US

```
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
Console.WriteLine("Cust ID = " +
    receipt.GetResDataCustId());
Console.WriteLine("Phone = " +
    receipt.GetResDataPhone());
Console.WriteLine("Email = " +
    receipt.GetResDataEmail());
Console.WriteLine("Note = " +
    receipt.GetResDataNote());
Console.WriteLine("MaskedPan = " +
    receipt.GetResDataMaskedPan());
Console.WriteLine("Exp Date = " +
    receipt.GetResDataExpdate());
Console.WriteLine("Crypt Type = " +
    receipt.GetResDataCryptType());
Console.WriteLine("Avs Street Number = " +
    receipt.GetResDataAvsStreetNumber());
Console.WriteLine("Avs Street Name = " +
    receipt.GetResDataAvsStreetName());
Console.WriteLine("Avs Zipcode = " +
    receipt.GetResDataAvsZipcode());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.2.1.1 Data Key

The ResAddCC sample code includes the following instruction from the Receipt object:

The data key response field is populated when you send a ResAddCC transaction or a ResTokenizeCC transaction (page 145). It is the profile identifier that all future financial Vault transactions will use to associate with the saved information.

The data key is a maximum 25-character alphanumeric string.

9.2.1.2 Vault Encrypted Add Credit Card - EncResAddCC

EncResAddCC transaction object definition

```
EncResAddCC encresaddcc = new EncResAddCC();
```

HttpsPostRequest object for EncResAddCC transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(encresaddcc);
```

EncResAddCC transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 58: EncResAddCC transaction object mandatory values

Value	Type	Limits	Set method
Encrypted Track2 data	String	40-character numeric	`encresaddcc..SetEncTrack2 (enc_track2);`
Device type	String	TBD	`encresaddcc..SetDeviceType (device_type);`
E-commerce indicator	String	1-character alphanumeric[1]	`encresaddcc..SetCryptType (crypt);`

Table 59: EncResAddCC transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`encresaddcc.SetCustId(cust_id);`
AVS information	Object	Not applicable. See Appendix E (page 288).	`encresaddcc..SetAvsInfo (avsCheck);`
Email address	String	30-character alphanumeric	`encresaddcc..SetEmail(email);`
Phone number	String	30-character alphanumeric	`encresaddcc..SetPhone(phone);`
Note	String	30-character alphanumeric	`encresaddcc..SetNote(note);`

Sample Encrypted ResAddCC - CA	Sample Encrypted ResAddCC - US
`namespace Moneris {`	`namespace Moneris {`

[1]Full explanation on page 259

Sample Encrypted ResAddCC - CA

```
using System;
public class TestCanadaEncResAddCC
{
public static void Main(string[] args)
{
/******************* REQUEST
    VARIABLES*******************************/
string store_id = "store5";
string api_token = "yesguy";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "nqa";
string amount = "1.00";
string device_type = "idtech_bdk";
string crypt = "7";
string enc_track2 =
    "02840085000000000416BC6FCE0D7A8B07E6278E6
    0D237CA9362767ADC2C93A2EA5D9BED3E4D1A791C3
    F4FC61C1800486A8A6B6CCAA00431353131FFFF314
    1594047A00090055103";
string processing_country_code = "CA";
bool status_check = false;

EncResAddCC encresaddcc = new EncResAddCC();
encresaddcc.SetEncTrack2(enc_track2);
encresaddcc.SetDeviceType(device_type);
encresaddcc.SetCryptType(crypt);
encresaddcc.SetCustId(cust_id);
encresaddcc.SetNote("Just a note");
encresaddcc.SetEmail("example@test.com");
encresaddcc.SetPhone("866-319-7450");

/*************** Address Verification Service
    **********************/
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
encresaddcc.SetAvsInfo(avsCheck);

HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(encresaddcc);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
```

Sample Encrypted ResAddCC - US

```
using System;
public class TestUSAEncResAddCC
{
public static void Main(string[] args)
{
/******************* REQUEST
    VARIABLES*******************************/
string store_id = "monusqa002";
string api_token = "qatoken";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "nqa";
string crypt = "7";
string processing_country_code = "US";
bool status_check = false;
string enc_track2 =
    "028400850000000004142348E7643B2599ACC0051
    7C5AB6FB164486B1A4A83E7A81048D6CBA51604FDD
    12B72C228028E727AF6664C7A0431393035FFFF314
    1594047A0009E79C903";
string device_type = "idtech";

EncResAddCC encresaddcc = new EncResAddCC();
encresaddcc.SetEncTrack2(enc_track2);
encresaddcc.SetDeviceType(device_type);
encresaddcc.SetCryptType(crypt);
encresaddcc.SetCustId(cust_id);
encresaddcc.SetNote("Just a note");
encresaddcc.SetEmail("example@test.com");
encresaddcc.SetPhone("866-319-7450");

/*************** Address Verification Service
    **********************/
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
encresaddcc.SetAvsInfo(avsCheck);

HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(encresaddcc);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
```

Sample Encrypted ResAddCC - CA

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

Sample Encrypted ResAddCC - US

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

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.2.2 Vault Add ACH - ResAddACH

Things to consider:

- Only the following SEC codes are currently supported: PPD, CCD, and WEB.
- The SEC code, along with the rest of the ACHInfo object data will be submitted with all future Vault transactions unless it is later updated.

ResAddACH transaction object definition

```
ResAddAch ressaddach = new ResAddAch();
```

HttpsPostRequest object for ResAddACH transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(ressaddach);
```

ResAddACH transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 60: ResAddACH transaction object mandatory values

Value	Type	Limits	Set method
ACH Info	Object	Not applicable. See 8.2 (page 94).	`ressaddach.SetAchInfo (achinfo);`

Table 61: ResAddACH transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`ressaddach.SetCustId(cust_id);`
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`
Email address	String	30-character alphanumeric	`ressaddach.SetEmail(email);`
Phone number	String	30-character alphanumeric	`ressaddach.SetPhone(phone);`
Note	String	30-character alphanumeric	`ressaddach.SetNote(note);`

Sample code

Sample ResAddACH - US

```
namespace Moneris
{
using Moneris;
```

[1]For more information, see Appendix C (page 280).

Sample ResAddACH - US

```
using System;
using System.Text;
using System.Collections;
public class TestUSAResAddAch
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string phone = "0000000000";
string email = "bob.smith@moneris.com";
string note = "my note";
string cust_id = "customer1";
//ACHInfo Variables
string sec = "ppd";
string cust_first_name = "Christian";
string cust_last_name = "M";
string cust_address1 = "3300 Bloor St W";
string cust_address2 = "4th floor west tower";
string cust_city = "Toronto";
string cust_state = "ON";
string cust_zip = "M1M1M1";
string routing_num = "490000018";
string account_num = "222222";
string check_num = "11";
string account_type = "checking";
string processing_country_code = "US";
bool status_check = false;
ACHInfo achinfo = new ACHInfo(sec, cust_first_name, cust_last_name,
cust_address1, cust_address2, cust_city, cust_state, cust_zip,
routing_num, account_num, check_num, account_type);
//alternatively, each field of ACHInfo can be set individually
/*ACHInfo achinfo = new ACHInfo();
//***********************MANDATORY ACH VARIABLES***************************
achinfo.SetSec(sec);
achinfo.SetRoutingNum(routing_num);
achinfo.SetAccountNum(account_num);
achinfo.SetAccountType(account_type);
//***********************OPTIONAL ACH VARIABLES***************************
achinfo.SetCustFirstName(cust_first_name);
achinfo.SetCustLastName(cust_last_name);
achinfo.SetCustAddress1(cust_address1);
achinfo.SetCustAddress2(cust_address2);
achinfo.SetCustCity(cust_city);
achinfo.SetCustState(cust_state);
achinfo.SetCustZip(cust_zip);
achinfo.SetCheckNum(check_num);
*/
ResAddAch ressaddach = new ResAddAch();
ressaddach.SetAchInfo(achinfo);
ressaddach.SetCustId(cust_id);
ressaddach.SetPhone(phone);
ressaddach.SetEmail(email);
ressaddach.SetNote(note);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
```

Sample ResAddACH - US

```
mpgReq.SetTransaction(ressaddach);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " + receipt.GetDataKey());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
Console.WriteLine("PaymentType = " + receipt.GetPaymentType());
Console.WriteLine("Cust ID = " + receipt.GetResDataCustId());
Console.WriteLine("Phone = " + receipt.GetResDataPhone());
Console.WriteLine("Email = " + receipt.GetResDataEmail());
Console.WriteLine("Note = " + receipt.GetResDataNote());
Console.WriteLine("Sec = " + receipt.GetResDataSec());
Console.WriteLine("Cust First Name = " + receipt.GetResDataCustFirstName());
Console.WriteLine("Cust Last Name = " + receipt.GetResDataCustLastName());
Console.WriteLine("Cust Address 1 = " + receipt.GetResDataCustAddress1());
Console.WriteLine("Cust Address 2 = " + receipt.GetResDataCustAddress2());
Console.WriteLine("Cust City = " + receipt.GetResDataCustCity());
Console.WriteLine("Cust State = " + receipt.GetResDataCustState());
Console.WriteLine("Cust Zip = " + receipt.GetResDataCustZip());
Console.WriteLine("Routing Num = " + receipt.GetResDataRoutingNum());
Console.WriteLine("Masked Account Num = " + receipt.GetResDataMaskedAccountNum());
Console.WriteLine("Check Num = " + receipt.GetResDataCheckNum());
Console.WriteLine("Account Type = " + receipt.GetResDataAccountType());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.2.3 Vault Add Temporary Token - ResTempAdd

ResTempAdd transaction object definition

```
ResTempAdd resTempAdd = new ResTempAdd();
```

HttpsPostRequest object for ResTempAdd transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resTempAdd);
```

ResTempAdd transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 62: ResTempAdd transaction object mandatory values

Value	Type	Limits	Set method
Credit card number	String	20-character numeric	`resTempAdd.SetPan(pan);`
Expiry date	String	4-character numeric	`resTempAdd.SetExpdate(expdate);`
Duration	String	TBD	`resTempAdd.SetDuration(duration);`
E-commerce indicator	String	1-character alphanumeric[1]	`resTempAdd.SetCryptType(crypt);`

Table 63: ResTempAdd transaction optional values

Value	Type	Limits	Set method
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck (status_check);`

Sample ResTempAdd - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaResTempAdd
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string pan = "5454545454545454";
string expdate = "1901"; //YYMM format
string crypt_type = "7";
string duration = "900";
string processing_country_code = "CA";
bool status_check = false;
ResTempAdd resTempAdd = new ResTempAdd();
resTempAdd.SetPan(pan);
resTempAdd.SetExpdate(expdate);
resTempAdd.SetDuration(duration);
resTempAdd.SetCryptType(crypt_type);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
```

Sample ResTempAdd - US

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAResTempAdd
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string pan = "5454545454545454";
string expdate = "1902"; //YYMM format
string crypt_type = "7";
string duration = "900";
string processing_country_code = "US";
bool status_check = false;
ResTempAdd resTempAdd = new ResTempAdd();
resTempAdd.SetPan(pan);
resTempAdd.SetExpdate(expdate);
resTempAdd.SetDuration(duration);
resTempAdd.SetCryptType(crypt_type);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
```

[1] Full explanation on page 259

[2] For more information, see Appendix C (page 280).

Sample ResTempAdd - CA

```
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resTempAdd);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
Console.WriteLine("MaskedPan = " +
    receipt.GetResDataMaskedPan());
Console.WriteLine("Exp Date = " +
    receipt.GetResDataExpdate());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Sample ResTempAdd - US

```
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resTempAdd);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
Console.WriteLine("MaskedPan = " +
    receipt.GetResDataMaskedPan());
Console.WriteLine("Exp Date = " +
    receipt.GetResDataExpdate());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.2.4 Vault Update Credit Card - ResUpdateCC

ResUpdateCC transaction object definition

```
ResUpdateCC resUpdateCC = new ResUpdateCC();
```

HttpsPostRequest object for ResUpdateCC transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resUpdateCC);
```

ResUpdateCC transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 64: ResUpdateCC transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alphanumeric	`resUpdateCC.SetData(data_key);`

Optional values that are submitted to the ResUpdateCC object are updated. Unsubmitted optional values (with one exception) remain unchanged. This allows you to change only the fields you want. The exception is that if you are making changes to the payment type, **all** of the shaded values in Table 65 must be submitted.

If you update a profile to a different payment type, it is automatically deactivated and a new credit card profile is created and assigned to the data key. The only values from the prior profile that will remain unchanged are the customer ID, phone number, email address, and note. For example, if a profile contains AVS information, but a ResUpdateCC transaction is submitted without an AVSInfo object, the existing AVSInfo details are deactivated and the new credit card information is registered without AVS.

Table 65: ResUpdateCC transaction optional values

Value	Type	Limits	Set method
Credit card number	String	20-character alphanumeric	`resUpdateCC.SetPan(pan);`
Expiry date	String	4-character alphanumeric (YYMM format)	`resUpdateCC.SetExpdate(expdate);`
E-commerce indicator	String	1-character alphanumeric[1]	`resUpdateCC.SetCryptType(crypt);`
Customer ID	String	50-character alphanumeric	`resUpdateCC.SetCustId(cust_id);`
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`
AVS information	Object	Not applicable. See Appendix E (page 288).	`resUpdateCC.SetAvsInfo(avsCheck);`

[1]Full explanation on page 259

[2]For more information, see Appendix C (page 280).

Table 65: ResUpdateCC transaction optional values

Value	Type	Limits	Set method
Email address	String	30-character alphanumeric	`resUpdateCC.SetEmail (email);`
Phone number	String	30-character alphanumeric	`resUpdateCC.SetPhone (phone);`
Note	String	30-character alphanumeric	`resUpdateCC.SetNote(note);`

Sample ResUpdateCC - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaResUpdateCC
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string data_key = "cIjurYyhGCAiGuCKdp94AspE7";
string pan = "4242424242424242";
string expdate = "1901";
string phone = "0000000000";
string email = "bob@smith.com";
string note = "my note";
string cust_id = "customer1";
string crypt_type = "7";
string processing_country_code = "CA";
bool status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
ResUpdateCC resUpdateCC = new ResUpdateCC();
resUpdateCC.SetData(data_key);
resUpdateCC.SetAvsInfo(avsCheck);
resUpdateCC.SetCustId(cust_id);
resUpdateCC.SetPan(pan);
resUpdateCC.SetExpdate(expdate);
resUpdateCC.SetPhone(phone);
resUpdateCC.SetEmail(email);
resUpdateCC.SetNote(note);
resUpdateCC.SetCryptType(crypt_type);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resUpdateCC);
mpgReq.SetStatusCheck(status_check);
```

Sample ResUpdateCC - US

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAResUpdateCC
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string data_key = "yd7qyMBTS1uU4BsLQvPAEeddY";
string pan = "4242424242424242";
string expdate = "1901"; //YYMM format
string phone = "0000000000";
string email = "bob@smith.com";
string note = "my note";
string cust_id = "customer1";
string crypt_type = "7";
string processing_country_code = "US";
bool status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
ResUpdateCC usResUpdateCC = new ResUpdateCC();
usResUpdateCC.SetAvsInfo(avsCheck);
usResUpdateCC.SetCustId(cust_id);
usResUpdateCC.SetPan(pan);
usResUpdateCC.SetExpdate(expdate);
usResUpdateCC.SetPhone(phone);
usResUpdateCC.SetEmail(email);
usResUpdateCC.SetNote(note);
usResUpdateCC.SetCryptType(crypt_type);
usResUpdateCC.SetData(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(usResUpdateCC);
mpgReq.SetStatusCheck(status_check);
```

Sample ResUpdateCC - CA

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

Sample ResUpdateCC - US

```
mpgReq.Send();

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

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.2.4.1 EncResUpdateCC

EncResUpdateCC transaction object definition

HttpsPostRequest object for EncResUpdateCC transaction

EncResUpdateCC transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 66: EncResUpdateCC transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alphanumeric	`encresupdatecc.SetData(data_key);`

Optional values that are submitted to the ResUpdateCC object are updated. Unsubmitted optional values (with one exception) remain unchanged. This allows you to change only the fields you want. The exception is that if you are making changes to the payment type, **all** of the shaded values in Table 67 must be submitted.

If you update a profile to a different payment type, it is automatically deactivated and a new credit card profile is created and assigned to the data key. The only values from the prior profile that will remain unchanged are the customer ID, phone number, email address, and note. For example, if a profile contains AVS information, but a ResUpdateCC transaction is submitted without an AVSInfo object, the existing AVSInfo details are deactivated and the new credit card information is registered without AVS.

Table 67: EncResUpdateCC transaction optional values

Value	Type	Limits	Set method
Encrypted Track2 data	String	40-character numeric	`encresupdatecc.SetEncTrack2 (enc_track2);`
Device type	String	TBD	`encresupdatecc.SetDeviceType (device_type);`
E-commerce indicator	String	1-character alphanumeric[1]	`encresupdatecc.SetCryptType (crypt);`
Customer ID	String	50-character alphanumeric	`encresupdatecc`
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck (status_check);`

[1]Full explanation on page 259

[2]For more information, see Appendix C (page 280).

Table 67: EncResUpdateCC transaction optional values

Value	Type	Limits	Set method
AVS information	Object	Not applicable. See Appendix E (page 288).	`encresupdatecc.SetAvsInfo (avsCheck);`
Email address	String	30-character alphanumeric	`encresupdatecc.SetEmail (email);`
Phone number	String	30-character alphanumeric	`encresupdatecc.SetPhone (phone);`
Note	String	30-character alphanumeric	`encresupdatecc.SetNote (note);`

Sample code

Sample EncResUpdateCC - CA

```
namespace Moneris
{
using System;
public class TestCanadaEncResUpdateCC
{
public static void Main(string[] args)
{
/******************* REQUEST
    VARIABLES*******************************/
string store_id = "store5";
string api_token = "yesguy";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "nqa";
string device_type = "idtech_bdk";
string crypt = "7";
string enc_track2 =
    "028400850000000000416BC6FCE0D7A8B07E6278E6
    0D237CA9362767ADC2C93A2EA5D9BED3E4D1A791C3
    F4FC61C1800486A8A6B6CCAA00431353131FFFF314
    1594047A00090055103";
string processing_country_code = "CA";
string data_key = "gF5IpsWD3s42r2TZxZyecE9Gs";
bool status_check = false;

EncResUpdateCC encresupdatecc = new
    EncResUpdateCC();
encresupdatecc.SetDataKey(data_key);
encresupdatecc.SetCustId(cust_id);
encresupdatecc.SetNote("Just a note2");
encresupdatecc.SetEmail("example1@test.com");
encresupdatecc.SetPhone("866-319-7450");
encresupdatecc.SetEncTrack2(enc_track2);
encresupdatecc.SetDeviceType(device_type);
encresupdatecc.SetCryptType(crypt);

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

Sample EncResUpdateCC - US

```
using System;
namespace Moneris{
using Moneris;
using System.Collections;
using System;
public class TestUSAEncResUpdateCC
{

public static void Main(string[] args)
{
String store_id = "monusqa002";
String api_token = "qatoken";
String data_key = "ZjjRgfpvUEBysJO5eSUAB242U";
String enc_track2 =
    "02840085000000000004142348E7643B2599ACC0051
    7C5AB6FB164486B1A4A83E7A81048D6CBA51604FDD
    12B72C228028E727AF6664C7A0431393035FFFF314
    1594047A0009E79C903";
String device_type = "idtech";
String phone = "55555555555";
String email = "test.user@moneris.com";
String note = "my note";
String cust_id = "customer2";
String crypt = "7";
String processing_country_code = "US";

AvsInfo avsinfo = new AvsInfo();
avsinfo.SetAvsStreetNumber("212");
avsinfo.SetAvsStreetName("Smith Street");
avsinfo.SetAvsZipCode("M1M1M1");
EncResUpdateCC enc_res_update_cc = new
    EncResUpdateCC ();
enc_res_update_cc.SetDataKey(data_key);
enc_res_update_cc.SetAvsInfo(avsinfo);
enc_res_update_cc.SetCustId(cust_id);
enc_res_update_cc.SetEncTrack2(enc_track2);
enc_res_update_cc.SetDeviceType(device_type);
enc_res_update_cc.SetPhone(phone);
```

Sample EncResUpdateCC - CA	Sample EncResUpdateCC - US

```
    **********************/
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("3300");
avsCheck.SetAvsStreetName("Bloor Street");
avsCheck.SetAvsZipCode("M2X2X2");
encresupdatecc.SetAvsInfo(avsCheck);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(encresupdatecc);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();

try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
//ResolveData
Console.WriteLine("\nCust ID = " +
    receipt.GetResDataCustId());
Console.WriteLine("Phone = " +
    receipt.GetResDataPhone());
Console.WriteLine("Email = " +
    receipt.GetResDataEmail());
Console.WriteLine("Note = " +
    receipt.GetResDataNote());
Console.WriteLine("MaskedPan = " +
    receipt.GetResDataMaskedPan());
Console.WriteLine("Exp Date = " +
    receipt.GetResDataExpdate());
Console.WriteLine("Crypt Type = " +
    receipt.GetResDataCryptType());
Console.WriteLine("Avs Street Number = " +
    receipt.GetResDataAvsStreetNumber());
Console.WriteLine("Avs Street Name = " +
    receipt.GetResDataAvsStreetName());
```
	```
	enc_res_update_cc.SetEmail(email);
	enc_res_update_cc.SetNote(note);
	enc_res_update_cc.SetCryptType(crypt);
	HttpsPostRequest mpgReq = new HttpsPostRequest
	    ();
	mpgReq.SetProcCountryCode(processing_country_
	    code);
	mpgReq.SetTestMode(true); //false or comment
	    out this line for production transactions
	mpgReq.SetStoreId(store_id);
	mpgReq.SetApiToken(api_token);
	mpgReq.SetTransaction(enc_res_update_cc);
	mpgReq.Send();

	try
	{
	Receipt receipt = mpgReq.GetReceipt();
	Console.WriteLine("DataKey = " +
	    receipt.GetDataKey());
	Console.WriteLine("ResponseCode = " +
	    receipt.GetResponseCode());
	Console.WriteLine("Message = " +
	    receipt.GetMessage());
	Console.WriteLine("TransDate = " +
	    receipt.GetTransDate());
	Console.WriteLine("TransTime = " +
	    receipt.GetTransTime());
	Console.WriteLine("Complete = " +
	    receipt.GetComplete());
	Console.WriteLine("TimedOut = " +
	    receipt.GetTimedOut());
	Console.WriteLine("ResSuccess = " +
	    receipt.GetResSuccess());
	Console.WriteLine("PaymentType = " +
	    receipt.GetPaymentType() + "\n");
	//Contents of ResolveData
	Console.WriteLine("Cust ID = " +
	    receipt.GetResCustId());
	Console.WriteLine("Phone = " +
	    receipt.GetResPhone());
	Console.WriteLine("Email = " +
	    receipt.GetResEmail());
	Console.WriteLine("Note = " +
	    receipt.GetResNote());
	Console.WriteLine("MaskedPan = " +
	    receipt.GetResMaskedPan());
	Console.WriteLine("Exp Date = " +
	    receipt.GetResExpDate());
	Console.WriteLine("Crypt Type = " +
	    receipt.GetResCryptType());
	Console.WriteLine("Avs Street Number = " +
	    receipt.GetResAvsStreetNumber());
	Console.WriteLine("Avs Street Name = " +
	    receipt.GetResAvsStreetName());
	Console.WriteLine("Avs Zipcode = " +
	    receipt.GetResAvsZipcode());
	}
	catch (Exception e)
	```

Sample EncResUpdateCC - CA	Sample EncResUpdateCC - US
`Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode()); Console.ReadLine(); } catch (Exception e) { Console.WriteLine(e); } } } }`	`{ Console.WriteLine(e); } } } }`

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.2.5 ResUpdateACH

If the profile that is being updated was already an ACH profile, all information contained within it will be updated as indicated by the submitted fields.

If the profile was of a different payment type (e.g., credit card), the old profile is deactivated and the new ACH information is associated with the data key.

ResUpdateAch transaction object definition

```
ResUpdateAch resUpdateAch = new ResUpdateAch();
```

HttpsPostRequest object for ResUpdateACH transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resUpdateAch);
```

ResUpdateACH transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 68: ResUpdateAch transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alphanumeric	`resUpdateAch.SetData(data_key);`
ACH Info	Object	Not applicable. See 8.2 (page 94).	`resUpdateAch.SetAchInfo(achinfo);`

Table 69: ResUpdateACH transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`resUpdateAch`
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck (status_check);`
Email address	String	30-character alphanumeric	`resUpdateAch.SetEmail (email);`
Phone number	String	30-character alphanumeric	`resUpdateAch.SetPhone (phone);`
Note	String	30-character alphanumeric	`resUpdateAch.SetNote(note);`

Sample ResUpdateAch

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAResUpdateAch
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string data_key = "0HjrtlV2VCu4DRRv8zwZmjbJk";
string phone = "0000000005";
string email = "bob@smith.com";
string note = "my note";
string cust_id = "customer1";
//ACHInfo Variables
string sec = "ppd";
string cust_first_name = "Christian";
string cust_last_name = "M";
string cust_address1 = "3300 Bloor St W";
string cust_address2 = "4th floor west tower";
string cust_city = "Toronto";
string cust_state = "ON";
string cust_zip = "M1M1M1";
string routing_num = "490000018";
string account_num = "222222";
string check_num = "11";
string account_type = "checking";
string processing_country_code = "US";
bool status_check = false;
ACHInfo achinfo = new ACHInfo();
achinfo.SetSec(sec);
achinfo.SetCustFirstName(cust_first_name);
achinfo.SetCustLastName(cust_last_name);
achinfo.SetCustAddress1(cust_address1);
achinfo.SetCustAddress2(cust_address2);
```

[1]For more information, see Appendix C (page 280).

Sample ResUpdateAch

```
achinfo.SetCustCity(cust_city);
achinfo.SetCustState(cust_state);
achinfo.SetCustZip(cust_zip);
achinfo.SetRoutingNum(routing_num);
achinfo.SetAccountNum(account_num);
achinfo.SetCheckNum(check_num);
achinfo.SetAccountType(account_type);
ResUpdateAch resUpdateAch = new ResUpdateAch();
resUpdateAch.SetDataKey(data_key);
resUpdateAch.SetAchInfo(achinfo);
resUpdateAch.SetCustId(cust_id);
resUpdateAch.SetPhone(phone);
resUpdateAch.SetEmail(email);
resUpdateAch.SetNote(note);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resUpdateAch);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " + receipt.GetDataKey());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
Console.WriteLine("PaymentType = " + receipt.GetPaymentType());
Console.WriteLine("Cust ID = " + receipt.GetResDataCustId());
Console.WriteLine("Phone = " + receipt.GetResDataPhone());
Console.WriteLine("Email = " + receipt.GetResDataEmail());
Console.WriteLine("Note = " + receipt.GetResDataNote());
Console.WriteLine("Sec = " + receipt.GetResDataSec());
Console.WriteLine("Cust First Name = " + receipt.GetResDataCustFirstName());
Console.WriteLine("Cust Last Name = " + receipt.GetResDataCustLastName());
Console.WriteLine("Cust Address 1 = " + receipt.GetResDataCustAddress1());
Console.WriteLine("Cust Address 2 = " + receipt.GetResDataCustAddress2());
Console.WriteLine("Cust City = " + receipt.GetResDataCustCity());
Console.WriteLine("Cust State = " + receipt.GetResDataCustState());
Console.WriteLine("Cust Zip = " + receipt.GetResDataCustZip());
Console.WriteLine("Routing Num = " + receipt.GetResDataRoutingNum());
Console.WriteLine("Masked Account Num = " + receipt.GetResDataMaskedAccountNum());
Console.WriteLine("Check Num = " + receipt.GetResDataCheckNum());
Console.WriteLine("Account Type = " + receipt.GetResDataAccountType());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.2.6 ResDelete

Note After a profile has been deleted, the details can no longer be retrieved.

ResDelete transaction object definition

```
ResDelete resDelete = new ResDelete(data_key);
```

HttpsPostRequest object for ResUpdateCC transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resDelete);
```

ResDelete transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 70: ResDelete transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alphanumeric	Not applicable (passed as argument)

Sample ResDelete - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaResDelete
{
public static void Main(string[] args)
{
string store_id = "store5";
string api_token = "yesguy";
string data_key = "PjVKjtEmc1FvFyjxHE4EwBMxi";
string processing_country_code = "CA";
bool status_check = false;
ResDelete resDelete = new ResDelete(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
```

Sample ResDelete - US

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAResDelete
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string data_key = "oJfm2psGzWFLsZEn6It42mMh4";
string processing_country_code = "US";
bool status_check = false;
ResDelete resDelete = new ResDelete(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
```

Sample ResDelete - CA

```
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resDelete);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();

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

Sample ResDelete - US

```
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resDelete);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
Console.WriteLine("Cust ID = " +
    receipt.GetResDataCustId());
Console.WriteLine("Phone = " +
    receipt.GetResDataPhone());
Console.WriteLine("Email = " +
    receipt.GetResDataEmail());
Console.WriteLine("Note = " +
    receipt.GetResDataNote());
Console.WriteLine("MaskedPan = " +
    receipt.GetResDataMaskedPan());
Console.WriteLine("Exp Date = " +
    receipt.GetResDataExpdate());
Console.WriteLine("Crypt Type = " +
    receipt.GetResDataCryptType());
Console.WriteLine("Avs Street Number = " +
    receipt.GetResDataAvsStreetNumber());
Console.WriteLine("Avs Street Name = " +
    receipt.GetResDataAvsStreetName());
Console.WriteLine("Avs Zipcode = " +
    receipt.GetResDataAvsZipcode());
Console.WriteLine("Presentation Type = " +
    receipt.GetResDataPresentationType());
Console.WriteLine("P Account Number = " +
    receipt.GetResDataPAccountNumber());
Console.WriteLine("Sec = " +
    receipt.GetResDataSec());
Console.WriteLine("Cust First Name = " +
    receipt.GetResDataCustFirstName());
Console.WriteLine("Cust Last Name = " +
    receipt.GetResDataCustLastName());
Console.WriteLine("Cust Address 1 = " +
    receipt.GetResDataCustAddress1());
```

Sample ResDelete - CA	Sample ResDelete - US
	`Console.WriteLine("Cust Address 2 = " + receipt.GetResDataCustAddress2());` `Console.WriteLine("Cust City = " + receipt.GetResDataCustCity());` `Console.WriteLine("Cust State = " + receipt.GetResDataCustState());` `Console.WriteLine("Cust Zip = " + receipt.GetResDataCustZip());` `Console.WriteLine("Routing Num = " + receipt.GetResDataRoutingNum());` `Console.WriteLine("Masked Account Num = " + receipt.GetResDataMaskedAccountNum());` `Console.WriteLine("Check Num = " + receipt.GetResDataCheckNum());` `Console.WriteLine("Account Type = " + receipt.GetResDataAccountType());` `Console.ReadLine();` `}` `catch (Exception e)` `{` `Console.WriteLine(e);` `}` `}` `}` `}`

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.2.7 ResLookupFull

ResLookupFull transaction object definition

```
ResLookupFull resLookupFull = new ResLookupFull(data_key);
```

HttpsPostRequest object for ResLookupFull transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resLookupFull);
```

ResLookupFull transaction values

Table 71: ResLookupFull transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alphanumeric	Not applicable (passed as argument)

Table 72: ResLookupFull transaction optional values

Value	Type	Limits	Set method
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Sample ResLookupFull - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaResLookupFull
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string data_key = "pi3ZMZoTTM8pLM9wuwws2KBxw";
string processing_country_code = "CA";
bool status_check = false;
ResLookupFull resLookupFull = new
    ResLookupFull(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resLookupFull);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
```

Sample ResLookupFull - US

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAResLookupFull
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string data_key = "AhcyWhamRPNnhyU8RYPxM3saK";
string processing_country_code = "US";
ResLookupFull resLookupFull = new
    ResLookupFull();
resLookupFull.SetData(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resLookupFull);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
Console.WriteLine("Cust ID = " +
```

[1]For more information, see Appendix C (page 280).

Sample ResLookupFull - CA	Sample ResLookupFull - US

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

Sample ResLookupFull - CA	Sample ResLookupFull - US
	`}` `}` `}`

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.2.8 ResLookupMasked

ResLookupMasked transaction object definition

```
ResLookupMasked resLookupMasked = new ResLookupMasked();
```

HttpsPostRequest object for ResLookupMasked transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resLookupMasked);
```

ResLookupMasked transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 73: ResLookupMasked transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alphanumeric	`resLookupMasked..SetData(data_key);`

Sample code

Sample ResLookupMasked - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaResLookupMasked
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string data_key = "pi3ZMZoTTM8pLM9wuwws2KBxw";
string processing_country_code = "CA";
bool status_check = false;
ResLookupMasked resLookupMasked = new
```

Sample ResLookupMasked - US

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAResLookupMasked
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string data_key = "AhcyWhamRPNnhyU8RYPxM3saK";
string processing_country_code = "US";
ResLookupMasked resLookupMasked = new
    ResLookupMasked();
```

Sample ResLookupMasked - CA

```
    ResLookupMasked();
resLookupMasked.SetData(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resLookupMasked);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
Console.WriteLine("Cust ID = " +
    receipt.GetResDataCustId());
Console.WriteLine("Phone = " +
    receipt.GetResDataPhone());
Console.WriteLine("Email = " +
    receipt.GetResDataEmail());
Console.WriteLine("Note = " +
    receipt.GetResDataNote());
Console.WriteLine("MaskedPan = " +
    receipt.GetResDataMaskedPan());
Console.WriteLine("Exp Date = " +
    receipt.GetResDataExpdate());
Console.WriteLine("Crypt Type = " +
    receipt.GetResDataCryptType());
Console.WriteLine("Avs Street Number = " +
    receipt.GetResDataAvsStreetNumber());
Console.WriteLine("Avs Street Name = " +
    receipt.GetResDataAvsStreetName());
Console.WriteLine("Avs Zipcode = " +
    receipt.GetResDataAvsZipcode());
Console.ReadLine();
}
catch (Exception e)
{
```

Sample ResLookupMasked - US

```
resLookupMasked.SetData(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resLookupMasked);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
Console.WriteLine("Cust ID = " +
    receipt.GetResDataCustId());
Console.WriteLine("Phone = " +
    receipt.GetResDataPhone());
Console.WriteLine("Email = " +
    receipt.GetResDataEmail());
Console.WriteLine("Note = " +
    receipt.GetResDataNote());
Console.WriteLine("MaskedPan = " +
    receipt.GetResDataMaskedPan());
Console.WriteLine("Exp Date = " +
    receipt.GetResDataExpdate());
Console.WriteLine("Crypt Type = " +
    receipt.GetResDataCryptType());
Console.WriteLine("Avs Street Number = " +
    receipt.GetResDataAvsStreetNumber());
Console.WriteLine("Avs Street Name = " +
    receipt.GetResDataAvsStreetName());
Console.WriteLine("Avs Zipcode = " +
    receipt.GetResDataAvsZipcode());
Console.WriteLine("Presentation Type = " +
    receipt.GetResDataPresentationType());
Console.WriteLine("P Account Number = " +
    receipt.GetResDataPAccountNumber());
Console.WriteLine("Sec = " +
```

Sample ResLookupMasked - CA	Sample ResLookupMasked - US

```
Console.WriteLine(e);
}
}
}
}
```
	```
	    receipt.GetResDataSec());
	Console.WriteLine("Cust First Name = " +
	    receipt.GetResDataCustFirstName());
	Console.WriteLine("Cust Last Name = " +
	    receipt.GetResDataCustLastName());
	Console.WriteLine("Cust Address 1 = " +
	    receipt.GetResDataCustAddress1());
	Console.WriteLine("Cust Address 2 = " +
	    receipt.GetResDataCustAddress2());
	Console.WriteLine("Cust City = " +
	    receipt.GetResDataCustCity());
	Console.WriteLine("Cust State = " +
	    receipt.GetResDataCustState());
	Console.WriteLine("Cust Zip = " +
	    receipt.GetResDataCustZip());
	Console.WriteLine("Routing Num = " +
	    receipt.GetResDataRoutingNum());
	Console.WriteLine("Masked Account Num = " +
	    receipt.GetResDataMaskedAccountNum());
	Console.WriteLine("Check Num = " +
	    receipt.GetResDataCheckNum());
	Console.WriteLine("Account Type = " +
	    receipt.GetResDataAccountType());
	Console.ReadLine();
	}
	catch (Exception e)
	{
	Console.WriteLine(e);
	}
	}
	}
	}
	```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.2.9 ResGetExpiring

ResGetExpiring transaction object definition

```
ResGetExpiring resGetExpiring = new ResGetExpiring();
```

HttpsPostRequest object for ResLookupFull transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resGetExpiring);
```

ResGetExpiring transaction values

ResGetExpiring transaction object mandatory values: None.

Sample code

Sample ResGetExpiring - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaResGetExpiring
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string processing_country_code = "CA";
bool status_check = false;
ResGetExpiring resGetExpiring = new
    ResGetExpiring();
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resGetExpiring);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
//ResolveData
foreach (string dataKey in receipt.GetDataKeys
    ())
{
Console.WriteLine("\nDataKey = " + dataKey);
Console.WriteLine("Payment Type = " +
    receipt.GetExpPaymentType(dataKey));
Console.WriteLine("Cust ID = " +
    receipt.GetExpCustId(dataKey));
```

Sample ResGetExpiring - US

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAResGetExpiring
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string processing_country_code = "US";
ResGetExpiring resGetExpiring = new
    ResGetExpiring();
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resGetExpiring);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
//ResolveData
foreach (string dataKey in receipt.GetDataKeys
    ())
{
Console.WriteLine("\nDataKey = " + dataKey);
Console.WriteLine("Payment Type = " +
    receipt.GetExpPaymentType(dataKey));
Console.WriteLine("Cust ID = " +
    receipt.GetExpCustId(dataKey));
Console.WriteLine("Phone = " +
    receipt.GetExpPhone(dataKey));
```

Sample ResGetExpiring - CA	Sample ResGetExpiring - US

`Console.WriteLine("Phone = " +`
`    receipt.GetExpPhone(dataKey));`
`Console.WriteLine("Email = " +`
`    receipt.GetExpEmail(dataKey));`
`Console.WriteLine("Note = " +`
`    receipt.GetExpNote(dataKey));`
`Console.WriteLine("Masked Pan = " +`
`    receipt.GetExpMaskedPan(dataKey));`
`Console.WriteLine("Exp Date = " +`
`    receipt.GetExpExpdate(dataKey));`
`Console.WriteLine("Crypt Type = " +`
`    receipt.GetExpCryptType(dataKey));`
`Console.WriteLine("Avs Street Number = " +`
`    receipt.GetExpAvsStreetNumber(dataKey));`
`Console.WriteLine("Avs Street Name = " +`
`    receipt.GetExpAvsStreetName(dataKey));`
`Console.WriteLine("Avs Zipcode = " +`
`    receipt.GetExpAvsZipCode(dataKey));`
`}`
`Console.ReadLine();`
`}`
`catch (Exception e)`
`{`
`Console.WriteLine(e);`
`}`
`}`
`}`
`}`
	`Console.WriteLine("Email = " +`
	`    receipt.GetExpEmail(dataKey));`
	`Console.WriteLine("Note = " +`
	`    receipt.GetExpNote(dataKey));`
	`Console.WriteLine("Masked Pan = " +`
	`    receipt.GetExpMaskedPan(dataKey));`
	`Console.WriteLine("Exp Date = " +`
	`    receipt.GetExpExpdate(dataKey));`
	`Console.WriteLine("Crypt Type = " +`
	`    receipt.GetExpCryptType(dataKey));`
	`Console.WriteLine("Avs Street Number = " +`
	`    receipt.GetExpAvsStreetNumber(dataKey));`
	`Console.WriteLine("Avs Street Name = " +`
	`    receipt.GetExpAvsStreetName(dataKey));`
	`Console.WriteLine("Avs Zipcode = " +`
	`    receipt.GetExpAvsZipCode(dataKey));`
	`Console.WriteLine("Presentation Type = " +`
	`    receipt.GetExpPresentationType(dataKey));`
	`Console.WriteLine("P Account Number = " +`
	`    receipt.GetExpPAccountNumber(dataKey));`
	`}`
	`Console.ReadLine();`
	`}`
	`catch (Exception e)`
	`{`
	`Console.WriteLine(e);`
	`}`
	`}`
	`}`
	`}`

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.2.10 ResIscorporateCard

ResIscorporateCard transaction object definition

```
ResIscorporatecard resIscorporatecard = new ResIscorporatecard();
```

HttpsPostRequest object for ResIscorporateCard transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resIscorporatecard);
```

ResIscorporateCard transaction values

Table 74: ResIscorporateCard transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alphanumeric	`resIscorporatecard..SetData(data_key);`

Table 75: ResIscorporateCard transaction optional values

Value	Type	Limits	Set method
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Sample code

Sample ResIscorporatecard - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaResIscorporatecard
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string data_key = "eLqsADfwqHDxIpJG9vLnELx01";
string processing_country_code = "CA";
bool status_check = false;
ResIscorporatecard resIscorporatecard = new
    ResIscorporatecard();
resIscorporatecard.SetData(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resIscorporatecard);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();

try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("CorporateCard = " +
```

Sample ResIscorporatecard - US

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAResIscorporatecard
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string data_key = "jh01NcJELdIohSVqKRdhQtNbl";
string processing_country_code = "US";
bool status_check = false;
ResIscorporatecard resIscorporatecard = new
    ResIscorporatecard();
resIscorporatecard.SetData(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resIscorporatecard);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();

try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("CorporateCard = " +
```

[1]For more information, see Appendix C (page 280).

Sample ResIscorporatecard - CA	Sample ResIscorporatecard - US
`    receipt.GetCorporateCard());`	`    receipt.GetCorporateCard());`
`Console.WriteLine("ResponseCode = " +`	`Console.WriteLine("ResponseCode = " +`
`    receipt.GetResponseCode());`	`    receipt.GetResponseCode());`
`Console.WriteLine("Message = " +`	`Console.WriteLine("Message = " +`
`    receipt.GetMessage());`	`    receipt.GetMessage());`
`Console.WriteLine("TransDate = " +`	`Console.WriteLine("TransDate = " +`
`    receipt.GetTransDate());`	`    receipt.GetTransDate());`
`Console.WriteLine("TransTime = " +`	`Console.WriteLine("TransTime = " +`
`    receipt.GetTransTime());`	`    receipt.GetTransTime());`
`Console.WriteLine("Complete = " +`	`Console.WriteLine("Complete = " +`
`    receipt.GetComplete());`	`    receipt.GetComplete());`
`Console.WriteLine("TimedOut = " +`	`Console.WriteLine("TimedOut = " +`
`    receipt.GetTimedOut());`	`    receipt.GetTimedOut());`
`Console.WriteLine("ResSuccess = " +`	`Console.WriteLine("ResSuccess = " +`
`    receipt.GetResSuccess());`	`    receipt.GetResSuccess());`
`Console.WriteLine("PaymentType = " +`	`Console.WriteLine("PaymentType = " +`
`    receipt.GetPaymentType());`	`    receipt.GetPaymentType());`
`Console.ReadLine();`	`Console.ReadLine();`
`}`	`}`
`catch (Exception e)`	`catch (Exception e)`
`{`	`{`
`Console.WriteLine(e);`	`Console.WriteLine(e);`
`}`	`}`
`}`	`}`
`}`	`}`
`}`	`}`

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.2.11 ResAddToken

ResAddToken transaction object definition

```
ResAddToken resAddToken = new ResAddToken(data_key, crypt_type);
```

HttpsPostRequest object for ResAddToken transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resAddToken);
```

ResAddToken transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 76: ResAddToken transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alphanumeric	`resAddToken.SetData(data_key);`
E-commerce indicator	String	1-character alphanumeric[1]	`resAddToken.SetCryptType(crypt);`

Table 77: ResAddToken transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`resAddToken`
AVS information	Object	Not applicable. See Appendix E (page 288).	`resAddToken.SetAvsInfo(avsCheck);`
Email address	String	30-character alphanumeric	`resAddToken.SetEmail(email);`
Phone number	String	30-character alphanumeric	`resAddToken.SetPhone(phone);`
Note	String	30-character alphanumeric	`resAddToken.SetNote(note);`

Sample ResAddToken - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaResAddToken
{
public static void Main(string[] args)
{
string store_id = "moneris";
string api_token = "hurgle";
string data_key = "ot-
    A8R8m9sjsUgltcyTIDNmOVuq9";
string expdate = "1602";
string phone = "0000000000";
string email = "bob@smith.com";
string note = "my note";
string cust_id = "customer1";
string crypt_type = "7";
string processing_country_code = "CA";
bool status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
```

Sample ResAddToken - US

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAResAddToken
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string data_key = "ot-
    70QJjCy90DokPVYUvWnVWAgoV";
string phone = "0000000000";
string email = "bob@smith.com";
string note = "my note";
string cust_id = "customer1";
string crypt_type = "7";
string processing_country_code = "US";
bool status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
```

[1]Full explanation on page 259

Sample ResAddToken - CA

```
avsCheck.SetAvsZipCode("M1M1M1");
ResAddToken resAddToken = new ResAddToken
    (data_key, crypt_type);
resAddToken.SetExpdate(expdate);
resAddToken.SetCustId(cust_id);
resAddToken.SetPhone(phone);
resAddToken.SetEmail(email);
resAddToken.SetNote(note);
resAddToken.SetAvsInfo(avsCheck);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resAddToken);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
Console.WriteLine("Cust ID = " +
    receipt.GetResDataCustId());
Console.WriteLine("Phone = " +
    receipt.GetResDataPhone());
Console.WriteLine("Email = " +
    receipt.GetResDataEmail());
Console.WriteLine("Note = " +
    receipt.GetResDataNote());
Console.WriteLine("MaskedPan = " +
    receipt.GetResDataMaskedPan());
Console.WriteLine("Exp Date = " +
    receipt.GetResDataExpdate());
Console.WriteLine("Crypt Type = " +
    receipt.GetResDataCryptType());
Console.WriteLine("Avs Street Number = " +
    receipt.GetResDataAvsStreetNumber());
Console.WriteLine("Avs Street Name = " +
```

Sample ResAddToken - US

```
ResAddToken resAddToken = new ResAddToken();
resAddToken.SetCustId(cust_id);
resAddToken.SetPhone(phone);
resAddToken.SetEmail(email);
resAddToken.SetNote(note);
resAddToken.SetAvsInfo(avsCheck);
resAddToken.SetData(data_key);
resAddToken.SetCryptType(crypt_type);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resAddToken);
mpgReq.Send();

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

Sample ResAddToken - CA	Sample ResAddToken - US

`    receipt.GetResDataAvsStreetName());`
`Console.WriteLine("Avs Zipcode = " +`
`    receipt.GetResDataAvsZipcode());`
`Console.ReadLine();`
`}`
`catch (Exception e)`
`{`
`Console.WriteLine(e);`
`}`
`}`
`}`
`}`
	`Console.WriteLine("Avs Zipcode = " +`
	`    receipt.GetResDataAvsZipcode());`
	`Console.ReadLine();`
	`}`
	`catch (Exception e)`
	`{`
	`Console.WriteLine(e);`
	`}`
	`}`
	`}`
	`}`

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.2.12 ResTokenizeCC

Basic transactions that can be tokenized are:

- Purchase
- Preauthorization
- Capture
- Reauth
- Refund
- Purchase Correction
- Independent Refund.

The tokenization process is outlined in Figure 4 .

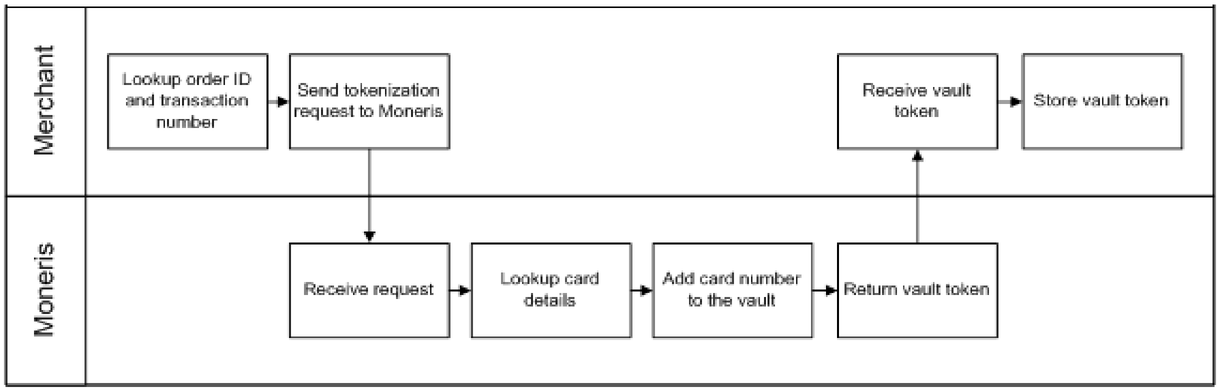


Figure 4: Tokenize process diagram

ResTokenizeCC transaction object definition

```
ResTokenizeCC resTokenizeCC = new ResTokenizeCC();
```

HttpsPostRequest object for ResTokenizeCC transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.SetTransaction(resTokenizeCC);
```

ResTokenizeCC transaction values

Table 78: ResTokenizeCC transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`resTokenizeCC.SetOrderId(order_id);`
Transaction number[1]	String	255-character alphanumeric	`resTokenizeCC.SetTxnNumber(txn_number);`

These mandatory values reference a previously processed credit card financial transaction. The credit card number, expiry date, and crypt type from the original transaction are registered in the Vault for future financial Vault transactions.

Table 79: ResTokenizeCC transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`resTokenizeCC`
Email address	String	30-character alphanumeric	`resTokenizeCC.SetEmail(email);`
Phone number	String	30-character alphanumeric	`resTokenizeCC.SetPhone(phone);`
Note	String	30-character alphanumeric	`resTokenizeCC.SetNote(note);`
AVS inform- ation	Object	Not applicable. See Appendix E (page 288).	

9.3 Financial Transactions

After a financial transaction is complete, the response fields indicate all the values that are currently saved under the profile that was used.

9.3.1 Customer ID Changes

Some financial transactions take the customer ID as an optional value. The customer ID may or may not already be in the Vault profile when the transaction is sent. Therefore, it is possible to change the value of the customer ID by performing a financial transaction

Table 80shows what the customer ID will be in the response field after a financial transaction is performed.

[1]The transaction number is a response field of the original transaction that you are now tokenizing.

Table 80: Customer ID use in response fields

Already in profile?	Passed in?	Version used in response
No	No	Customer ID not used in transaction
No	Yes	Passed in
Yes	No	Profile
Yes	Yes	Passed in

9.3.2 ResPurchaseCC

ResPurchaseCC transaction object definition

```
ResPurchaseCC resPurchaseCC = new ResPurchaseCC();
```

HttpsPostRequest object for ResPurchaseCC transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resPurchaseCC);
```

ResPurchaseCC transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 81: ResPurchaseCC transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alphanumeric	`resPurchaseCC.SetData(data_key);`
Order ID	String	50-character alphanumeric	`resPurchaseCC.SetOrderId(order_id);`
Amount	String	9-character decimal	`resPurchaseCC.SetAmount(amount);`
E-commerce indicator	String	1-character alphanumeric[1]	`resPurchaseCC.SetCryptType(crypt);`

Table 82: ResPurchaseCC transaction optional values

Value	Type	Limits	Set method
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

[1]Full explanation on page 259

[2]For more information, see Appendix C (page 280).

Table 82: ResPurchaseCC transaction optional values

Value	Type	Limits	Set method
Expiry date[1]	String	4-character numeric YYMM format. (Note that this is reversed from the date displayed on the card, which is MMYY)	`resPurchaseCC.SetExpdate(expdate);`
Customer ID	String	50-character alphanumeric	`resPurchaseCC`
Dynamic descriptor	String	20-character alphanumeric[2]	`resPurchaseCC.SetDynamicDescriptor (dynamic_descriptor);`
Customer information	Object	Not applicable. See Section Appendix D (page 282).	`resPurchaseCC.SetCustInfo (customer);`
AVS inform-ation	Object	Not applicable. See Appendix E (page 288).	`resPurchaseCC.SetAvsInfo (avsCheck);`
CVD inform-ation	Object	Not applicable. See Appendix F (page 294) .	`resPurchaseCC.SetCvdInfo (cvdCheck);`
Recurring billing	Object	Not applicable. See Section Appendix G (page 297).	`resPurchaseCC.SetRecur(recurring_ cycle);`

Sample ResPurchaseCC - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaResPurchaseCC
{
public static void Main(string[] args)
{
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string store_id = "store1";
string api_token = "yesguy";
string data_key = "eLqsADfwqHDxIpJG9vLnELx01";
string amount = "1.00";
string cust_id = "customer1"; //if sent will
    be submitted, otherwise cust_id from
    profile will be used
string crypt_type = "1";
string descriptor = "my descriptor";
```

Sample ResPurchaseCC - US

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAResPurchaseCC
{
public static void Main(string[] args)
{
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string store_id = "monusqa002";
string api_token = "qatoken";
string data_key = "5rnXvoHdrJPJ6DwZlSqKH3pFo";
string amount = "1.00";
string cust_id = "customer1"; //if sent will
    be submitted, otherwise cust_id from
    profile will be used
string crypt_type = "1";
string descriptor = "my descriptor";
```

[1]For temporary tokens only (see "Charging a Temporary Token" on page 110).

[2]See "Definition of Request Fields" (page 258) for proper length definition

Sample ResPurchaseCC - CA

```
string processing_country_code = "CA";
bool status_check = false;
ResPurchaseCC resPurchaseCC = new
    ResPurchaseCC();
resPurchaseCC.SetData(data_key);
resPurchaseCC.SetOrderId(order_id);
resPurchaseCC.SetCustId(cust_id);
resPurchaseCC.SetAmount(amount);
resPurchaseCC.SetCryptType(crypt_type);
resPurchaseCC.SetDynamicDescriptor
    (descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resPurchaseCC);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
Console.WriteLine("IsVisaDebit = " +
```

Sample ResPurchaseCC - US

```
string processing_country_code = "US";
bool status_check = false;
ResPurchaseCC resPurchaseCC = new
    ResPurchaseCC();
resPurchaseCC.SetData(data_key);
resPurchaseCC.SetOrderId(order_id);
resPurchaseCC.SetCustId(cust_id);
resPurchaseCC.SetAmount(amount);
resPurchaseCC.SetCryptType(crypt_type);
resPurchaseCC.SetDynamicDescriptor
    (descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resPurchaseCC);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
Console.WriteLine("IsVisaDebit = " +
```

Sample ResPurchaseCC - CA	Sample ResPurchaseCC - US

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

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.3.3 ResPurchaseACH

ResPurchaseACH transaction object definition

```
ResPurchaseAch resPurchaseAch = new ResPurchaseAch();
```

HttpsPostRequest object for ResPurchaseACH transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resPurchaseAch);
```

ResPurchaseACH transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 83: ResPurchaseACH transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alphanumeric	`resPurchaseAch.SetData(data_key);`
Order ID	String	50-character alphanumeric	`resPurchaseAch.SetOrderId(order_id);`
Amount	String	9-character decimal	`resPurchaseAch.SetAmount(amount);`

Table 84: ResPurchaseACH transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`resPurchaseAch`
Customer inform-ation	Object	Not applicable. See Section Appendix D (page 282).	`resPurchaseAch.SetCustInfo (customer);`
Recurring billing	Object	Not applicable. See Section Appendix G (page 297).	`resPurchaseAch.SetRecur(recur-ring_cycle);`

Sample ResPurchaseAch - US

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAResPurchaseAch
{
public static void Main(string[] args)
{
/********************** Request Variables ****************************/
String store_id = "monusqa002";
String api_token = "qatoken";
/********************** Transaction Variables ************************/
string order_id = "Test" + DateTime.Now.ToString("yyyyMMddhhmmss");
String data_key = "0HjrtlV2VCu4DRRv8zwZmjbJk";
String cust_id = "Hilton_1";
String amount = "1.00";
String processing_country_code = "US";
/************************ Request Object ******************************/
ResPurchaseAch resPurchaseAch = new ResPurchaseAch();
resPurchaseAch.SetDataKey(data_key);
resPurchaseAch.SetOrderId(order_id);
resPurchaseAch.SetCustId(cust_id);
resPurchaseAch.SetAmount(amount);

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resPurchaseAch);
mpgReq.Send();
/************************ Receipt Object ******************************/
```

Sample ResPurchaseAch - US

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

Console.WriteLine("DataKey = " + receipt.GetDataKey());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " + receipt.GetResSuccess());
Console.WriteLine("PaymentType = " + receipt.GetPaymentType() + "\n");
Console.WriteLine("Cust ID = " + receipt.GetResCustId());
Console.WriteLine("Phone = " + receipt.GetResPhone());
Console.WriteLine("Email = " + receipt.GetResEmail());
Console.WriteLine("Note = " + receipt.GetResNote());
Console.WriteLine("Sec = " + receipt.GetResSec());
Console.WriteLine("Cust First Name = " + receipt.GetResCustFirstName());
Console.WriteLine("Cust Last Name = " + receipt.GetResCustLastName());
Console.WriteLine("Cust Address1 = " + receipt.GetResCustAddress1());
Console.WriteLine("Cust Address2 = " + receipt.GetResCustAddress2());
Console.WriteLine("Cust City = " + receipt.GetResCustCity());
Console.WriteLine("Cust State = " + receipt.GetResCustState());
Console.WriteLine("Cust Zip = " + receipt.GetResCustZip());
Console.WriteLine("Routing Num = " + receipt.GetResRoutingNum());
Console.WriteLine("Account Num = " + receipt.GetResAccountNum());
Console.WriteLine("Masked Account Num = " + receipt.GetResMaskedAccountNum());
Console.WriteLine("Check Num = " + receipt.GetResCheckNum());
Console.WriteLine("Account Type = " + receipt.GetResAccountType());
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.3.4 ResPreauthCC

ResPreauthCC transaction object definition

```
ResPreauthCC resPreauthCC = new ResPreauthCC();
```

HttpsPostRequest object for ResPreauthCC transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resPreauthCC);
```

ResPreauthCC transaction values

Table 1: ResPreauthCC transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25- character alphanumeric	`resPreauthCC.SetData(data_key);`
Order ID	String	50-character alphanumeric	`resPreauthCC.SetOrderId(order_id);`
Amount	String	9-character decimal	`resPreauthCC.SetAmount(amount);`
E-com-merce indicator	String	1-character alphanumeric[1]	`resPreauthCC.SetCryptType(crypt);`

Table 2: ResPreauthCC transaction optional values

Value	Type	Limits	Set method
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck (status_check);`
Expiry date[3]	String	4-character alphanumeric (YYMM format)	`resPreauthCC.SetExpdate (expdate);`
Customer ID	String	50-character alphanumeric	`resPreauthCC`
Customer information	Object	Not applicable. See Section Appendix D (page 282).	`resPreauthCC.SetCustInfo (customer);`
AVS inform-ation	Object	Not applicable. See Appendix E (page 288).	`resPreauthCC.SetAvsInfo (avsCheck);`
CVD inform-ation	Object	Not applicable. See Appendix F (page 294).	`resPreauthCC.SetCvdInfo (cvdCheck);`

Sample ResPreauthCC - CA	Sample ResPreauthCC - US

`namespace Moneris`
`{`	

[1]Full explanation on page 259

[2]For more information, see Appendix C (page 280).

[3]For temporary tokens only (see "Charging a Temporary Token" on page 110).

Sample ResPreauthCC - CA	Sample ResPreauthCC - US
```
using System;
using System.Text;
using System.Collections;
public class TestCanadaResPreauthCC
{
public static void Main(string[] args)
{
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string store_id = "store1";
string api_token = "yesguy";
string data_key = "YeMnLZ8i2p02gbwSB8i8Q02Fo";
string amount = "1.00";
string cust_id = "customer1"; //if sent will
    be submitted, otherwise cust_id from
    profile will be used
string crypt_type = "1";
string dynamic_descriptor = "my descriptor";
string processing_country_code = "CA";
bool status_check = false;
ResPreauthCC resPreauthCC = new ResPreauthCC
    ();
resPreauthCC.SetData(data_key);
resPreauthCC.SetOrderId(order_id);
resPreauthCC.SetCustId(cust_id);
resPreauthCC.SetAmount(amount);
resPreauthCC.SetCryptType(crypt_type);
resPreauthCC.SetDynamicDescriptor(dynamic_
    descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resPreauthCC);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
```

Sample ResPreauthCC - CA	Sample ResPreauthCC - US
```
    receipt.GetTransTime());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
Console.WriteLine("IsVisaDebit = " +
    receipt.GetIsVisaDebit());
Console.WriteLine("Cust ID = " +
    receipt.GetResDataCustId());
Console.WriteLine("Phone = " +
    receipt.GetResDataPhone());
Console.WriteLine("Email = " +
    receipt.GetResDataEmail());
Console.WriteLine("Note = " +
    receipt.GetResDataNote());
Console.WriteLine("Masked Pan = " +
    receipt.GetResDataMaskedPan());
Console.WriteLine("Exp Date = " +
    receipt.GetResDataExpdate());
Console.WriteLine("Crypt Type = " +
    receipt.GetResDataCryptType());
Console.WriteLine("Avs Street Number = " +
    receipt.GetResDataAvsStreetNumber());
Console.WriteLine("Avs Street Name = " +
    receipt.GetResDataAvsStreetName());
Console.WriteLine("Avs Zipcode = " +
    receipt.GetResDataAvsZipcode());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.3.5 Vault Independent Refund - ResIndRefundCC

ResIndRefundCC transaction object definition

```
ResIndRefundCC resIndRefundCC = new ResIndRefundCC();
```

HttpsPostRequest object for ResIndRefundCC transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resIndRefundCC);
```

ResIndRefundCC transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 85: ResIndRefundCC transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alphanumeric	`resIndRefundCC.SetData(data_key);`
Order ID	String	50-character alphanumeric	`resIndRefundCC.SetOrderId(order_id);`
Amount	String	9-character decimal	`resIndRefundCC.SetAmount(amount);`
E-commerce indicator	String	1-character alphanumeric[1]	`resIndRefundCC.SetCryptType(crypt);`

Table 86: ResIndRefundCC transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`resIndRefundCC.SetCustId(cust_id);`
Expiry date[2]	String	4-character alpha-numeric (YYMM format)	`resIndRefundCC.SetExpdate(expdate);`
Status[3] Check[4]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`
Dynamic descriptor	String	20-character alpha-numeric[5]	`resIndRefundCC.SetDynamicDescriptor (dynamic_descriptor);`

[1]Full explanation on page 259

[2]For temporary tokens only (see "Charging a Temporary Token" on page 110).

[3]Status Check applies to Canadian integrations only.

[4]For more information, see Appendix C (page 280).

[5]See "Definition of Request Fields" (page 258) for proper length definition

Sample ResIndRefundCC - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaResIndRefundCC
{
public static void Main(string[] args)
{
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string store_id = "store1";
string api_token = "yesguy";
string data_key = "qJD5kCZiCjsfabKH7WuxoHyZx";
string amount = "1.00";
string cust_id = "customer1";
string crypt_type = "1";
string processing_country_code = "CA";
bool status_check = false;
ResIndRefundCC resIndRefundCC = new
    ResIndRefundCC();
resIndRefundCC.SetOrderId(order_id);
resIndRefundCC.SetCustId(cust_id);
resIndRefundCC.SetAmount(amount);
resIndRefundCC.SetCryptType(crypt_type);
resIndRefundCC.SetData(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resIndRefundCC);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("TransType = " +
```

Sample ResIndRefuncCC - US

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAResIndRefundCC
{
public static void Main(string[] args)
{
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string store_id = "monusqa002";
string api_token = "qatoken";
string data_key = "DJWLDaVOv9XGjOVI0OXr8EIT4";
string amount = "1.00";
string cust_id = "customer1";
string crypt_type = "1";
string dynamic_descriptor = "123456";
string processing_country_code = "US";
ResIndRefundCC resIndRefundCC = new
    ResIndRefundCC();
resIndRefundCC.SetOrderId(order_id);
resIndRefundCC.SetCustId(cust_id);
resIndRefundCC.SetAmount(amount);
resIndRefundCC.SetCryptType(crypt_type);
resIndRefundCC.SetData(data_key);
resIndRefundCC.SetDynamicDescriptor(dynamic_
    descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(resIndRefundCC);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("DataKey = " +
    receipt.GetDataKey());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
```

Sample ResIndRefundCC - CA

```
    receipt.GetTransType());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
Console.WriteLine("IsVisaDebit = " +
    receipt.GetIsVisaDebit());
Console.WriteLine("Cust ID = " +
    receipt.GetResDataCustId());
Console.WriteLine("Phone = " +
    receipt.GetResDataPhone());
Console.WriteLine("Email = " +
    receipt.GetResDataEmail());
Console.WriteLine("Note = " +
    receipt.GetResDataNote());
Console.WriteLine("Masked Pan = " +
    receipt.GetResDataMaskedPan());
Console.WriteLine("Exp Date = " +
    receipt.GetResDataExpdate());
Console.WriteLine("Crypt Type = " +
    receipt.GetResDataCryptType());
Console.WriteLine("Avs Street Number = " +
    receipt.GetResDataAvsStreetNumber());
Console.WriteLine("Avs Street Name = " +
    receipt.GetResDataAvsStreetName());
Console.WriteLine("Avs Zipcode = " +
    receipt.GetResDataAvsZipcode());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Sample ResIndRefuncCC - US

```
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
Console.WriteLine("ResSuccess = " +
    receipt.GetResSuccess());
Console.WriteLine("PaymentType = " +
    receipt.GetPaymentType());
Console.WriteLine("Cust ID = " +
    receipt.GetResDataCustId());
Console.WriteLine("Phone = " +
    receipt.GetResDataPhone());
Console.WriteLine("Email = " +
    receipt.GetResDataEmail());
Console.WriteLine("Note = " +
    receipt.GetResDataNote());
Console.WriteLine("Masked Pan = " +
    receipt.GetResDataMaskedPan());
Console.WriteLine("Exp Date = " +
    receipt.GetResDataExpdate());
Console.WriteLine("Crypt Type = " +
    receipt.GetResDataCryptType());
Console.WriteLine("Avs Street Number = " +
    receipt.GetResDataAvsStreetNumber());
Console.WriteLine("Avs Street Name = " +
    receipt.GetResDataAvsStreetName());
Console.WriteLine("Avs Zipcode = " +
    receipt.GetResDataAvsZipcode());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.3.6 ResIndRefundAch

ResIndRefundAch transaction object definition

```
ResIndRefundAch resIndRefundAch = new ResIndRefundAch();
```

HttpsPostRequest object for ResIndRefundAch transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resIndRefundAch);
```

ResIndRefundAch transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 87: ResIndRefundAch transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alpha-numeric	`resIndRefundAch.SetData(data_key);`
Order ID	String	50-character alpha-numeric	`resIndRefundAch.SetOrderId(order_id);`
Amount	String	9-character decimal	`resIndRefundAch.SetAmount(amount);`

Table 88: ResIndRefundCC transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`resIndRefundAch`

Sample code

Sample ResIndRefundAch - US

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestUSAResIndRefundAch
{
public static void Main(string[] args)
{
string order_id = "Test" + DateTime.Now.ToString("yyyyMMddhhmmss");
string store_id = "monusqa002";
string api_token = "qatoken";
string data_key = "AhcyWhamRPNnhyU8RYPxM3saK";
string amount = "1.00";
string cust_id = "customer1";
string processing_country_code = "US";
ResIndRefundAch resIndRefundAch = new ResIndRefundAch();
resIndRefundAch.SetOrderId(order_id);
resIndRefundAch.SetCustId(cust_id);
resIndRefundAch.SetAmount(amount);
resIndRefundAch.SetData(data_key);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
```

Sample ResIndRefundAch - US

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

Vault response fields

For a list and explanation of (Receipt object) response fields that are available after sending this Vault transaction, see Appendix B Definition of Response Fields.

9.4 Hosted Tokenization

Moneris Hosted Tokenization (HT) is a solution for online e-commerce merchants who do not want to handle credit card numbers directly on their websites, yet want the ability to fully customize their check-out webpage appearance.

When an HT transaction is initiated, the Moneris Payment Gateway displays (on the merchant's behalf) a single text box on the merchant's check-out page. The cardholder can then securely enter the credit card information into the text box. Upon submission of the payment information on the checkout page, Moneris Payment Gateway returns a temporary token representing the credit card number to the merchant. This is then used in an API call to process a financial transaction directly with Moneris to charge the card. After receiving a response to the financial transaction, the merchant generates a receipt and allows the cardholder to continue with online shopping.

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

10 Mag Swipe Transaction Set

Mag Swipe transactions allow customers to swipe a credit card and submit the Track2 details.

These transactions support the submission of Track2 as well as a manual entry of the credit card number and expiry date. If all three fields are submitted, the Track2 details are used to process the transaction.

10.1 Mag Swipe Transaction Definitions

Purchase

Verifies funds on the customer’s card, removes the funds and prepares them for deposit into the merchant’s account.

Pre-Authorization

Verifies and locks funds on the customer’s credit card. The funds are locked for a specified amount of time based on the card issuer.

To retrieve the funds that have been locked by a Pre-Authorization transaction so that they may be settled in the merchant’s account, a Completion transaction must be performed. A Pre-Authorization may only be "completed" once.

Completion

Retrieves funds that have been locked (by a Mag Swipe Pre-Authorization transaction), and prepares them for settlement into the merchant’s account.

Force Post

Retrieves the locked funds and prepares them for settlement into the merchant’s account.

This is used when a merchant obtains the authorization number directly from the issuer by a third-party authorization method (such as by phone).

Purchase Correction

Restores the **full** amount of a previous Mag Swipe Purchase or Mag Swipe Completion transaction to the cardholder's card, and removes any record of it from the cardholder's statement. The order ID and transaction number from the original transaction are required, but the credit card does not need to be re-swiped.

This transaction can be used against a Purchase or Completion transaction that occurred same day provided that the batch containing the original transaction remains open. When using the automated closing feature, Batch Close occurs daily between 10 and 11 pm Eastern Time.

This transaction is sometimes referred to as "void".

Refund

Restores all or part of the funds from a Mag Swipe Purchase or Mag Swipe Completion transaction to the cardholder's card. Unlike a Purchase Correction, there is a record of the refund.

Independent Refund

Credits a specified amount to the cardholder's credit card.

This does not require a previous transaction (such as Mag Swipe Purchase) to be logged in the Moneris Payment Gateway. However, a credit card must be swiped to provide the Track2 data.

10.1.1 Encrypted Mag Swipe Transactions

Encrypted Mag Swipe transactions allow the customer to swipe or key in a credit card using a Moneris-provided encrypted mag swipe reader, and submit the encrypted Track2 details.

The encrypted mag swipe reader can be used for processing:

- Swiped card-present transactions
- Manually keyed card-present transactions
- Manually keyed card-not-present transactions.

Encrypted Mag Swipe transactions are identical to the regular Mag Swipe transactions from the customer's perspective. However, the card data must be swiped or keyed in via a Moneris-provided encrypted mag swipe reader. Contact Moneris for more details.

Only Mag Swipe Purchase and Mag Swipe Pre-Authorization have encrypted versions. Their explanations appear in this document as subsections of the regular (unencrypted) Mag Swipe Purchase and Mag Swipe Pre-Authorization transactions respectively.

10.2 Mag Swipe Purchase

Track2Purchase transaction object definition

```
Track2Purchase track2purchase = new Track2Purchase();
```

HttpsPostRequest object for Track2Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(track2purchase);
```

Mag Swipe Purchase transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 89: Track2Purchase transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`track2purchase.SetOrderId(order_id);`
Amount	String	9-character decimal	`track2purchase.SetAmount(amount);`

Table 89: Track2Purchase transaction object mandatory values (continued)

Value	Type	Limits	Set method

Credit card number
OR
Track2 data
	String	20-character numeric
		OR
		40-character numeric
			`track2purchase.SetPan(pan);`
			OR
			`track2purchase.SetTrack2(track2);`

Expiry date	String	4-character alpha-numeric (YYMM format)	`track2purchase.SetExpdate (expdate);`
POS code	String	2-character numeric	`track2purchase.SetPosCode(pos_ code);`

Table 90: Mag Swipe Purchase transaction optional values

Value	Type	Limits	Set method
AVS information	Object	Not applicable. See Appendix E (page 288).	`track2purchase.SetAvsInfo(avsCheck);`
Commcard invoice	String	17-character alpha-numeric	`track2purchase.SetCommcardInvoice (commcard_invoice);`
Commcard tax amount	String	9-character decimal	`track2purchase.SetCommcardTaxAmount (commcard_tax_amount);`
Customer ID	String	50-character alpha-numeric	`track2purchase.SetCustId(cust_id);`
CVD information	Object	Not applicable. See Section 1 (page 1).	`track2purchase.SetCvdInfo(cvdCheck);`
Dynamic descriptor	String	20-character alpha-numeric[1]	`track2purchase.SetDynamicDescriptor (dynamic_descriptor);`
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Sample Track2Purchase - CA	Sample Track2Purchase - US

`namespace Moneris`
`{`
`using System;`
`using System.Text.RegularExpressions;`
	`namespace Moneris`
	`{`
	`using System;`
	`using System.Text.RegularExpressions;`

[1]See "Definition of Request Fields" (page 258) for proper length definition.

[2]For more information, see Appendix C (page 280).

Sample Track2Purchase - CA	Sample Track2Purchase - US

`public class TestCanadaTrack2Purchase`
`{`
`public static void Main(string[] args)`
`{`
`string store_id = "store1";`
`string api_token = "yesguy";`
`string order_id = "Test" +`
`    DateTime.Now.ToString("yyyyMMddhhmmss");`
`string cust_id = "LBriggs";`
`string amount = "1.00";`
`string track2 = "";`
`//string track2 =`
`    ";5258968987035454=06061015454001060101?";`
`string pan = "4242424242424242";`
`string exp_date = "1903"; //must send '0000'`
`    if swiped`
`string pos_code = "00";`
`string commcard_invoice = "INV98798";`
`string commcard_tax_amount = "1.00";`
`string processing_country_code = "CA";`
`bool status_check = false;`
`Track2Purchase track2purchase = new`
`    Track2Purchase();`
`track2purchase.SetOrderId(order_id);`
`track2purchase.SetCustId(cust_id);`
`track2purchase.SetAmount(amount);`
`track2purchase.SetTrack2(track2);`
`track2purchase.SetPan(pan);`
`track2purchase.SetExpdate(exp_date);`
`track2purchase.SetPosCode(pos_code);`
`track2purchase.SetCommcardInvoice(commcard_`
`    invoice);`
`track2purchase.SetCommcardTaxAmount(commcard_`
`    tax_amount);`
`HttpsPostRequest mpgReq = new HttpsPostRequest`
`    ();`
`mpgReq.SetProcCountryCode(processing_country_`
`    code);`
`mpgReq.SetTestMode(true); //false or comment`
`    out this line for production transactions`
`mpgReq.SetStoreId(store_id);`
`mpgReq.SetApiToken(api_token);`
`mpgReq.SetTransaction(track2purchase);`
`mpgReq.SetStatusCheck(status_check);`
`mpgReq.Send();`
`try`
`{`
`Receipt receipt = mpgReq.GetReceipt();`
`Console.WriteLine("CardType = " +`
`    receipt.GetCardType());`
`Console.WriteLine("TransAmount = " +`
`    receipt.GetTransAmount());`
`Console.WriteLine("TxnNumber = " +`
`    receipt.GetTxnNumber());`
`Console.WriteLine("ReceiptId = " +`
`    receipt.GetReceiptId());`
`Console.WriteLine("TransType = " +`
`    receipt.GetTransType());`
	`public class TestUSATrack2Purchase`
	`{`
	`public static void Main(string[] args)`
	`{`
	`string store_id = "monusqa002";`
	`string api_token = "qatoken";`
	`string order_id = "Test" +`
	`    DateTime.Now.ToString("yyyyMMddhhmmss");`
	`string cust_id = "LBriggs";`
	`string amount = "1.00";`
	`string track2 =`
	`    ";5258968987035454=06061015454001060101?";`
	`string pan = "";`
	`string exp = ""; //must send '0000' if swiped`
	`string pos_code = "00";`
	`string commcard_invoice = "INV98798";`
	`string commcard_tax_amount = "1.00";`
	`string descriptor = "my descriptor";`
	`string processing_country_code = "US";`
	`bool status_check = false;`
	`Track2Purchase track2purchase = new`
	`    Track2Purchase();`
	`track2purchase.SetOrderId(order_id);`
	`track2purchase.SetCustId(cust_id);`
	`track2purchase.SetAmount(amount);`
	`track2purchase.SetTrack2(track2);`
	`track2purchase.SetPan(pan);`
	`track2purchase.SetExpdate(exp);`
	`track2purchase.SetPosCode(pos_code);`
	`track2purchase.SetDynamicDescriptor`
	`    (descriptor);`
	`track2purchase.SetCommcardInvoice(commcard_`
	`    invoice);`
	`track2purchase.SetCommcardTaxAmount(commcard_`
	`    tax_amount);`
	`HttpsPostRequest mpgReq = new HttpsPostRequest`
	`    ();`
	`mpgReq.SetProcCountryCode(processing_country_`
	`    code);`
	`mpgReq.SetTestMode(true); //false or comment`
	`    out this line for production transactions`
	`mpgReq.SetStoreId(store_id);`
	`mpgReq.SetApiToken(api_token);`
	`mpgReq.SetTransaction(track2purchase);`
	`mpgReq.SetStatusCheck(status_check);`
	`mpgReq.Send();`
	`try`
	`{`
	`Receipt receipt = mpgReq.GetReceipt();`
	`Console.WriteLine("CardType = " +`
	`    receipt.GetCardType());`
	`Console.WriteLine("TransAmount = " +`
	`    receipt.GetTransAmount());`
	`Console.WriteLine("TxnNumber = " +`
	`    receipt.GetTxnNumber());`
	`Console.WriteLine("ReceiptId = " +`
	`    receipt.GetReceiptId());`
	`Console.WriteLine("TransType = " +`

Sample Track2Purchase - CA	Sample Track2Purchase - US

`Console.WriteLine("ReferenceNum = " +`
`    receipt.GetReferenceNum());`
`Console.WriteLine("ResponseCode = " +`
`    receipt.GetResponseCode());`
`Console.WriteLine("BankTotals = " +`
`    receipt.GetBankTotals());`
`Console.WriteLine("Message = " +`
`    receipt.GetMessage());`
`Console.WriteLine("AuthCode = " +`
`    receipt.GetAuthCode());`
`Console.WriteLine("Complete = " +`
`    receipt.GetComplete());`
`Console.WriteLine("TransDate = " +`
`    receipt.GetTransDate());`
`Console.WriteLine("TransTime = " +`
`    receipt.GetTransTime());`
`Console.WriteLine("Ticket = " +`
`    receipt.GetTicket());`
`Console.WriteLine("TimedOut = " +`
`    receipt.GetTimedOut());`
`//Console.WriteLine("StatusCode = " +`
`    receipt.GetStatusCode());`
`//Console.WriteLine("StatusMessage = " +`
`    receipt.GetStatusMessage());`
`Console.ReadLine();`
`}`
`catch (Exception e)`
`{`
`Console.WriteLine(e);`
`}`
`}`
`}`
`}`
	`    receipt.GetTransType());`
	`Console.WriteLine("ReferenceNum = " +`
	`    receipt.GetReferenceNum());`
	`Console.WriteLine("ResponseCode = " +`
	`    receipt.GetResponseCode());`
	`Console.WriteLine("BankTotals = " +`
	`    receipt.GetBankTotals());`
	`Console.WriteLine("Message = " +`
	`    receipt.GetMessage());`
	`Console.WriteLine("AuthCode = " +`
	`    receipt.GetAuthCode());`
	`Console.WriteLine("Complete = " +`
	`    receipt.GetComplete());`
	`Console.WriteLine("TransDate = " +`
	`    receipt.GetTransDate());`
	`Console.WriteLine("TransTime = " +`
	`    receipt.GetTransTime());`
	`Console.WriteLine("Ticket = " +`
	`    receipt.GetTicket());`
	`Console.WriteLine("TimedOut = " +`
	`    receipt.GetTimedOut());`
	`//Console.WriteLine("StatusCode = " +`
	`    receipt.GetStatusCode());`
	`//Console.WriteLine("StatusMessage = " +`
	`    receipt.GetStatusMessage());`
	`Console.ReadLine();`
	`}`
	`catch (Exception e)`
	`{`
	`Console.WriteLine(e);`
	`}`
	`}`
	`}`
	`}`

10.2.1 Encrypted Mag Swipe Purchase

EncTrack2Purchase transaction object definition

```
EncTrack2Purchase encpurchase = new EncTrack2Purchase();
```

HttpsPostRequest object for EncTrack2Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(encpurchase);
```

Encrypted Mag Swipe Purchase transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 91: EncTrack2Purchase transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`encpurchase.SetOrderId (order_id);`
Amount	String	9-character decimal	`encpurchase.SetAmount (amount);`
Encrypted Track2 data	String	40-character numeric	`encpurchase.SetEncTrack2 (enc_track2);`
POS code	String	2-character numeric	`encpurchase.SetPosCode(pos_ code);`
Device type	String	TBD	`encpurchase.SetDeviceType (device_type);`

Table 92: EncTrack2Purchase transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`encpurchase`
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`
AVS inform-ation	Object	Not applicable. See Appendix E (page 288).	`encpurchase.SetAvsInfo(avsCheck);`
Dynamic descriptor	String	20-character alpha-numeric[2]	`encpurchase.SetDynamicDescriptor (dynamic_descriptor);`

Sample EncTrack2Purchase - CA

```
namespace Moneris
{
using System;
using System.Text.RegularExpressions;
public class TestCanadaEncTrack2Purchase
{
public static void Main(string[] args)
{
string store_id = "store5";
string api_token = "yesguy";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "LBriggs";
string amount = "1.00";
string pos_code = "00";
```

Sample EncTrack2Purchase - US

```
namespace Moneris
{
using System;
using System.Text.RegularExpressions;
public class TestUSAEncTrack2Purchase
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "LBriggs";
string amount = "1.00";
string pos_code = "00";
```

[1]For more information, see Appendix C (page 280).

[2]See "Definition of Request Fields" (page 258) for proper length definition

Sample EncTrack2Purchase - CA

```
string device_type = "idtech_bdk";
string processing_country_code = "CA";
bool status_check = false;
string dynamic_descriptor = "my descriptor";
string enc_track2 =
"02D901801F4F2800039B%*4924********4030^TESTCA
    RD/MONERIS
    ^***************************************"
+
    "**?*;4924********4030=*******************
    *?*A7150C78335A5024949516FDA9A68A91C4FBAB1
    279DD1DE2283D"
+
    "BEBB2C6B3FDEACF7B5B314219D76C00890F347A96
    40EFE90023E31622F5FD95C14C0362DD2EAB28ADEB
    46B8B577DA1A1"
+
    "8B707BCC7E48068EFF1882CFB4B369BDC4BB646C8
    70D6083239860B23837EA91DB3F1D8AD066DAAACE2
    B2DA18D563E4F"
+
    "1EF997696337B8999E9C707DEC4CB0410B887291C
    AF2EE449573D01613484B80760742A3506C3141593
    9320000A00028"
+ "3C5E03";
EncTrack2Purchase encpurchase = new
    EncTrack2Purchase();
encpurchase.SetOrderId(order_id);
encpurchase.SetCustId(cust_id);
encpurchase.SetAmount(amount);
encpurchase.SetEncTrack2(enc_track2);
encpurchase.SetPosCode(pos_code);
encpurchase.SetDeviceType(device_type);
encpurchase.SetDynamicDescriptor(dynamic_
    descriptor);
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
encpurchase.SetAvsInfo(avsCheck);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(encpurchase);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
```

Sample EncTrack2Purchase - US

```
string device_type = "idtech";
string processing_country_code = "US";
bool status_check = false;
string dynamic_descriptor = "my descriptor";
string enc_track2 =
"02D901801F4F2800039B%*4924********4030^TESTCA
    RD/MONERIS
    ^***************************************"
+
    "**?*;4924********4030=*******************
    *?*A7150C78335A5024949516FDA9A68A91C4FBAB1
    279DD1DE2283D"
+
    "BEBB2C6B3FDEACF7B5B314219D76C00890F347A96
    40EFE90023E31622F5FD95C14C0362DD2EAB28ADEB
    46B8B577DA1A1"
+
    "8B707BCC7E48068EFF1882CFB4B369BDC4BB646C8
    70D6083239860B23837EA91DB3F1D8AD066DAAACE2
    B2DA18D563E4F"
+
    "1EF997696337B8999E9C707DEC4CB0410B887291C
    AF2EE449573D01613484B80760742A3506C3141593
    9320000A00028"
+ "3C5E03";

EncTrack2Purchase encpurchase = new
    EncTrack2Purchase();
encpurchase.SetOrderId(order_id);
encpurchase.SetCustId(cust_id);
encpurchase.SetAmount(amount);
encpurchase.SetEncTrack2(enc_track2);
encpurchase.SetPosCode(pos_code);
encpurchase.SetDeviceType(device_type);
encpurchase.SetDynamicDescriptor(dynamic_
    descriptor);
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
encpurchase.SetAvsInfo(avsCheck);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(encpurchase);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
```

Sample EncTrack2Purchase - CA	Sample EncTrack2Purchase - US
`Console.WriteLine("TxnNumber = " +`	`    receipt.GetTransAmount());`
`    receipt.GetTxnNumber());`	`Console.WriteLine("TxnNumber = " +`
`Console.WriteLine("ReceiptId = " +`	`    receipt.GetTxnNumber());`
`    receipt.GetReceiptId());`	`Console.WriteLine("ReceiptId = " +`
`Console.WriteLine("TransType = " +`	`    receipt.GetReceiptId());`
`    receipt.GetTransType());`	`Console.WriteLine("TransType = " +`
`Console.WriteLine("ReferenceNum = " +`	`    receipt.GetTransType());`
`    receipt.GetReferenceNum());`	`Console.WriteLine("ReferenceNum = " +`
`Console.WriteLine("ResponseCode = " +`	`    receipt.GetReferenceNum());`
`    receipt.GetResponseCode());`	`Console.WriteLine("ResponseCode = " +`
`Console.WriteLine("BankTotals = " +`	`    receipt.GetResponseCode());`
`    receipt.GetBankTotals());`	`Console.WriteLine("BankTotals = " +`
`Console.WriteLine("Message = " +`	`    receipt.GetBankTotals());`
`    receipt.GetMessage());`	`Console.WriteLine("Message = " +`
`Console.WriteLine("AuthCode = " +`	`    receipt.GetMessage());`
`    receipt.GetAuthCode());`	`Console.WriteLine("AuthCode = " +`
`Console.WriteLine("Complete = " +`	`    receipt.GetAuthCode());`
`    receipt.GetComplete());`	`Console.WriteLine("Complete = " +`
`Console.WriteLine("TransDate = " +`	`    receipt.GetComplete());`
`    receipt.GetTransDate());`	`Console.WriteLine("TransDate = " +`
`Console.WriteLine("TransTime = " +`	`    receipt.GetTransDate());`
`    receipt.GetTransTime());`	`Console.WriteLine("TransTime = " +`
`Console.WriteLine("Ticket = " +`	`    receipt.GetTransTime());`
`    receipt.GetTicket());`	`Console.WriteLine("Ticket = " +`
`Console.WriteLine("TimedOut = " +`	`    receipt.GetTicket());`
`    receipt.GetTimedOut());`	`Console.WriteLine("TimedOut = " +`
`Console.WriteLine("MaskedPan = " +`	`    receipt.GetTimedOut());`
`    receipt.GetMaskedPan());`	`Console.WriteLine("MaskedPan = " +`
`Console.WriteLine("CardLevelResult = " +`	`    receipt.GetMaskedPan());`
`    receipt.GetCardLevelResult());`	`Console.WriteLine("CardLevelResult = " +`
`Console.WriteLine("AVS Response = " +`	`    receipt.GetCardLevelResult());`
`    receipt.GetAvsResultCode());`	`Console.WriteLine("AVS Response = " +`
`}`	`    receipt.GetAvsResultCode());`
`catch (Exception e)`	`}`
`{`	`catch (Exception e)`
`Console.WriteLine(e);`	`{`
`}`	`Console.WriteLine(e);`
`}`	`}`
`}`	`}`
`}`	`}`
	`}`

10.3 Mag Swipe Pre-Authorization

Track2PreAuth transaction object definition

```
Track2PreAuth track2preauth = new Track2PreAuth();
```

HttpsPostRequest object for Track2PreAuth transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(track2preauth);
```

Mag Swipe Pre-Authorization transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 93: Track2PreAuth transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`track2preauth.SetOrderId (order_id);`
Amount	String	9-character decimal	`track2preauth.SetAmount (amount);`
Credit card number OR Track2 data	String	20-character numeric OR 40-character numeric	`track2preauth.SetPan(pan);` OR `track2preauth.SetPan(pan);`
Expiry date	String	4-character alphanumeric (YYMM format)	`track2preauth.SetExpdate (expdate);`
POS code	String	2-character numeric	`track2preauth.SetPosCode (pos_code);`

Table 94: Mag Swipe Pre-Authoriation transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`track2preauth.SetCustId(cust_id);`
Dynamic descriptor	String	20-character alpha-numeric[1]	`track2preauth.SetDynamicDescriptor (dynamic_descriptor);`
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`
Commcard invoice[3]	String	17-character alpha-numeric	`track2preauth.SetCommcardInvoice (commcard_invoice);`
Commcard tax amount[4]	String	9-character decimal	`track2preauth.SetCommcardTaxAmount (commcard_tax_amount);`

Sample code

Sample Mag Swipe Pre-Authorization - CA	Sample Mag Swipe Pre-Authorization - US

`namespace Moneris`
`{`
	`namespace Moneris`
	`{`

[1]See "Definition of Request Fields" (page 258) for proper length definition

[2]For more information, see Appendix C (page 280).

[3]Available to US integrations only.

[4]Available to US integrations only.

Sample Mag Swipe Pre-Authorization - CA

```
using System;
using System.Text.RegularExpressions;
public class TestCanadaTrack2Preauth
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "LBriggs";
string amount = "5.00";
//string track2 =
    ";5258968987035454=06061015454001060101?";
string track2 = "";
string pan = "4242424242424242";
string exp = "1906"; //must send '0000' if
    swiped
string pos_code = "00";
string processing_country_code = "CA";
bool status_check = false;
Track2PreAuth track2preauth = new
    Track2PreAuth();
track2preauth.SetOrderId(order_id);
track2preauth.SetCustId(cust_id);
track2preauth.SetAmount(amount);
track2preauth.SetTrack2(track2);
track2preauth.SetPan(pan);
track2preauth.SetExpdate(exp);
track2preauth.SetPosCode(pos_code);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(track2preauth);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
```

Sample Mag Swipe Pre-Authorization - US

```
using System;
using System.Text.RegularExpressions;
public class TestUSATrack2Preauth
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "LBriggs";
string amount = "5.00";
string track2 =
    ";5258968987035454=06061015454001060101?";
string pan = null;
string exp = "0000"; //must send '0000' if
    swiped
string pos_code = "00";
string commcard_invoice = "INV98798";
string commcard_tax_amount = "1.00";
string descriptor = "my descriptor";
string processing_country_code = "US";
bool status_check = false;
Track2PreAuth track2preauth = new
    Track2PreAuth();
track2preauth.SetOrderId(order_id);
track2preauth.SetCustId(cust_id);
track2preauth.SetAmount(amount);
track2preauth.SetTrack2(track2);
track2preauth.SetPan(pan);
track2preauth.SetExpdate(exp);
track2preauth.SetPosCode(pos_code);
track2preauth.SetDynamicDescriptor
    (descriptor);
track2preauth.SetCommcardInvoice(commcard_
    invoice);
track2preauth.SetCommcardTaxAmount(commcard_
    tax_amount);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(track2preauth);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
```

Sample Mag Swipe Pre-Authorization - CA

```
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
//Console.WriteLine("StatusCode = " +
    receipt.GetStatusCode());
//Console.WriteLine("StatusMessage = " +
    receipt.GetStatusMessage());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

Sample Mag Swipe Pre-Authorization - US

```
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
//Console.WriteLine("StatusCode = " +
    receipt.GetStatusCode());
//Console.WriteLine("StatusMessage = " +
    receipt.GetStatusMessage());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

10.3.1 Encrypted Mag Swipe Pre-Authorization

EncTrack2Preauth transaction object definition

```
EncTrack2PreAuth enctrack2preauth = new EncTrack2PreAuth();
```

HttpsPostRequest object for EncTrack2Preauth transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(enctrack2preauth);
```

Encrypted Mag Swipe Pre-Authorization transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 95: EncTrack2Preauth transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`enctrack2preauth.SetOrderId (order_id);`
Amount	String	9-character decimal	`enctrack2preauth.SetAmount (amount);`

Credit card number
OR
Track2
	String	20-character numeric
		OR
		40-character numeric
			`enctrack2preauth.SetPan(pan);`
			OR
			`enctrack2preauth.SetTrack2 (track2);`

Expiry date	String	4-character alphanumeric (YYMM format)	`enctrack2preauth.SetExpdate (expdate);`
POS code	String	2-character numeric	`enctrack2preauth.SetPosCode (pos_code);`
Device type	String	TBD	`enctrack2preauth.SetDeviceType (device_type);`

Table 96: EncTrack2Preauth transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`enctrack2preauth.SetCustId (cust_id);`
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Sample code

Sample Encrypted Mag Swipe Preauth - CA	Sample Encrypted Mag Swipe Preauth - US

```
namespace Moneris
{
using System;
using System.Text.RegularExpressions;
public class TestCanadaEncTrack2Preauth
{
public static void Main(string[] args)
{
string store_id = "store5";
string api_token = "yesguy";
string order_id = "Test" +
```
	```
	namespace Moneris
	{
	using System;
	using System.Text.RegularExpressions;
	public class TestUSAEncTrack2Preauth
	{
	public static void Main(string[] args)
	{
	string store_id = "monusqa002";
	string api_token = "qatoken";
	string order_id = "Test" +
	```

[1]For more information, see Appendix C (page 280).

Sample Encrypted Mag Swipe Preauth - CA

```
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "LBriggs";
string amount = "5.00";
string pos_code = "00";
string device_type = "idtech_bdk";
string processing_country_code = "CA";
bool status_check = false;
string enc_track2 = "ENCRYPTEDTRACK2DATA";
string descriptor = "nqa";

EncTrack2PreAuth enctrack2preauth = new
    EncTrack2PreAuth();
enctrack2preauth.SetOrderId(order_id);
enctrack2preauth.SetCustId(cust_id);
enctrack2preauth.SetAmount(amount);
enctrack2preauth.SetEncTrack2(enc_track2);
enctrack2preauth.SetPosCode(pos_code);
enctrack2preauth.SetDeviceType(device_type);
enctrack2preauth.SetDynamicDescriptor
    (descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(enctrack2preauth);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
```

Sample Encrypted Mag Swipe Preauth - US

```
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "LBriggs";
string amount = "5.00";
string pos_code = "00";
string device_type = "idtech";
string processing_country_code = "US";
bool status_check = false;
string enc_track2 = "my encrypted data";
EncTrack2PreAuth enctrack2preauth = new
    EncTrack2PreAuth();
enctrack2preauth.SetOrderId(order_id);
enctrack2preauth.SetCustId(cust_id);
enctrack2preauth.SetAmount(amount);
enctrack2preauth.SetEncTrack2(enc_track2);
enctrack2preauth.SetPosCode(pos_code);
enctrack2preauth.SetDeviceType(device_type);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(enctrack2preauth);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
```

Sample Encrypted Mag Swipe Preauth - CA	Sample Encrypted Mag Swipe Preauth - US

`Console.WriteLine("TransDate = " + receipt.GetTransDate());`
`Console.WriteLine("TransTime = " + receipt.GetTransTime());`
`Console.WriteLine("Ticket = " + receipt.GetTicket());`
`Console.WriteLine("TimedOut = " + receipt.GetTimedOut());`
`Console.WriteLine("MaskedPan = " + receipt.GetMaskedPan());`
`Console.WriteLine("CardLevelResult = " + receipt.GetCardLevelResult());`
`}`
`catch (Exception e)`
`{`
`Console.WriteLine(e);`
`}`
`}`
`}`
`}`
	`Console.WriteLine("Ticket = " + receipt.GetTicket());`
	`Console.WriteLine("TimedOut = " + receipt.GetTimedOut());`
	`Console.WriteLine("MaskedPan = " + receipt.GetMaskedPan());`
	`Console.WriteLine("CardLevelResult = " + receipt.GetCardLevelResult());`
	`Console.ReadLine();`
	`}`
	`catch (Exception e)`
	`{`
	`Console.WriteLine(e);`
	`}`
	`}`
	`}`
	`}`

10.4 Mag Swipe Completion

Track2Completion transaction object definition

```
Track2Completion track2completion = new Track2Completion();
```

HttpsPostRequest object for Track2Completion transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(track2completion);
```

Mag Swipe Completion transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 97: Track2Completion transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`track2completion.SetOrderId(order_id);`
Transaction number	String	255-character variable character	`track2completion.SetTxnNumber(txn_number);`
Amount	String	9-character decimal	`track2completion.SetAmount(amount);`
POS code	String	2-character numeric	`track2completion.SetPosCode(pos_code);`

Table 98: Mag Swipe Completion transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`track2completion`
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`
Dynamic descriptor	String	20-character alphanumeric[2]	`track2completion.SetDynamicDescriptor (dynamic_descriptor);`
Commcard invoice[3]	String	17-character alphanumeric	`track2completion.SetCommcardInvoice (commcard_invoice);`
Commcard tax amount[4]	String	9-character decimal	`track2completion.SetCommcardTaxAmount (commcard_tax_amount);`

Sample Mag Swipe Completion - CA

```
namespace Moneris
{
using System;
public class TestCanadaTrack2Completion
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string order_id = "Test20150625035422";
string txn_number = "87028-0_10";
string amount = "1.00";
string pos_code = "00";
string dynamic_descriptor = "123456";
string processing_country_code = "CA";
bool status_check = false;
Track2Completion track2completion = new
    Track2Completion();
track2completion.SetOrderId(order_id);
track2completion.SetTxnNumber(txn_number);
track2completion.SetAmount(amount);
track2completion.SetPosCode(pos_code);
track2completion.SetDynamicDescriptor(dynamic_
    descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
```

Sample Mag Swipe Completion - US

```
namespace Moneris
{
using System;
public class TestUSATrack2Completion
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string order_id = "Test169976210";
string txn_number = "106667-0_25";
string amount = "1.00";
string pos_code = "00";
string commcard_invoice = "INVC090";
string commcard_tax_amount = "1.00";
string dynamic_descriptor = "123456";
string cust_id = "my customer id";
string processing_country_code = "US";
bool status_check = false;
Track2Completion track2completion = new
    Track2Completion();
track2completion.SetOrderId(order_id);
track2completion.SetCustId(cust_id);
track2completion.SetTxnNumber(txn_number);
track2completion.SetAmount(amount);
track2completion.SetPosCode(pos_code);
track2completion.SetCommcardInvoice(commcard_
    invoice);
track2completion.SetCommcardTaxAmount
    (commcard_tax_amount);
```

[1]For more information, see Appendix C (page 280).

[2]See "Definition of Request Fields" (page 258) for proper length definition

[3]Available to US integrations only.

[4]Available to US integrations only.

Sample Mag Swipe Completion - CA	Sample Mag Swipe Completion - US

`mpgReq.SetStoreId(store_id);`
`mpgReq.SetApiToken(api_token);`
`mpgReq.SetTransaction(track2completion);`
`mpgReq.SetStatusCheck(status_check);`
`mpgReq.Send();`
`try`
`{`
`Receipt receipt = mpgReq.GetReceipt();`
`Console.WriteLine("CardType = " +`
`    receipt.GetCardType());`
`Console.WriteLine("TransAmount = " +`
`    receipt.GetTransAmount());`
`Console.WriteLine("TxnNumber = " +`
`    receipt.GetTxnNumber());`
`Console.WriteLine("ReceiptId = " +`
`    receipt.GetReceiptId());`
`Console.WriteLine("TransType = " +`
`    receipt.GetTransType());`
`Console.WriteLine("ReferenceNum = " +`
`    receipt.GetReferenceNum());`
`Console.WriteLine("ResponseCode = " +`
`    receipt.GetResponseCode());`
`Console.WriteLine("ISO = " + receipt.GetISO`
`    ());`
`Console.WriteLine("BankTotals = " +`
`    receipt.GetBankTotals());`
`Console.WriteLine("Message = " +`
`    receipt.GetMessage());`
`Console.WriteLine("AuthCode = " +`
`    receipt.GetAuthCode());`
`Console.WriteLine("Complete = " +`
`    receipt.GetComplete());`
`Console.WriteLine("TransDate = " +`
`    receipt.GetTransDate());`
`Console.WriteLine("TransTime = " +`
`    receipt.GetTransTime());`
`Console.WriteLine("Ticket = " +`
`    receipt.GetTicket());`
`Console.WriteLine("TimedOut = " +`
`    receipt.GetTimedOut());`
`//Console.WriteLine("StatusCode = " +`
`    receipt.GetStatusCode());`
`//Console.WriteLine("StatusMessage = " +`
`    receipt.GetStatusMessage());`
`Console.ReadLine();`
`}`
`catch (Exception e)`
`{`
`Console.WriteLine(e);`
`}`
`}`
`}`
`}`
	`track2completion.SetDynamicDescriptor(dynamic_`
	`    descriptor);`
	`HttpsPostRequest mpgReq = new HttpsPostRequest`
	`    ();`
	`mpgReq.SetProcCountryCode(processing_country_`
	`    code);`
	`mpgReq.SetTestMode(true); //false or comment`
	`    out this line for production transactions`
	`mpgReq.SetStoreId(store_id);`
	`mpgReq.SetApiToken(api_token);`
	`mpgReq.SetTransaction(track2completion);`
	`mpgReq.SetStatusCheck(status_check);`
	`mpgReq.Send();`
	`try`
	`{`
	`Receipt receipt = mpgReq.GetReceipt();`
	`Console.WriteLine("CardType = " +`
	`    receipt.GetCardType());`
	`Console.WriteLine("TransAmount = " +`
	`    receipt.GetTransAmount());`
	`Console.WriteLine("TxnNumber = " +`
	`    receipt.GetTxnNumber());`
	`Console.WriteLine("ReceiptId = " +`
	`    receipt.GetReceiptId());`
	`Console.WriteLine("TransType = " +`
	`    receipt.GetTransType());`
	`Console.WriteLine("ReferenceNum = " +`
	`    receipt.GetReferenceNum());`
	`Console.WriteLine("ResponseCode = " +`
	`    receipt.GetResponseCode());`
	`Console.WriteLine("ISO = " + receipt.GetISO`
	`    ());`
	`Console.WriteLine("BankTotals = " +`
	`    receipt.GetBankTotals());`
	`Console.WriteLine("Message = " +`
	`    receipt.GetMessage());`
	`Console.WriteLine("AuthCode = " +`
	`    receipt.GetAuthCode());`
	`Console.WriteLine("Complete = " +`
	`    receipt.GetComplete());`
	`Console.WriteLine("TransDate = " +`
	`    receipt.GetTransDate());`
	`Console.WriteLine("TransTime = " +`
	`    receipt.GetTransTime());`
	`Console.WriteLine("Ticket = " +`
	`    receipt.GetTicket());`
	`Console.WriteLine("TimedOut = " +`
	`    receipt.GetTimedOut());`
	`//Console.WriteLine("StatusCode = " +`
	`    receipt.GetStatusCode());`
	`//Console.WriteLine("StatusMessage = " +`
	`    receipt.GetStatusMessage());`
	`Console.ReadLine();`
	`}`
	`catch (Exception e)`
	`{`
	`Console.WriteLine(e);`
	`}`

Sample Mag Swipe Completion - CA	Sample Mag Swipe Completion - US
	`}` `}` `}`

10.5 Mag Swipe Force Post

Track2ForcePost transaction object definition

```
Track2ForcePost track2forcePost = new Track2ForcePost();
```

HttpsPostRequest object for Track2ForcePost transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(track2forcePost);
```

Mag Swipe Force Post transaction mandatory arguments

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 99: Track2ForcePost transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`track2forcePost.SetOrderId (order_id);`
Amount	String	9-character decimal	`track2forcePost.SetAmount (amount);`

Credit card number
OR
Track2 data
	String	20-character numeric
		OR
		40-character numeric
			`track2forcePost.SetPan(pan);`
			OR
			`track2forcePost.SetTrack2 (track2);`

Expiry date	String	4-character alphanumeric (YYMM format)	`track2forcePost.SetExpdate (expdate);`
POS code	String	2-character numeric	`track2forcePost.SetPosCode(pos_ code);`
Authorization code	String	8-character alphanumeric	`track2forcePost.SetAuthCode (auth_code);`

Table 100: Mag Swipe Force Post transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`track2forcePost.SetCustId(cust_id);`
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Sample code

Sample Mag Swipe Force Post - CA

```
namespace Moneris
{
using System;
public class TestCanadaTrack2ForcePost
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string amount = "10.00";
string track2 = "";
string pan = "4242424242424242";
string expiry_date = "1212";
string pos_code = "00";
string auth_code = "AU4R6";
string processing_country_code = "CA";
bool status_check = false;
Track2ForcePost track2forcePost = new
    Track2ForcePost();
track2forcePost.SetOrderId(order_id);
track2forcePost.SetAmount(amount);
track2forcePost.SetTrack2(track2);
track2forcePost.SetPan(pan);
track2forcePost.SetExpdate(expiry_date);
track2forcePost.SetPosCode(pos_code);
track2forcePost.SetAuthCode(auth_code);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(track2forcePost);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
```

Sample Mag Swipe Force Post - US

```
namespace Moneris
{
using System;
public class TestUSATrack2ForcePost
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string amount = "10.00";
string track2 = "";
string pan = "4242424242424242";
string expiry_date = "1212";
string pos_code = "00";
string auth_code = "AU4R6";
string processing_country_code = "US";
bool status_check = false;
Track2ForcePost track2forcePost = new
    Track2ForcePost();
track2forcePost.SetOrderId(order_id);
track2forcePost.SetAmount(amount);
track2forcePost.SetTrack2(track2);
track2forcePost.SetPan(pan);
track2forcePost.SetExpdate(expiry_date);
track2forcePost.SetPosCode(pos_code);
track2forcePost.SetAuthCode(auth_code);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(track2forcePost);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
```

[1]For more information, see Appendix C (page 280).

Sample Mag Swipe Force Post - CA

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

Sample Mag Swipe Force Post - US

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

10.6 Mag Swipe Purchase Correction

Track2PurchaseCorrection transaction object definition

```
Track2PurchaseCorrection track2void = new Track2PurchaseCorrection();
```

HttpsPostRequest object for Track2PurchaseCorrection transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(track2void);
```

Mag Swipe Purchase Correction transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 101: Track2PurchaseCorrection transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`track2purchasecorrection.SetOrderId (order_id);`
Transaction number	String	255-character alpha-numeric	`track2purchasecorrection.SetTxnNumber (txn_number);`

Table 102: Mag Swipe Purchase Correction transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`track2purchasecorrection`
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`
Dynamic descriptor	String	20-character alphanumeric [2]	`track2purchasecorrection.SetDynamicDescriptor (dynamic_descriptor);`

Sample Mag Swipe Purchase Correction - CA

```
namespace Moneris
{
using System;
public class
    TestCanadaTrack2PurchaseCorrection
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string order_id = "Test20150625030621";
string txn_number = "86949-0_10";
string dynamic_descriptor = "123456";
string cust_id = "my customer id";
string processing_country_code = "CA";
bool status_check = false;
Track2PurchaseCorrection track2void = new
    Track2PurchaseCorrection();
track2void.SetOrderId(order_id);
track2void.SetCustId(cust_id);
track2void.SetTxnNumber(txn_number);
track2void.SetDynamicDescriptor(dynamic_
```

Sample Mag Swipe Purchase Correction - US

```
namespace Moneris
{
using System;
public class TestUSATrack2PurchaseCorrection
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string order_id = "mvt2713975506";
string txn_number = "911707-0_10";
string dynamic_descriptor = "123456";
string cust_id = "my customer id";
string processing_country_code = "US";
bool status_check = false;
Track2PurchaseCorrection track2void = new
    Track2PurchaseCorrection();
track2void.SetOrderId(order_id);
track2void.SetTxnNumber(txn_number);
track2void.SetCustId(cust_id);
track2void.SetDynamicDescriptor(dynamic_
    descriptor);
```

[1]For more information, see Appendix C (page 280).

[2]See "Definition of Request Fields" (page 258) for proper length definition

Sample Mag Swipe Purchase Correction - CA	Sample Mag Swipe Purchase Correction - US

```
    descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(track2void);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
//Console.WriteLine("StatusCode = " +
    receipt.GetStatusCode());
//Console.WriteLine("StatusMessage = " +
    receipt.GetStatusMessage());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
```
	```
	HttpsPostRequest mpgReq = new HttpsPostRequest
	    ();
	mpgReq.SetProcCountryCode(processing_country_
	    code);
	mpgReq.SetTestMode(true); //false or comment
	    out this line for production transactions
	mpgReq.SetStoreId(store_id);
	mpgReq.SetApiToken(api_token);
	mpgReq.SetTransaction(track2void);
	mpgReq.SetStatusCheck(status_check);
	mpgReq.Send();
	try
	{
	Receipt receipt = mpgReq.GetReceipt();
	Console.WriteLine("CardType = " +
	    receipt.GetCardType());
	Console.WriteLine("TransAmount = " +
	    receipt.GetTransAmount());
	Console.WriteLine("TxnNumber = " +
	    receipt.GetTxnNumber());
	Console.WriteLine("ReceiptId = " +
	    receipt.GetReceiptId());
	Console.WriteLine("TransType = " +
	    receipt.GetTransType());
	Console.WriteLine("ReferenceNum = " +
	    receipt.GetReferenceNum());
	Console.WriteLine("ResponseCode = " +
	    receipt.GetResponseCode());
	Console.WriteLine("ISO = " + receipt.GetISO
	    ());
	Console.WriteLine("BankTotals = " +
	    receipt.GetBankTotals());
	Console.WriteLine("Message = " +
	    receipt.GetMessage());
	Console.WriteLine("AuthCode = " +
	    receipt.GetAuthCode());
	Console.WriteLine("Complete = " +
	    receipt.GetComplete());
	Console.WriteLine("TransDate = " +
	    receipt.GetTransDate());
	Console.WriteLine("TransTime = " +
	    receipt.GetTransTime());
	Console.WriteLine("Ticket = " +
	    receipt.GetTicket());
	Console.WriteLine("TimedOut = " +
	    receipt.GetTimedOut());
	//Console.WriteLine("StatusCode = " +
	    receipt.GetStatusCode());
	//Console.WriteLine("StatusMessage = " +
	    receipt.GetStatusMessage());
	Console.ReadLine();
	}
	catch (Exception e)
	{
	Console.WriteLine(e);
	}
	}
	}
	```

Sample Mag Swipe Purchase Correction - CA	Sample Mag Swipe Purchase Correction - US
`}` `}`	`}`

10.7 Mag Swipe Refund

Track2Refundtransaction object definition

```
Track2Refund track2refund = new Track2Refund();
```

HttpsPostRequest object for Track2Refund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(track2refund);
```

Mag Swipe Refund transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 103: Track2Refund transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`track2refund.SetOrderId (order_id);`
Amount	String	9-character decimal	`track2refund.SetAmount (amount);`
Transaction number	String	255-character alphanumeric	`track2refund.SetTxnNumber (txn_number);`

Table 104: Mag Swipe Refund transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`track2refund`
Status Check[1]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`
Dynamic descriptor	String	20-character alpha-numeric[2]	`track2refund.SetDynamicDescriptor (dynamic_descriptor);`

[1]For more information, see Appendix C (page 280).

[2]See "Definition of Request Fields" (page 258) for proper length definition

Sample Mag Swipe Refund - CA

```
namespace Moneris
{
using System;
public class TestCanadaTrack2Refund
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string order_id = "Test20150625035152"; //will
    prompt user for input
string txn_number = "87017-0_10";
string amount = "1.00";
string dynamic_descriptor = "123456";
string cust_id = "customer id";
string processing_country_code = "CA";
bool status_check = false;
Track2Refund track2refund = new Track2Refund
    ();
track2refund.SetOrderId(order_id);
track2refund.SetAmount(amount);
track2refund.SetCustId(cust_id);
track2refund.SetTxnNumber(txn_number);
track2refund.SetDynamicDescriptor(dynamic_
    descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(track2refund);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
```

Sample Mag Swipe Refund - US

```
namespace Moneris
{
using System;
public class TestUSATrack2Refund
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string order_id = "Test706209401"; //will
    prompt user for input
string txn_number = "106665-0_25";
string amount = "1.00";
string dynamic_descriptor = "123456";
string processing_country_code = "US";
bool status_check = false;
Track2Refund track2refund = new Track2Refund
    ();
track2refund.SetOrderId(order_id);
track2refund.SetAmount(amount);
track2refund.SetTxnNumber(txn_number);
track2refund.SetDynamicDescriptor(dynamic_
    descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(track2refund);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
```

Sample Mag Swipe Refund - CA	Sample Mag Swipe Refund - US

```
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
    receipt.GetTimedOut());
//Console.WriteLine("StatusCode = " +
    receipt.GetStatusCode());
//Console.WriteLine("StatusMessage = " +
    receipt.GetStatusMessage());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```
	```
	    receipt.GetAuthCode());
	Console.WriteLine("Complete = " +
	    receipt.GetComplete());
	Console.WriteLine("TransDate = " +
	    receipt.GetTransDate());
	Console.WriteLine("TransTime = " +
	    receipt.GetTransTime());
	Console.WriteLine("Ticket = " +
	    receipt.GetTicket());
	Console.WriteLine("TimedOut = " +
	    receipt.GetTimedOut());
	//Console.WriteLine("StatusCode = " +
	    receipt.GetStatusCode());
	//Console.WriteLine("StatusMessage = " +
	    receipt.GetStatusMessage());
	Console.ReadLine();
	}
	catch (Exception e)
	{
	Console.WriteLine(e);
	}
	}
	}
	}
	```

10.8 Mag Swipe Independent Refund

Note	If you receive a TRANSACTION NOT ALLOWED error, it may mean the Mag Swipe Independent Refund transaction is not supported on your account. Contact Moneris to have it temporarily (re-)enabled.

Track2IndependentRefund transaction object definition

```
Track2IndependentRefund track2indrefund = new Track2IndependentRefund();
```

HttpsPostRequest object for Track2IndependentRefund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(track2indrefund);
```

Mag Swipe Independent Refund transaction values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 105: Mag Swipe Independent Refund transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alphanumeric	`track2indrefund.SetOrderId(order_id);`
Amount	String	9-character decimal	`track2indrefund.SetAmount(amount);`
Credit card number	String	20-character numeric	`track2indrefund.SetPan(pan);`
Track2 data	String	40-character numeric	`track2indrefund.SetTrack2(track2);`
Expiry date	String	4-character alphanumeric (YYMM format)	`track2indrefund.SetExpdate(expdate);`
POS code	String	2-character numeric	`track2indrefund.SetPosCode(pos_code);`

Table 106: Mag Swipe Independent Refund transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alphanumeric	`track2indrefund.SetCustId(cust_id);`
Dynamic descriptor	String	20-character alphanumeric[1]	`track2indrefund.SetDynamicDescriptor (dynamic_descriptor);`
Status Check[2]	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Sample Mag Swipe Independent Refund - CA

```
namespace Moneris
{
using System;
public class TestCanadaTrack2IndependentRefund
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "Ced_Benson32";
string amount = "5.00";
string track2 = "";
string pan = "4242424242424242";
string exp = "1903"; //must send '0000' if
```

Sample Mag Swipe Independent Refund - US

```
namespace Moneris
{
using System;
public class TestUSATrack2IndependentRefund
{
public static void Main(string[] args)
{
string store_id = "monusqa002";
string api_token = "qatoken";
string order_id = "Test" +
    DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "Ced_Benson32";
string amount = "5.00";
string track2 =
    ";5258968987035454=06061015454001060101?";
string pan = "";
```

[1]See "Definition of Request Fields" (page 258) for proper length definition

[2]For more information, see Appendix C (page 280).

Sample Mag Swipe Independent Refund - CA

```
    swiped
string pos_code = "00";
string processing_country_code = "CA";
bool status_check = false;
Track2IndependentRefund track2indrefund = new
    Track2IndependentRefund();
track2indrefund.SetOrderId(order_id);
track2indrefund.SetCustId(cust_id);
track2indrefund.SetAmount(amount);
track2indrefund.SetTrack2(track2);
track2indrefund.SetPan(pan);
track2indrefund.SetExpdate(exp);
track2indrefund.SetPosCode(pos_code);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(track2indrefund);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
```

Sample Mag Swipe Independent Refund - US

```
string exp_date = "0000"; //YYMM format
string pos_code = "00";
string processing_country_code = "US";
bool status_check = false;
Track2IndependentRefund track2indrefund = new
    Track2IndependentRefund();
track2indrefund.SetOrderId(order_id);
track2indrefund.SetCustId(cust_id);
track2indrefund.SetAmount(amount);
track2indrefund.SetTrack2(track2);
track2indrefund.SetPan(pan);
track2indrefund.SetExpdate(exp_date);
track2indrefund.SetPosCode(pos_code);
HttpsPostRequest mpgReq = new HttpsPostRequest
    ();
mpgReq.SetProcCountryCode(processing_country_
    code);
mpgReq.SetTestMode(true); //false or comment
    out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(track2indrefund);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " +
    receipt.GetCardType());
Console.WriteLine("TransAmount = " +
    receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " +
    receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " +
    receipt.GetReceiptId());
Console.WriteLine("TransType = " +
    receipt.GetTransType());
Console.WriteLine("ReferenceNum = " +
    receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " +
    receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO
    ());
Console.WriteLine("BankTotals = " +
    receipt.GetBankTotals());
Console.WriteLine("Message = " +
    receipt.GetMessage());
Console.WriteLine("AuthCode = " +
    receipt.GetAuthCode());
Console.WriteLine("Complete = " +
    receipt.GetComplete());
Console.WriteLine("TransDate = " +
    receipt.GetTransDate());
Console.WriteLine("TransTime = " +
    receipt.GetTransTime());
Console.WriteLine("Ticket = " +
    receipt.GetTicket());
Console.WriteLine("TimedOut = " +
```

Sample Mag Swipe Independent Refund - CA	Sample Mag Swipe Independent Refund - US

`    receipt.GetTimedOut());`
`//Console.WriteLine("StatusCode = " +`
`    receipt.GetStatusCode());`
`//Console.WriteLine("StatusMessage = " +`
`    receipt.GetStatusMessage());`
`Console.ReadLine();`
`}`
`catch (Exception e)`
`{`
`Console.WriteLine(e);`
`}`
`}`
`}`
`}`
	`    receipt.GetTimedOut());`
	`//Console.WriteLine("StatusCode = " +`
	`    receipt.GetStatusCode());`
	`//Console.WriteLine("StatusMessage = " +`
	`    receipt.GetStatusMessage());`
	`Console.ReadLine();`
	`}`
	`catch (Exception e)`
	`{`
	`Console.WriteLine(e);`
	`}`
	`}`
	`}`
	`}`

11 Transaction Risk Management Tool

Any of the transaction objects that are defined in this section can be passed to the HttpsPostRequest connection object defined in Section 4 (page 24).

The Transaction Risk Management Tool (TRMT) is available to **Canadian integrations** only.

11.1 Introduction to Queries

There are 3 types of transactions associated with the Transaction Risk Management Tool (TRMT):

- Session Query (page 190)
- Attribute Query (page 196)

The Session Query and Attribute Query are used at the time of the transaction to obtain the risk assessment.

Moneris recommends that you use the Session Query as much as possible for obtaining your risk assessment because it uses the device fingerprint as well as other transaction information when providing the risk scores.

To use the Session Query, you must implement two components:

- Tags on your website to collect the device fingerprinting information
- Session Query transaction.

If you are not able to collect the necessary information for the Session Query (such as the device fingerprint), then use the Attribute Query.

11.2 Session Query

Once a device profiling session has been initiated upon a client device, the Session Query API is used at the time of the transaction or even to obtain a device identifier or ‘fingerprint’, attribute list and risk assessment for the client device.

SessionQuery transaction object definition

```
SessionQuery sq = new SessionQuery();
```

HttpsPostRequest object for SessionQuery transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(sq);
```

Session Query transaction values

Table 107: SessionQuery transaction object mandatory values

Value	Type	Limits	Set method
	Description		
Session ID	String	9-character decimal Permitted characters: [a-z], [A-Z], 0-9, _, -	`sq.SetSessionId(session_id);`
	Web server session identifier generated when device profiling was initiated.		
Service type	String	TBD	`sq.SetServiceType(service_type);`
	Which output fields are returned. session -- returns IP and device related attributes.		
Event type	String	TBD	`sq.SetEventType(service_type);`
	Defines the type of transaction or event for reporting purposes. payment - Purchasing of goods/services.		
Account login	String	TBD	`sq.SetAccountLogin("13195417-8CA0-46cd-960D-14C158E4DBB2");`
	TBD		
Password hash	String	TBD	`sq.SetPasswordHash("489c830f10f7c601d30599a0deaf66e64d2aa50a");`
	TBD		
Account number	String	TBD	`sq.SetAccountNumber("3E17A905-AC8A-4c8d-A417-3DADA2A55220");`
	TBD		
Account name	String	TBD	`sq.SetAccountName("4590FCC0-DF4A-44d9-A57B-AF9DE98B84DD");`
	TBD		
Account email	String	TBD	`sq.SetAccountEmail("3CAE72EF-6B69-4a25-93FE-2674735E78E8@test.threatmetrix.com");`
	TBD		

Table 107: SessionQuery transaction object mandatory values (continued)

Value	Type	Limits	Set method / Description
Credit card number	String	20-character numeric No spaces or dashes	`sq.SetPan(pan);`
			Most credit card numbers today are 16 digits, but some 13-digit numbers are still accepted by some issuers. This field has been intentionally expanded to 20 digits in consideration for future expansion and potential support of private label card ranges.
Account address street 1	String	32-character alphanumeric	`sq.SetAccountAddressStreet1("3300 Bloor St W");`
			First portion of the street address component of the billing address.
Account Address street 2	String	32-character alphanumeric	`sq.SetAccountAddressStreet2("4th Flr West Tower");`
			Second portion of the street address component of the billing address.
Account address city	String	50-character alphanumeric	`sq.SetAccountAddressCity("Toronto");`
			The city component of the billing address.
Account address state/-province	String	64-character alphanumeric	`sq.SetAccountAddressState("Ontario");`
			The state component of the billing address.
Account address coun-try	String	2-character alphanumeric	`sq.SetAccountAddressCountry("CA");`
			ISO2 country code of the billing addresses.
Account address zip/-postal code	String	8-character alphanumeric	`sq.SetAccountAddressZip("M8X2X2");`
			Zip/postal code of the billing address.
Shipping address street 1	String	32-character alphanumeric	`sq.SetAccountAddressStreet1("3300 Bloor St W");`
			First portion of the street address component of the shipping address.

Table 107: SessionQuery transaction object mandatory values (continued)

Value	Type	Limits	Set method
	Description		
Shipping address street 2	String	32-character alphanumeric	`sq.SetAccountAddressStreet2("4th Flr West Tower");`
	Second portion of the street address component of the shipping address.		
Shipping address city	String	50-character alphanumeric	`sq.SetAccountAddressCity("Toronto");`
	City component of the shipping address.		
Shipping address state/-province	String	64-character alphanumeric	`sq.SetAccountAddressState("Ontario");`
	State component of the shipping address.		
Shipping address coun-try	String	2-character alphanumeric	`sq.SetAccountAddressCountry("CA");`
	ISO2 country code of the account address country.		
Shipping address zip	String	8-character alphanumeric	`sq.SetAccountAddressZip("M8X2X2");`
	The zip/postal code component of the shipping address.		
Local attribute 1	String	255-character alphanumeric	`sq.SetLocalAttrib1("a");`
	Can be used to pass custom attribute data. These are used if you wish to correlate some data with the returned device information.		
Local attribute 2	String	255-character alphanumeric	`sq.SetLocalAttrib2("b");`
	Can be used to pass custom attribute data. These are used if you wish to correlate some data with the returned device information.		
Local attribute 3	String	255-character alphanumeric	`sq.SetLocalAttrib3("c");`
	Can be used to pass custom attribute data. These are used if you wish to correlate some data with the returned device information.		
Local attribute 4	String	255-character alphanumeric	`sq.SetLocalAttrib4("d");`
	Can be used to pass custom attribute data. These are used if you wish to correlate some data with the returned device information.		

Table 107: SessionQuery transaction object mandatory values (continued)

Value	Type	Limits	Set method
	Description		
Local attribute 5	String	255-character alphanumeric	`sq.SetLocalAttrib5("e");`
	Can be used to pass custom attribute data. These are used if you wish to correlate some data with the returned device information.		
Transaction amount	String	255-character alphanumeric Must contain 2 decimal places	`sq.SetTransactionAmount("1.00");`
	The numeric currency amount.		
Transaction currency	String	10-character numeric	`sq.SetTransactionCurrency("840");`
	The currency type that the transaction was denominated in. If TransactionAmount is passed, the TransactionCurrency is required. Values to be used are: • CAD – 124 • USD – 840		

Sample code

Sample Session Query - CA

```
namespace Moneris
{
using System;
using System.Collections;
public class TestCanadaRiskCheckSession
{
public static void Main(string[] args)
{
string store_id = "moneris";
string api_token = "hurgle";
string order_id = "Test" + DateTime.Now.ToString("yyyyMMddhhmmss");
string session_id = "abc123";
string service_type = "session";
//string event_type = "LOGIN";
string processing_country_code = "CA";
bool status_check = false;
SessionQuery sq = new SessionQuery();
sq.SetOrderId(order_id);
sq.SetSessionId(session_id);
sq.SetServiceType(service_type);
sq.SetEventType(service_type);
```

Sample Session Query - CA

```
//sq.SetPolicy("");
//sq.SetDeviceId("4EC40DE5-0770-4fa0-BE53-981C067C598D");
sq.SetAccountLogin("13195417-8CA0-46cd-960D-14C158E4DBB2");
sq.SetPasswordHash("489c830f10f7c601d30599a0deaf66e64d2aa50a");
sq.SetAccountNumber("3E17A905-AC8A-4c8d-A417-3DADA2A55220");
sq.SetAccountName("4590FCC0-DF4A-44d9-A57B-AF9DE98B84DD");
sq.SetAccountEmail("3CAE72EF-6B69-4a25-93FE-2674735E78E8@test.threatmetrix.com");
//sq.SetAccountTelephone("5556667777");
sq.SetPan("4242424242424242");
//sq.SetAccountAddressStreet1("3300 Bloor St W");
//sq.SetAccountAddressStreet2("4th Flr West Tower");
//sq.SetAccountAddressCity("Toronto");
//sq.SetAccountAddressState("Ontario");
//sq.SetAccountAddressCountry("CA");
//sq.SetAccountAddressZip("M8X2X2");
//sq.SetShippingAddressStreet1("3300 Bloor St W");
//sq.SetShippingAddressStreet2("4th Flr West Tower");
//sq.SetShippingAddressCity("Toronto");
//sq.SetShippingAddressState("Ontario");
//sq.SetShippingAddressCountry("CA");
//sq.SetShippingAddressZip("M8X2X2");
//sq.SetLocalAttrib1("a");
//sq.SetLocalAttrib2("b");
//sq.SetLocalAttrib3("c");
//sq.SetLocalAttrib4("d");
//sq.SetLocalAttrib5("e");
//sq.SetTransactionAmount("1.00");
//sq.SetTransactionCurrency("840");
//set SessionAccountInfo
sq.SetTransactionCurrency("CAN");
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(sq);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Hashtable results = new Hashtable();
string[] rules;
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
// results = receipt.GetResult();
//Iterate through the response
// IDictionaryEnumerator r = results.GetEnumerator();
// while (r.MoveNext())
// {
// Console.WriteLine(r.Key.ToString() + " = " + r.Value.ToString());
// }
//Iterate through the rules that were fired
rules = receipt.GetRules();
for (int i = 0; i < rules.Length; i++)
{
Console.WriteLine("RuleName = " + rules[i]);
Console.WriteLine("RuleCode = " + receipt.GetRuleCode(rules[i]));
```

Sample Session Query - CA

```
Console.WriteLine("RuleMessageEn = " + receipt.GetRuleMessageEn(rules[i]));
Console.WriteLine("RuleMessageFr = " + receipt.GetRuleMessageFr(rules[i]));
}
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
} // end TestRiskCheckSession
}
```

11.2.1 Session Query Transaction Flow

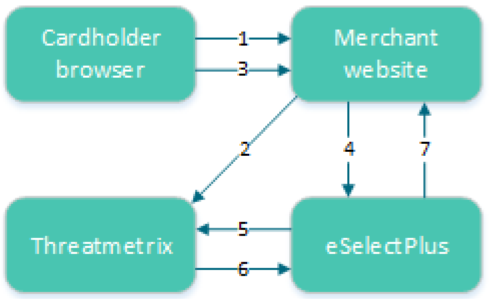


Figure 5: Session Query transaction flow

1. Cardholder logs onto the merchant website.
2. When the page has loaded in the cardholder's browser, special tags within the site allow information from the device to be gathered and sent to ThreatMetrix as the device fingerprint.

   The HTML tags should be placed where the cardholder is resident on the page for a couple of seconds to get the broadest data possible.
3. Customer submits a transaction.
4. Merchant’s web application makes a Session Query transaction to the Moneris Payment Gateway using the same session id that was included in the device fingerprint. This call must be made within 30 minutes of profiling (2).
5. Moneris Payment Gateway submits the Session Query data to ThreatMetrix.
6. ThreatMetrix uses the Session Query data and the device fingerprint information to assess the transaction against the rules. A score is generated based on the rules.
7. The merchant uses the returned device information in its risk analysis to make a business decision. The merchant may wish to continue or cancel with the cardholder’s payment transaction.

11.3 Attribute Query

The Attribute Query is used to obtain a risk assessment of transaction-related identifiers such as the email address and the card number. Unlike the Session Query, the Attribute Query does not require the device fingerprinting information to be provided.

AttributeQuery transaction object definition

```
AttributeQuery aq = new AttributeQuery();
```

HttpsPostRequest object for AttributeQuery transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

Attribute Query transaction values

Table 108: Attribute Query transaction object mandatory values

Value	Type	Limits	Set method
	Description		
Service type	String	TBD	`aq.setServiceType(service_type);`
	Which output fields are returned. session -- returns IP and device related attributes.		
Device ID	String	36-character alphanumeric	`aq.setDeviceId("");`
	Unique device identifier generated by a previous call to the ThreatMetrix session-query API.		
Credit card number	String	20-character numeric No spaces or dashes	`aq.SetPan(pan);`
	Most credit card numbers today are 16 digits, but some 13-digit numbers are still accepted by some issuers. This field has been intentionally expanded to 20 digits in consideration for future expansion and potential support of private label card ranges.		
IP address	String	64-character alphanumeric	`aq.setIPAddress("192.168.0.1");`
	True IP address. Results will be returned as true_ip_geo, true_ip_score and so on.		
IP forwarded	String	64-character alphanumeric	`aq.setIPForwarded ("192.168.1.0");`
	The IP address of the proxy. If the IPAddress is supplied, results will be returned as proxy_ip_geo and proxy_ip_score. If the IP Address is not supplied, this IP address will be treated as the true IP address and results will be returned as true_ip_geo, true_ip_score and so on		
Account address street 1	String	32-character alphanumeric	`aq.setAccountAddressStreet1 ("3300 Bloor St W");`
	First portion of the street address component of the billing address.		

Table 108: Attribute Query transaction object mandatory values (continued)

Value	Type	Limits	Set method
	Description		
Account Address Street 2	String	32-character alphanumeric	`aq.setAccountAddressStreet2("4th Flr West Tower");`
	Second portion of the street address component of the billing address.		
Account address city	String	50-character alphanumeric	`aq.setAccountAddressCity ("Toronto");`
	The city component of the billing address.		
Account address state/-province	String	64-character alphanumeric	`aq.setAccountAddressState ("Ontario");`
	The state component of the billing address.		
Account address coun-try	String	2-character alphanumeric	`aq.setAccountAddressCountry ("CA");`
	ISO2 country code of the billing addresses.		
Account address zip/-postal code	String	8-character alphanumeric	`aq.setAccountAddressZip ("M8X2X2");`
	Zip/postal code of the billing address.		
Shipping address street 1	String	32-character alphanumeric	`aq.setShippingAddressStreet1 ("3300 Bloor St W");`
	Account address country		
Shipping Address Street 2	String	32-character alphanumeric	`aq.setShippingAddressStreet2 ("4th Flr West Tower");`
	Second portion of the street address component of the shipping address.		
Shipping Address City	String	50-character alphanumeric	`aq.setShippingAddressCity ("Toronto");`
	City component of the shipping address.		
Shipping Address State/Province	String	64-character alphanumeric	`aq.setShippingAddressState ("Ontario");`
	State/Province component of the shipping address.		
Shipping Address Coun-try	String	2-character alphanumeric	`aq.setShippingAddressCountry ("CA");`
	ISO2 country code of the account address country.		
Shipping Address zip/-postal code	String	8-character alphanumeric	
	The zip/postal code component of the shipping address.		

Sample Attribute Query - CA

```
namespace Moneris
{
using System;
using System.Collections;
public class TestRiskCheckAttribute
{
public static void Main(string[] args)
{
string store_id = "moneris";
string api_token = "hurgle";
string order_id = "Test" + DateTime.Now.ToString("yyyyMMddhhmmss");
string service_type = "session";
string processing_country_code = "CA";
bool status_check = false;
AttributeQuery aq = new AttributeQuery();
aq.SetOrderId(order_id);
aq.SetServiceType(service_type);
aq.setDeviceId("");
aq.setAccountLogin("13195417-8CA0-46cd-960D-14C158E4DBB2");
aq.setPasswordHash("489c830f10f7c601d30599a0deaf66e64d2aa50a");
aq.setAccountNumber("3E17A905-AC8A-4c8d-A417-3DADA2A55220");
aq.setAccountName("4590FCC0-DF4A-44d9-A57B-AF9DE98B84DD");
aq.setAccountEmail("3CAE72EF-6B69-4a25-93FE-2674735E78E8@test.threatmetrix.com");
//aq.setCCNumberHash("4242424242424242");
//aq.setIPAddress("192.168.0.1");
//aq.setIPForwarded("192.168.1.0");
aq.setAccountAddressStreet1("3300 Bloor St W");
aq.setAccountAddressStreet2("4th Flr West Tower");
aq.setAccountAddressCity("Toronto");
aq.setAccountAddressState("Ontario");
aq.setAccountAddressCountry("CA");
aq.setAccountAddressZip("M8X2X2");
aq.setShippingAddressStreet1("3300 Bloor St W");
aq.setShippingAddressStreet2("4th Flr West Tower");
aq.setShippingAddressCity("Toronto");
aq.setShippingAddressState("Ontario");
aq.setShippingAddressCountry("CA");
aq.setShippingAddressZip("M8X2X2");
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(aq);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Hashtable results = new Hashtable();
string[] rules;
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
results = receipt.GetResult();
//Iterate through the response
IDictionaryEnumerator response = results.GetEnumerator();
while (response.MoveNext())
{
```

Sample Attribute Query - CA

```
Console.WriteLine(response.Key.ToString() + " = " + response.Value.ToString());
}
//Iterate through the rules that were fired
rules = receipt.GetRules();
for (int i = 0; i < rules.Length; i++)
{
Console.WriteLine("RuleName = " + rules[i]);
Console.WriteLine("RuleCode = " + receipt.GetRuleCode(rules[i]));
Console.WriteLine("RuleMessageEn = " + receipt.GetRuleMessageEn(rules[i]));
Console.WriteLine("RuleMessageFr = " + receipt.GetRuleMessageFr(rules[i]));
}
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
} // end TestRiskCheckAttribute
}
```

11.3.1 Attribute Query Transaction Flow

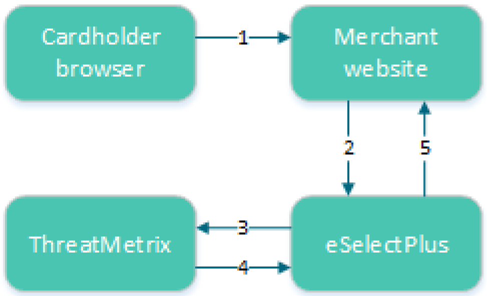


Figure 6: Attribute query transaction flow

1. Cardholder logs onto merchant website and submits a transaction.
2. The merchant's web application makes an Attribute Query transaction that includes the session ID to the Moneris Payment Gateway.
3. Moneris Payment Gateway submits Attribute Query data to ThreatMetrix.
4. ThreatMetrix uses the Attribute Query data to assess the transaction against the rules. A score is generated based on the rules.
5. The merchant uses the returned device information in its risk analysis to make a business decision. The merchant may wish to continue or cancel with the cardholder's payment trans-action.

11.4 Handling Response Information

When reviewing the response information and determining how to handle the transaction, it is recom-mended that you (either manually or through automated logic on your site) use the following pieces of information:

- Risk score
- Rules triggered (such as Rule Codes, Rule Names, Rule Messages)
- Results obtained from Verified by Visa, MasterCard Secure Code, AVS, CVD and the financial transaction authorization
- Response codes for the Transaction Risk Management Transaction that are included by automated processes.

11.4.1 TRMT Response Fields

Table 109: Receipt object response values for TRMT

Value	Type	Limits	Get method
	Definition		
Response Code	String	3-character alphanumeric	
	See Table 110 (page 202)		
Message	String	TBD	
	Response message		
Event type	String	TBD	
	Type of transaction or event returned in the response.		
Org ID	String	TBD	
	ThreatMetrix-defined unique transaction identifier		
Policy	String	TBD	
	Policy used for the Session Query will be returned with the return request. If the Policy was not included, then the Policy name default is returned.		
Policy score	String	TBD	
	The sum of all the risks weights from triggered rules within the selected policy in the range [-100...100]		
Request duration	String	TBD	
	Length of time it takes for the transaction to be processed.		
Request ID	String	TBD	
	Unique number and will always be returned with the return request.		
Request result	String	TBD	
	See Table 111 (page 203).		

Table 109: Receipt object response values for TRMT (continued)

Value	Type	Limits	Get method
	Definition		
Review status	String	TBD	
	The transaction status based on the assessments and risk scores.		
Risk rating	String	TBD	
	The rating based on the assessments and risk scores.		
Service type	String	TBD	
	The service type will be returned in the attribute query response.		
Session ID	String	TBD	
	Temporary identifier unique to the visitor will be returned in the return request.		
Summary risk score	String	TBD	
	Based on all of the returned values in the range [-100 ... 100]		
Transaction ID	String	TBD	
	This is the transaction identifier and will always be returned in the response when supplied as input.		
Unknown session	String	TBD	
	If present, the value is "yes". It indicates the session ID that was passed was not found.		
ITD Enhanced AVS Response Code	String	1-character alphabetic	

	The ITD (Internet Transaction Data) reviews several methods for performing a credit card transaction online. The ITDReponse indicates the AmEx ITD validation results. Applicable for AmEx and JCB only.
	
	Y = data matches
	N = data does not match
	U = data not checked
	R = retry
	S = Service not allowed [space] = data not sent		

Table 110: Response code descriptions

Value	Definition
001	Success
981	Data error

Value	Definition
982	Duplicate order ID
983	Invalid transaction
984	Previously asserted
985	Invalid activity description
986	Invalid impact description
987	Invalid confidence description
988	Cannot find previous

Table 111: Request result values and descriptions

Value	Definition
fail_incomplete	ThreatMetrix was unable to process the request due to incomplete or incorrect input data
fail_invalid_telephone_ number	Format of the supplied telephone number was invalid
fail_access	ThreatMetrix was unable to process the request because of API verification failing
fail_internal_error	ThreatMetrix encountered an error while processing the request
fail_invalid_device_id	Format of the supplied device_id was invalid
fail_invalid_email_address	Format of the supplied email address was invalid
fail_invalid_ip_address_ parameter	Format of a supplied ip_address parameter was invalid
fail_temporarily_unavailable	Request failed because the service is temporarily unavailable
fail_verification	API query limit reached
success	ThreatMetrix was able to process the request successfully

11.4.2 Understanding the Risk Score

For each Session Query or Attribute Query, a score with a value between -100 and +100 is returned based on the rules that were triggered for the transaction.

Table 112 defines the risk scores ranges.

Table 112: Session Query and Attribute Query risk score definitions

Risk score	Visa definition
-100 to -1	A lower score indicates a higher probability that the transaction is fraudulent.
0	Neutral transaction
1 to 100	A higher score indicates a lower probability that the transaction is fraudulent. **Note**: All e-commerce transactions have some level of risk associated with them. Therefore, it is rare to see risk score in the high positive values.

When evaluating the risk of a transaction, the risk score gives an initial indicator of the potential risk that the transaction is fraudulent. Because some of the rules that are evaluated on each transaction may not be relevant to your business scenario, review the rules that were triggered for the transaction before determining how to handle the transaction.

11.4.3 Understanding the Rule Codes, Rule Names and Rule Messages

The rule codes, rule names and rule messages provide details about what rules were triggered during the assessment of the information provided in the Session or Attribute Query. Each rule code has a rule name and rule message. The rule name and rule message are typically similar. Table 113 provides additional information on each rule.

When evaluating the risk of a transaction, it is recommended that you review the rules that were triggered for the transaction and assess the relevance to your business. (That is, how does it relate to the typical buying habits of your customer base?)

If you are automating some or all of the decision-making processes related to handling the responses, you may want to use the rule codes. If you are documenting manual processes, you may want to refer to the more user-friendly rule name or rule message.

Table 113: Rule names, numbers and messages

Rule name	Rule number	Rule message
	Rule explanation	
White lists		
DeviceWhitelisted	WL001	Device White Listed
	Device is on the white list. This indicates that the device has been flagged as always "ok". **Note**: This rule is currently not in use.	
IPWhitelisted	WL002	IP White Listed
	IP address is on the white list. This indicates the device has been flagged as always "ok". **Note**: This rule is currently not in use.	

Table 113: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
EmailWhitelisted	WL003	Email White Listed
	Email address is on the white list. This indicates that the device has been flagged as always "ok". **Note**: This rule is currently not in use.	
Event velocity		
2DevicePayment	EV003	2 Device Payment Velocity
	Multiple payments were detected from this device in the past 24 hours.	
2IPPaymentVelocity	EV006	2 IP Payment Velocity
	Multiple payments were detected from this IP within the past 24 hours.	
2ProxyPaymentVelocity	EV008	2 Proxy Payment Velocity
	The device has used 3 or more different proxies during a 24 hour period. This could be a risk or it could be someone using a legitimate corporate proxy.	
Email		
3EmailPerDeviceDay	EM001	3 Emails for the Device ID in 1 Day
	This device has presented 3 different email IDs within the past 24 hours.	
3EmailPerDeviceWeek	EM002	3 emails for the Device ID in 1 week
	This device has presented 3 different email IDs within the past week.	
3DevciePerEmailDay	EM003	3 Device Ids for email address in 1 day
	This email has been presented from three different devices in the past 24 hours.	
3DevciePerEmailWeek	EM004	3 Device Ids for email address in 1 week
	This email has been presented from three different devices in the past week.	
EmailDistanceTravelled	EM005	Email Distance Travelled
	This email address has been associated with different physical locations in a short period of time.	

Table 113: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
3EmailPerSmartIDHour	EM006	3 Emails for SmartID in 1 Hour
	The SmartID for this device has been associated with 3 different email addresses in 1 hour.	
GlobalEMailOverOneMonth	EM007	Global Email over 1 month
	The e-mail address involved in the transaction over 30 days ago. This generally indicates that the transaction is less risky. **Note**: This rule is set so that it does not impact the policy score or risk rating.	
ComputerGeneratedEmailAddress	EM008	Computer Generated Email Address
	This transaction used a computer-generated email address.	
Account Number		
3AccountNumberPerDeviceDay	AN001	3 Account Numbers for device in 1 day
	This device has presented 3 different user accounts within the past 24 hours.	
3AccountNumberPerDeviceWeek	AN002	3 Account Numbers for device in 1 week
	This device has presented 3 different user accounts within the past week.	
3DevciePerAccountNumberDay	AN003	3 Device IDs for account number in 1 day
	This user account been used from three different devices in the past 24 hours.	
3DevciePerAccountNumberWeek	AN004	3 Device IDs for account number in 1 week
	This card number has been used from three different devices in the past week.	
AccountNumberDistanceTravelled	AN005	Account Number distance travelled
	This card number has been used from a number of physically different locations in a short period of time.	
Credit card/payments		
3CreditCardPerDeviceDay	CP001	3 credit cards for device in 1 day
	This device has used three credit cards within 24 hours.	

Table 113: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
3CreditCardPerDeviceWeek	CP002	3 credit cards for device in 1 week
	This device has used three credit cards within 1 week.	
3DevicePerCreditCardDay	CP003	3 device ids for credit card in 1 day
	This credit card has been used on three different devices in 24 hours.	
3DevciePerCreditCardWeek	CP004	3 device ids for credit card in 1 week
	This credit card has been used on three different devices in 1 week.	
CredtCardDistanceTravelled	CP005	Credit Card has travelled
	The credit card has been used at a number of physically different locations in a short period of time.	
CreditCardShipAddressGeoMismatch	CP006	Credit Card and Ship Address do not match
	The credit card was issued in a region different from the Ship To Address information provided.	
CreditCardBillAddressGeoMismatch	CP007	Credit Card and Billing Address do not match
	The credit card was issued in a region different from the Billing Address information provided.	
CreditCardDeviceGeoMismatch	CP008	Credit Card and device location do not match
	The device is located in a region different from where the card was issued.	
CreditCardBINShipAddressGeoMismatch	CP009	Credit Card issuing location and Shipping address do not match
	The credit card was issued in a region different from the Ship To Address information provided.	
CreditCardBINBillAddressGeoMismatch	CP010	Credit Card issuing location and Billing address do not match
	The credit card was issued in a region different from the Billing Address information provided.	

Table 113: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
CreditCardBINDeviceGeoMismatch	CP011	Credit Card issuing location and location of the device do not match
	The device is located in a region different from where the card was issued.	
TransactionValueDay	CP012	Daily Transaction Value Threshold
	The transaction value exceeds the daily threshold.	
TransactionValueWeek	CP013	Weekly Transaction Value Threshold
	The transaction value exceeds the weekly threshold.	
Proxy rules		
3ProxyPerDeviceDay	PX001	3 Proxy Ips in 1 day
	This device has used three different proxy servers in the past 24 hours.	
AnonymousProxy	PX002	Anonymous Proxy IP
	This device is using an anonymous proxy	
UnusualProxyAttributes	PX003	Unusual Proxy Attributes
	This transaction is coming from a source with unusual proxy attributes.	
AnonymousProxy	PX004	Anonymous Proxy
	This device is connecting through an anonymous proxy connection.	
HiddenProxy	PX005	Hidden Proxy
	This device is connecting via a hidden proxy server.	
OpenProxy	PX006	Open Proxy
	This transaction is coming from a source that is using an open proxy.	
TransparentProxy	PX007	Transparent Proxy
	This transaction is coming from a source that is using a transparent proxy.	
DeviceProxyGeoMismatch	PX008	Proxy and True GEO Match
	This device is connecting through a proxy server that didn’t match the devices geo-location.	

Table 113: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
ProxyTrueISPMismatch	PX009	Proxy and True ISP Match
	This device is connecting through a proxy server that doesn’t match the true IP address of the device.	
ProxyTrueOrganizationMismatch	PX010	Proxy and True Org Match
	The Proxy information and True ISP information for this source do not match.	
DeviceProxyRegionMismatch	PX011	Proxy and True Region Match
	The proxy and device region location information do not match.	
ProxyNegativeReputation	PX012	Proxy IP Flagged Risky in Reputation Network
	This device is connecting from a proxy server with a known negative reputation.	
SatelliteProxyISP	PX013	Satellite Proxy
	This transaction is coming from a source that is using a satellite proxy.	
GEO		
DeviceCountriesNotAllowed	GE001	True GEO in Countries Not Allowed blacklist
	This device is connecting from a high-risk geographic location.	
DeviceCountriesNotAllowed	GE002	True GEO in Countries Not Allowed (negative whitelist)
	The device is from a region that is not on the whitelist of regions that are accepted.	
DeviceProxyGeoMismatch	GE003	True GEO different from Proxy GEO
	The true geographical location of this device is different from the proxy geographical location.	
DeviceAccountGeoMismatch	GE004	Account Address different from True GEO
	This device has presented an account billing address that doesn't match the devices geolocation.	
DeviceShipGeoMismatch	GE005	Device and Ship Geo mismatch
	The location of the device and the shipping address do not match.	

Table 113: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
DeviceShipGeoMismatch	GE006	Device and Ship Geo mismatch
	The location of the device and the shipping address do not match.	
Device		
SatelliteISP	DV001	Satellite ISP
	This transaction is from a source that is using a satellite ISP.	
MidsessionChange	DV002	Session Changed Mid-session
	This device changed session details and identifiers in the middle of a session.	
LanguageMismatch	DV003	Language Mismatch
	The language of the user does not match the primary language spoken in the location where the True IP is registered.	
NoDeviceID	DV004	No Device ID
	No device ID was available for this transaction.	
Dial-upConnection	DV005	Dial-up connection
	This device uses a less identifiable dial-up connection.	
DeviceNegativeReputation	DV006	Device Blacklisted in Reputational Network
	This device has a known negative reputation as reported to the fraud network.	
DeviceGlobalBlacklist	DV007	Device on the Global Black List
	This device has been flagged on the global blacklist of known problem devices.	
DeviceCompromisedDay	DV008	Device compromised in last day
	This device has been reported as compromised in the last 24 hours.	
DeviceCompromisedHour	DV009	Device compromised in last hour
	This device has been reported as compromised in the last hour.	
FlashImagesCookiesDisabled	DV010	Flash Images Cookies Disabled
	Key browser functions/identifiers have been disabled on this device.	

Table 113: Rule names, numbers and messages (continued)

Rule name	Rule number	Rule message
	Rule explanation	
FlashCookiesDisabled	DV011	Flash Cookies Disabled
	Key browser functions/identifiers have been disabled on this device.	
FlashDisabled	DV012	Flash Disabled
	Key browser functions/identifiers have been disabled on this device.	
ImagesDisabled	DV013	Images Disabled
	Key browser functions/identifiers have been disabled on this device.	
CookiesDisabled	DV014	Cookies Disabled
	Key browser functions/identifiers have been disabled on this device.	
DeviceDistanceTravelled	DV015	Device Distance Travelled
	The device has been used from multiple physical locations in a short period of time.	
PossibleCookieWiping	DV016	Cookie Wiping
	This device appears to be deleting cookies after each session.	
PossibleCookieCopying	DV017	Possible Cookie Copying
	This device appears to be copying cookies.	
PossibleVPNConnection	DV018	Possibly using a VPN Connection
	This device may be using a VPN connection	

11.4.4 Examples of Risk Response

11.4.4.1 Session Query

Sample Risk Response - Session Query

```
<?xml version="1.0"?>
<response>
<receipt>
    <ResponseCode>001</ResponseCode>
    <Message>Success</Message>
<Result>
```

Sample Risk Response - Session Query

```
    <session_id>abc123</session_id>
    <unknown_session>yes</unknown_session>
    <event_type>payment</event_type>
    <service_type>session</service_type>
    <policy_score>-25</policy_score>
    <transaction_id>riskcheck42</transaction_id>
    <org_id>11kue096</org_id>
    <request_id>91C1879B-33D4-4D72-8FCB-B60A172B3CAC</request_id>
    <risk_rating>medium</risk_rating>
    <request_result>success</request_result>
    <summary_risk_score>-25</summary_risk_score>
    <Policy>default</policy>
    <review_status>review</review_status>
</Result>
<Rule>
    <RuleName>ComputerGeneratedEMail</RuleName>
    <RuleCode>UN001</RuleCode>
    <RuleMessageEn>Unknown Rule</RuleMessageEn>
    <RuleMessageFr>Regle Inconnus</RuleMessageFr>
</Rule>
<Rule>
    <RuleName>NoDeviceID</RuleName>
    <RuleCode>DV004</RuleCode>
    <RuleMessageEn>No Device ID</RuleMessageEn>
    <RuleMessageFr>null</RuleMessageFr>
</Rule>
</receipt>
</response>
```

11.4.4.2 Attribute Query

Sample Risk Response - Attribute Query

```
<?xml version="1.0"?>
<response>
<receipt>
    <ResponseCode001</ReponseCode>
    <Message = Success</Message>
<Result>
    <org_id>11kue096</org_id>
    <request_id>443D7FB5-CC5C-4917-A57E-27EAC824069C</request_id>
    <service_type>session</service_type>
    <risk_rating>medium</risk_rating>
    <summary_risk_score>-25</summary_risk_score>
    <request_result>success</request_result>
    <policy>default</policy>
    <policy_score>-25</policy_score>
    <transaction_id>riskcheck19</transaction_id>
    <review_status>review</review_status>
</Result>
<Rule>
    <RuleName>ComputerGeneratedEMail</RuleName>
    <RuleCode>UN001</RuleCode>
    <RuleMessageEn>Unknown Rule</RuleMessageEn>
    <RuleMessageFr>Regle Inconnus</RuleMessageFr>
</Rule>
<Rule>
```

Sample Risk Response - Attribute Query

```
    <RuleName>NoDeviceID</RuleName>
    <RuleCode>DV004</RuleCode>
    <RuleMessageEn>No Device ID</RuleMessageEn>
    <RuleMessageFr>null</RuleMessageFr>
</Rule>
</receipt>
</response>
```

11.4.4.3 Assertion Query

Sample Risk Response - Assertion Query

```
<?xml version="1.0"?>
<response>
<receipt>
    <ResponseCode>001</ResponseCode>
    <Message>Successful Assertion</Message>
<Result>
    <request_id>967F1AB1-4F19-4A13-9945-B5B19D784305</request_id>
    <request_result>success<request_result>
    <request_duration>51</request_duration>
</Result>
</receipt>
</response>
```

11.5 Inserting the Profiling Tags Into Your Website

Place the profiling tags on an HTML page served by your web application such that ThreatMetrix can collect device information from the customer's web browser. The tags must be placed on a page that a visitor would display in a browser window for 3-5 seconds (such as a page that requires a user to input data). After the device is profiled, a Session Query may be used to obtain the detail device information for risk assessment before submitting a financial payment transaction.

There are two profiling tags that require two variables. Those tags are `org_id` and `session_id`. `session_id` must match the session ID value that is to be passed in the Session Query transaction. The valid `org_id` values are:

11kue096
QA testing environment.

lbhqgx47
Production environment.

Below is an HTML sample of the profiling tags.

Note	Your site must replace `<my_session_id>` in the sample code with a unique alphanumeric value each time you fingerprint a new customer.

```
<p style="background:url(https://h.online-metrix.net/fp/clear.png?org_id=11kue096&session_id=<my_
    session_id>&m=1)">
</p>

<img src="https://h.onlinemetrix.net/fp/clear.png?org_id=11kue096&session_id=<my_session_id>&m=2" alt=""
    >

<script src="https://h.onlinemetrix.net/fp/check.js?org_id=11kue096&session_id=<my_session_id>"
type="text/javascript">
</script>

<object type="application/x-shockwave-flash"

data="https://h.onlinemetrix.net/fp/fp.swf?org_id=11kue096&session_id=<my_session_id>"
width="1" height="1" id="obj_id">
<param name="movie"
value="https://h.onlinemetrix.net/fp/fp.swf?org_id=11kue096&session_id=<my_session_id>" />
<div></div>
</object>
```

12 Convenience Fee

12.1 About Convenience Fee

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

12.2 Purchase - Convenience Fee

Note	Convenience Fee Purchase with Customer Information is also supported.

Convenience Fee Purchase transaction object definition

HttpsPostRequest object for Convenience Fee Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(TRANSACTION);
```

Convenience Fee Purchase transaction object values

For a full description of mandatory and optional values, see "Definition of Request Fields" on page 258

Table 1: Convenience Fee Purchase transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alphanumeric	
Amount	String	9-character decimal	
Credit card number	String	20-character numeric	
Expiry date	String	4-character numeric YYMM format	

Table 1: Convenience Fee Purchase transaction object mandatory values (continued)

Value	Type	Limits	Set Method
E-commerce indicator	String	1-character alphanumeric	
Convenience fee amount	String	9-character decimal	

Table 2: Convenience Fee Purchase transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alphanumeric	
Dynamic descriptor	String	20-character alphanumeric	
Commercial card invoice	String	17-character alphanumeric	
Commercial card tax amount	String	9-character decimal	
Customer information	Object		
AVS information	Object		
CVD information	Object		
Convenience Fee	Object		

Sample Convenience Fee Purchase - CA	Sample Convenience Fee Purchase - US
`SAMPLE CODE TO COME`	

12.3 ACH Debit - Convenience Fee

Note	Convenience Fee ACH Debit with Customer Information is also supported.

Convenience Fee ACH Debit transaction object definition

HttpsPostRequest object for Convenience Fee ACH Debit transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.SetTransaction(TRANSACTION);
```

Convenience Fee ACH Debit transaction object values

Sample Convenience Fee ACH Debit - US

12.4 Purchase with VbV and Mastercard Secure Code

Convenience Fee Purchase with VbV and MCSC transaction object definition

HttpsPostRequest object for Convenience Fee Purchase with VbV and MCSC transaction

Convenience Fee Purchase with VbV and MCSC transaction object values

Sample Purchase with VbV and MC Secure Code - CA	Sample Purchase with VbV and MC Secure Code - US

13 Visa Checkout

Delete this text and replace it with your own content.

13.1 Transaction Types - Visa Checkout

Below is a list of transactions supported by the Visa Checkout API, other terms used for the transaction type are indicated in brackets.

VdotMePurchase (sale)
Call to Moneris to obtain funds on the Visa Checkout `callId` and ready them for deposit into the merchant's account. It also updates the customer's Visa Checkout transaction history.

VdotMePreAuth (authorisation / pre-authorization)
Call to Moneris to verify funds on the Visa Checkout `callid` and reserve those funds for your merchant account. The funds are locked for a specified amount of time, based on the card issuer. To retrieve the funds from this call so that they may be settled in the merchant's account, a `VdotMeCompletion` must be performed. It also updates the customer's Visa Checkout transaction history.

VdotMeCompletion (Completion / Capture)
Call to Moneris to obtain funds reserved by `VdotMePreAuth` call. This transaction call retrieves the locked funds and readies them for settlement into the merchant's account. This call must be made typically within 72 hours of performing `VdotMePreAuth`. It also updates the customer's Visa Checkout transaction history.

VdotMePurchaseCorrection (Void / Purchase Correction)
Call to Moneris to void the VdotMePurchases and VdotMeCompletions the same day* that they occurred on. It also updates the customer's Visa Checkout transaction history.

VdotMeRefund (Credit)
Call to Moneris to refund against a `VdotMePurchase` or `VdotMeCompletion` to refund any part, or all of the transaction. It also updates the customer's Visa Checkout transaction history.

VdotMeInfo (Credit)
Call to Moneris to refund against a `VdotMePurchase` or `VdotMeCompletion` to refund any part, or all of the transaction. It also updates the customer's Visa Checkout transaction history.

13.2 Transaction Flow - Visa Checkout

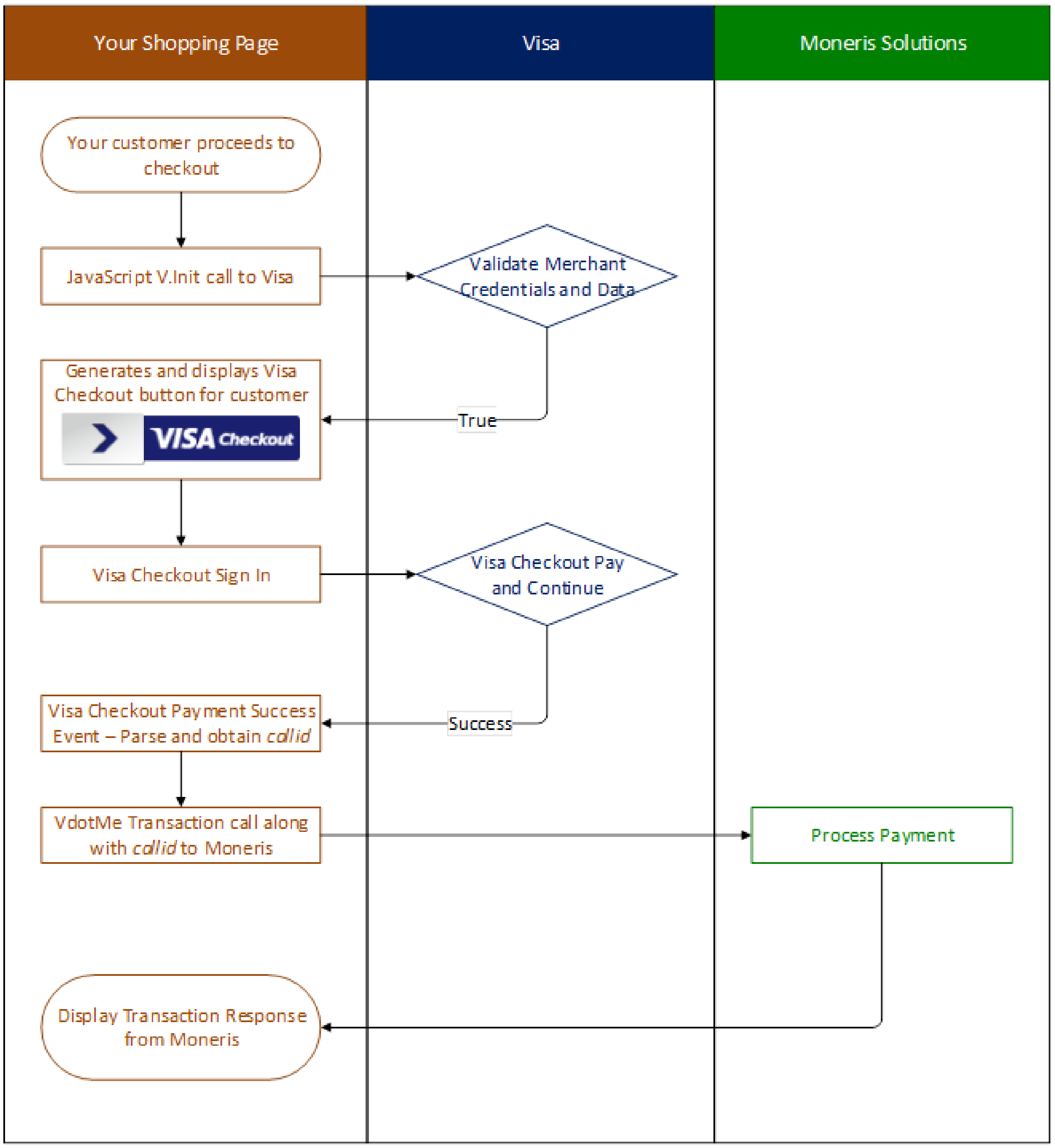
1. Create Visa Checkout Lightbox integration by following the Visa documentation, which is available on Visa Developer portal:
   Simple Visa Checkout button with no custom data:
   https://developer.visa.com/vme/merchant/documents/Getting_Started_With_Visa_Checkout/Quick_Start_Tutorial.html#Adding_a_Visa_Checkout_Button_to_a_Web_Page
   Advanced Visa Checkout button with custom data:

https://developer.visa.com/vme/merchant/documents/Visa_Checkout_JavaScript_Integration_Guide/JavaScript_and_Button_Reference.html

2. If you get a payment success event from the above Visa Lightbox script, you will have to parse and obtain the `callid` from their JSON response. You can obtain other additional details about cardholder by decrypting and parsing the Visa Lightbox's JSON response.
3. 3. Once you have obtained the callid from Visa Lightbox, you can make appropriate `VdotMe` transaction call to Moneris to process your transaction and obtain your funds.

NOTE	During Visa Checkout testing in our QA test environment, please use `apikey` for the `V.Init` call in your JavaScript.

VISA Checkout Process – Successful Process


Your Shopping Page
Visa
Moneris Solutions
Your customer proceeds to checkout
JavaScript V.Init call to Visa
Validate Merchant Credentials and Data
True
Generates and displays Visa Checkout button for customer
VISA Checkout
Visa Checkout Sign In
Visa Checkout Pay and Continue
Success
Visa Checkout Payment Success Event – Parse and obtain *callid*
VdotMe Transaction call along with *callid* to Moneris
Process Payment
Display Transaction Response from Moneris

13.3 Visa Checkout Purchase

VdotMePurchase transaction object definition

```
VdotMePurchase vmepurchase = new VdotMePurchase();
```

HttpsPostRequest for VdotMePurchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(vmepurchase);
```

VdotMePurchase transaction object values

Table 1: VdotMePurchase transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`vmepurchase.SetOrderId(order_id);`
Call ID	String	20-character numeric	`vmepurchase.SetCallId(call_id);`
Amount	String	9-character decimal	`vmepurchase.SetAmount(amount);`
E-commerce indicator	String	1-character alphanumeric	`vmepurchase.SetCryptType(crypt);`

Table 2: VdotMePurchase transaction object optional values

Value	Type	Limits	Set Method
Dynamic descriptor	String	20-character alphanumeric	`vDotMePurchaseCorrection.SetDynamicDescriptor(dynamic_descriptor);`
Status check	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Sample VdotMePurchase - CA

```
using System;
using System.Collections.Generic;
using System.Text;
using Moneris;
namespace Moneris
{
class TestCanadaVdotMePurchase
{
public static void Main(string[] args)
{
string store_id = "store2";
string api_token = "yesguy";
string cust_id = "Joe Doe";
string order_id = "VmeOrder" + DateTime.Now.ToString("yyyyMMddhhmmss");
string amount = "8.00";
string crypt_type = "7";
string call_id = "2374837188642083454";
string dynamic_descriptor = "inv123";
string processing_country_code = "CA";
bool status_check = false;
VdotMePurchase vmepurchase = new VdotMePurchase();
vmepurchase.SetOrderId(order_id);
vmepurchase.SetCustId(cust_id);
```

Sample VdotMePurchase - CA

```
vmepurchase.SetAmount(amount);
vmepurchase.SetCallId(call_id);
vmepurchase.SetCryptType(crypt_type);
vmepurchase.SetDynamicDescriptor(dynamic_descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(vmepurchase);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("StatusCode = " + receipt.GetStatusCode());
Console.WriteLine("StatusMessage = " + receipt.GetStatusMessage());
Console.WriteLine("\r\nPress the enter key to exit");
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

13.4 Visa Checkout PreAuth

`VdotMePreAuth` is virtually identical to the `VdotMePurchase` with the exception of the transaction type name.

If the order could not be completed for some reason, such as an order is cancelled, made in error or not fulfillable, the `VdotMePreAuth` transaction must be reversed within 72 hours.

To reverse an authorization, perform a `VdotMeCompletion` transaction for $0.00 (zero dollars).

VdotMePreAuth transaction object definition

```
VdotMePreauth vMePreauthRequest = new VdotMePreauth();
```

HttpsPostRequest object for VdotMePreAuth transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(vMePreauthRequest);
```

VdotMePreAuth transaction object values

Table 1: VdotMePreAuth transaction object mandatory values

Value	Type	Limits	Set Method
Amount	String	9-character decimal	`vDotMeReauthRequest.SetAmount (amount);`
Call ID	String	20-character numeric	`vDotMeReauthRequest.SetCallId(call_ id);`
Order ID	String	50-character alpha-numeric	`vDotMeReauthRequest.SetOrderId (order_id);`
E-commerce indicator	String	1-character alpha-numeric	`vDotMeReauthRequest.SetCryptType (crypt);`

Table 2: VdotMePreAuth transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`vMePreauthRequest.SetCustId(cust_id);`
Dynamic descriptor	String	20-character alpha-numeric	`vDotMeReauthRequest.SetDynamicDescriptor (dynamic_descriptor);`

Sample VdotMePreAuth - CA

```
using System;
namespace Moneris
{
class TestCanadaVdotMePreauth
{
public static void Main(string[] args)
{
string store_id = "store2";
string api_token = "yesguy";
string amount = "5.00";
string crypt_type = "7";
string order_id = "VmeOrder" + DateTime.Now.ToString("yyyyMMddhhmmss");
string call_id = "2336392495138357172";
string cust_id = "my customer id";
string processing_country_code = "CA";
bool status_check = false;
VdotMePreauth vMePreauthRequest = new VdotMePreauth();
vMePreauthRequest.SetOrderId(order_id);
vMePreauthRequest.SetAmount(amount);
vMePreauthRequest.SetCallId(call_id);
```

Sample VdotMePreAuth - CA

```
vMePreauthRequest.SetCustId(cust_id);
vMePreauthRequest.SetCryptType(crypt_type);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(vMePreauthRequest);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("StatusCode = " + receipt.GetStatusCode());
Console.WriteLine("StatusMessage = " + receipt.GetStatusMessage());
Console.WriteLine("\r\nPress the enter key to exit");
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

13.5 Visa Checkout Completion

The `VdotMeCompletion` transaction is used to secure the funds locked by a `VdotMePreAuth` transaction.

You may also perform this transaction at $0.00 (zero dollars) to reverse a `VdotMePreauth` transaction that you are unable to fulfill.

VdotMeCompletion transaction object definition

```
VdotMeCompletion vmecompletion = new VdotMeCompletion();
```

HttpsPostRequest object for VdotMeCompletion transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.SetTransaction(vmecompletion);
```

VdotMeCompletion transaction object values

Table 1: VdotMeCompletion transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`vmecompletion.SetOrderId(order_id);`
Transaction number	String	255-character alpha-numeric	`vmecompletion.SetTxnNumber(txn_number);`
Completion amount	String	9-character decimal	`vmecompletion.SetCompAmount(amount);`
E-commerce indicator	String	1-character alpha-numeric	`vmecompletion.SetCryptType(crypt);`

Table 2: VdotMeCompletion transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`vmecompletion.SetCustId(cust_id);`
Dynamic descriptor	String	20-character alpha-numeric	`vmecompletion.SetDynamicDescriptor(dynamic_descriptor);`

Sample VdotMeCompletion - CA

```
using System;
namespace Moneris
{
class TestCanadaVdotMeCompletion
{
public static void Main(string[] args)
{
string store_id = "store2";
string api_token = "yesguy";
string order_id = "VmeOrder20150626023358";
string txn_number = "737541-0_10";
string comp_amount = "1.00";
string ship_indicator = "P";
string crypt_type = "7";
string cust_id = "mycustomerid";
string dynamic_descriptor = "inv 123";
string processing_country_code = "CA";
bool status_check = false;
VdotMeCompletion vmecompletion = new VdotMeCompletion();
vmecompletion.SetOrderId(order_id);
vmecompletion.SetTxnNumber(txn_number);
vmecompletion.SetAmount(comp_amount);
vmecompletion.SetCryptType(crypt_type);
vmecompletion.SetDynamicDescriptor(dynamic_descriptor);
```

Sample VdotMeCompletion - CA

```
vmecompletion.SetCustId(cust_id);
vmecompletion.SetShipIndicator(ship_indicator);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(vmecompletion);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("StatusCode = " + receipt.GetStatusCode());
Console.WriteLine("StatusMessage = " + receipt.GetStatusMessage());
Console.WriteLine("\r\nPress the enter key to exit");
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

13.6 Visa Checkout Purchase Correction

`VdotMePurchaseCorrection` is used to cancel a `VdotMeCompletion` or `VdotMePurchase` transaction that was performed in the current batch. No other transaction types can be corrected using this method.

No amount is required because it is always for 100% of the original transaction.

VdotMePurchaseCorrection transaction object definition

```
VdotMePurchaseCorrection vDotMePurchaseCorrection = new VdotMePurchaseCorrection();
```

HttpsPostRequest object for VdotMePurchaseCorrection transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.SetTransaction(vDotMePurchaseCorrection);
```

TRANSACTIONNAMEHERE transaction object values

Table 1: VdotMePurchaseCorrection transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`vDotMePurchaseCorrection.SetOrderId (order_id);`
Transaction number	String	255-character alpha-numeric	`vDotMePurchaseCorrection.SetTxnNumber (txn_number);`

Table 2: VdotMePurchaseCorrection transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`vDotMePurchaseCorrection.SetCustId(cust_id);`
Status check	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Sample VdotMePurchaseCorrection - CA

```
using System;
using Moneris;
namespace ACME
{
class TestCanadaVdotMePurchaseCorrection
{
public static void Main(string[] args)
{
string store_id = "store2";
string api_token = "yesguy";
string order_id = "VmeOrder20150626022834";
string txn_number = "737534-0_10";
string crypt_type = "7";
string cust_id = "my customer id";
string processing_country_code = "CA";
bool status_check = false;
VdotMePurchaseCorrection vDotMePurchaseCorrection = new VdotMePurchaseCorrection();
vDotMePurchaseCorrection.SetOrderId(order_id);
vDotMePurchaseCorrection.SetCustId(cust_id);
vDotMePurchaseCorrection.SetTxnNumber(txn_number);
vDotMePurchaseCorrection.SetCryptType(crypt_type);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(vDotMePurchaseCorrection);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
```

Sample VdotMePurchaseCorrection - CA

```
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("StatusCode = " + receipt.GetStatusCode());
Console.WriteLine("StatusMessage = " + receipt.GetStatusMessage());
Console.WriteLine("\r\nPress the enter key to exit");
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

13.7 Visa Checkout Refund

`VdotMeRefund` will credit a specified amount to the cardholder's credit card and update their Visa Checkout transaction history. A refund can be sent up to the full value of the original `VdotMeCompletion` or `VdotMePurchase`.

VdotMeRefund transaction object definition

```
VdotMeRefund vDotMeRefundRequest = new VdotMeRefund();
```

HttpsPostRequest object for VdotMeRefund transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(vDotMeRefundRequest);
```

VdotMeRefund transaction object values

Table 1: VdotMeRefund transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`vDotMeRefundRequest.SetOrderId(order_id);`
Amount	String	9-character decimal	`vDotMeRefundRequest.SetAmount(amount);`
Transaction number	String	255-character alpha-numeric	`vDotMeRefundRequest.SetTxnNumber(txn_number);`
E-commerce indicator	String	1-character alpha-numeric	`vDotMeRefundRequest.SetCryptType(crypt);`

Table 2: VdotMeRefund transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`vDotMeRefundRequest.SetCustId(cust_id);`
Dynamic descriptor	String	20-character alpha-numeric	`vDotMeRefundRequest.SetDynamicDescriptor(dynamic_descriptor);`
Status check	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Sample VdotMeRefund - CA

```
using System;
using Moneris;
namespace ACME
{
class TestCanadaVdotMeRefund
{
public static void Main(string[] args)
{
string store_id = "store2";
string api_token = "yesguy";
string order_id = "VmeOrder20150626023725";
string txn_number = "737545-0_10";
string amount = "1.00";
string crypt_type = "7";
string dynamic_descriptor = "inv 123";
string cust_id = "my customer id";
string processing_country_code = "CA";
bool status_check = false;
VdotMeRefund vDotMeRefundRequest = new VdotMeRefund();
vDotMeRefundRequest.SetOrderId(order_id);
vDotMeRefundRequest.SetAmount(amount);
vDotMeRefundRequest.SetCustId(cust_id);
vDotMeRefundRequest.SetTxnNumber(txn_number);
vDotMeRefundRequest.SetCryptType(crypt_type);
```

Sample VdotMeRefund - CA

```
vDotMeRefundRequest.SetDynamicDescriptor(dynamic_descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(vDotMeRefundRequest);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("StatusCode = " + receipt.GetStatusCode());
Console.WriteLine("StatusMessage = " + receipt.GetStatusMessage());
Console.WriteLine("\r\nPress the enter key to exit");
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

13.8 Visa Checkout Information

`VdotMeInfo` will get customer information from their Visa Checkout wallet. The details returned are dependent on what the customer has stored in Visa Checkout.

VdotMeInfo transaction object definition

```
VdotMeInfo vmeinfo = new VdotMeInfo();
```

HttpsPostRequest object for VdotMeInfo transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(vmeinfo);
```

VdotMeInfo transaction object values

Table 1: VdotMeInfo transaction object mandatory values

Value	Type	Limits	Set Method
Call ID	String	20-character numeric	`vmeinfo.SetCallId(call_id);`

Table 2: VdotMeInfo transaction object optional values

Value	Type	Limits	Set Method
Status check	Boolean	true/false	`mpgReq.SetStatusCheck(status_check);`

Sample VdotMeInfo - CA

```
using System;
using System.Collections.Generic;
using System.Text;
namespace Moneris
{
public class TestCanadaVdotMeInfo
{
public static void Main(string[] args)
{
string store_id = "store2";
string api_token = "yesguy";
string call_id = "5840726785406561048";
string processing_country_code = "CA";
bool status_check = false;
VdotMeInfo vmeinfo = new VdotMeInfo();
vmeinfo.SetCallId(call_id);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(vmeinfo);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("Response Code: " + receipt.GetResponseCode());
Console.WriteLine("Response Message: " + receipt.GetMessage());
Console.WriteLine("Currency Code: " + receipt.GetCurrencyCode());
Console.WriteLine("Payment Totals: " + receipt.GetPaymentTotal());
Console.WriteLine("User First Name: " + receipt.GetUserFirstName());
Console.WriteLine("User Last Name: " + receipt.GetUserLastName());
Console.WriteLine("Username: " + receipt.GetUserName());
Console.WriteLine("User Email: " + receipt.GetUserEmail());
Console.WriteLine("Encrypted User ID: " + receipt.GetEncUserId());
Console.WriteLine("Creation Time Stamp: " + receipt.GetCreationTimeStamp());
Console.WriteLine("Name on Card: " + receipt.GetNameOnCard());
Console.WriteLine("Expiration Month: " + receipt.GetExpirationDateMonth());
Console.WriteLine("Expiration Year: " + receipt.GetExpirationDateYear());
Console.WriteLine("Last 4 Digits: " + receipt.GetLastFourDigits());
Console.WriteLine("Bin Number (6 Digits): " + receipt.GetBinSixDigits());
Console.WriteLine("Card Brand: " + receipt.GetCardBrand());
```

Sample VdotMeInfo - CA

```
Console.WriteLine("Card Type: " + receipt.GetVdotMeCardType());
Console.WriteLine("Billing Person Name: " + receipt.GetPersonName());
Console.WriteLine("Billing Address Line 1: " + receipt.GetBillingAddressLine1());
Console.WriteLine("Billing City: " + receipt.GetBillingCity());
Console.WriteLine("Billing State/Province Code: " + receipt.GetBillingStateProvinceCode());
Console.WriteLine("Billing Postal Code: " + receipt.GetBillingPostalCode());
Console.WriteLine("Billing Country Code: " + receipt.GetBillingCountryCode());
Console.WriteLine("Billing Phone: " + receipt.GetBillingPhone());
Console.WriteLine("Billing ID: " + receipt.GetBillingId());
Console.WriteLine("Billing Verification Status: " + receipt.GetBillingVerificationStatus());
Console.WriteLine("Partial Shipping Country Code: " + receipt.GetPartialShippingCountryCode());
Console.WriteLine("Partial Shipping Postal Code: " + receipt.GetPartialShippingPostalCode());
Console.WriteLine("Shipping Person Name: " + receipt.GetShippingPersonName());
Console.WriteLine("Shipping Address Line 1: " + receipt.GetShipAddressLine1());
Console.WriteLine("Shipping City: " + receipt.GetShippingCity());
Console.WriteLine("Shipping State/Province Code: " + receipt.GetShippingStateProvinceCode());
Console.WriteLine("Shipping Postal Code: " + receipt.GetShippingPostalCode());
Console.WriteLine("Shipping Country Code: " + receipt.GetShippingCountryCode());
Console.WriteLine("Shipping Phone: " + receipt.GetShippingPhone());
Console.WriteLine("Shipping Default: " + receipt.GetShippingDefault());
Console.WriteLine("Shipping ID: " + receipt.GetShippingId());
Console.WriteLine("Shipping Verification Status: " + receipt.GetShippingVerificationStatus());
Console.WriteLine("isExpired: " + receipt.GetIsExpired());
Console.WriteLine("Base Image File Name: " + receipt.GetBaseImageFileName());
Console.WriteLine("Height: " + receipt.GetHeight());
Console.WriteLine("Width: " + receipt.GetWidth());
Console.WriteLine("Issuer Bid: " + receipt.GetIssuerBid());
Console.WriteLine("Risk Advice: " + receipt.GetRiskAdvice());
Console.WriteLine("Risk Score: " + receipt.GetRiskScore());
Console.WriteLine("AVS Response Code: " + receipt.GetAvsResponseCode());
Console.WriteLine("CVV Response Code: " + receipt.GetCvvResponseCode());
Console.WriteLine("\r\nPress the enter key to exit");
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

14 MasterCard MasterPass

- "Transaction Types - MasterPass" below
- "Transaction Flow for MasterPass Transactions" on the facing page

MasterPass is a digital wallet service offered to MasterCard cardholders. MasterPass functionality can be integrated into the Moneris Payment Gateway via the API.

14.1 Transaction Types - MasterPass

Below is a list of transactions supported by the MasterPass API.

paypass_send_shopping cart
Mandatory call to Moneris to obtain MPRequestToken and MPRedirectUrl. Your customers must be redirect to Url specified in MPRedirectUrl to proceed with checkout.

paypass_retrieve_checkout_data
Mandatory call to Moneris after customer is redirect back to your site. This call allows you to obtain customer profile details such as billing address, shipping address, masked card number, expiry date, customer contact information and cavv value.

paypass_purchase
Call to Moneris to obtain funds from the MasterPass oauthtoken and ready them for deposit into the merchant account. This call can only made after making a `paypass_retreive_checkout_data` call.

paypass_preauth
Call to Moneris to verify funds on the MasterPass oauthtoken and reserve those funds for your merchant account. The funds are locked for a specified amount of time, based on the card issuer. This call can only made after making `paypass_retreive_checkout_data` call. To retrieve the funds from this call so that they may be settled in the merchant's account, a completion must be performed.

paypass_completion
Call to Moneris to obtain funds reserved by `paypass_preauth` or `paypass_cavv_preauth`. This transaction call retrieves the locked funds and readies them for settlement into the merchant's account. This call must be made typically within 72 hours of performing `paypass_preauth` or `paypass_cavv_preauth`.

paypass_purchasecorrection
Call to Moneris to void the `paypass_purchase` or `paypass_completion` the same day* that they occurred on.

paypass_refund
Call to Moneris to refund against a `paypass_purchase` or `paypass_completion` to refund any part or all of the transaction.

paypass_cavv_purchase
Verified by Visa or MasterCard SecureCode transaction call to Moneris using Cavv value to obtain funds from the MasterPass oauthtoken and ready them for deposit into the merchant account. This call can only made after making `paypass_retreive_checkout_data` call. Cavv value can be obtained from `paypass_retrieve_checkout_data` or by performing a `paypass_txn` call.

paypass_cavv_preauth
Verified by Visa/MasterCard SecureCode transaction call to Moneris using Cavv value to verify funds on the MasterPass oauthtoken and reserve those funds for your merchant account. The funds are locked for a specified amount of time, based on the card issuer. This call can only made after making a `paypass_retreive_checkout_data` call. To retrieve the funds from this call so that they may be settled in the merchant's account, a completion must be performed.

paypass_txn
Optional call to Moneris to perform Verified by Visa or MasterCard SecureCode MPI transaction to obtain Cavv value. This call should be performed if you don't receive AuthenticationOptionsCAvv value in response after performing `paypass_retrieve_checkout_data`.

14.2 Transaction Flow for MasterPass Transactions

1. Once your customer has selected MasterPass and proceeds to pay, you must make a `paypass_send_shopping_cart` call to Moneris to obtain MPRequestToken and MPRedirectUrl.
2. Your website will then redirect to your customer to url specified in MPRedirectUrl from step 1.
3. Once customer completes their process on MasterPass, MasterPass will redirect customer back to your site along with response.
4. Using variables from step 3 response, you must then make `paypass_retrieve_checkout_data` call to Moneris. This call will verify the response token from MasterPass response and it will provide you with customer profile details from MasterPass.
5. **OPTIONAL**: Calculate shipping cost using data from `paypass_retrieve_checkout_data` call and add it to the total amount
6. Now, make a `paypass_purchase` or `paypass_preauth` call to Moneris to charge the card and obtain funds.

NOTE	If paypass_retrieve_checkout_data provides you with CAVV data, you can perform paypass_cavv_purchase or paypass_cavv_preauth transaction.

14.3 MasterPass Send Shopping Cart

PayPassSendShoppingCart transaction object definition

```
PaypassSendShoppingCart paypassSendShoppingCart = new PaypassSendShoppingCart
();
```

HttpsPostRequest for PayPassSendShoppingCart transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(paypassSendShoppingCart);
```

PaypassSendShoppingCart transaction is a mandatory call to Moneris to obtain the MPRequestToken and MPRedirectURL. Your customers must be redirect to Url specified in MPRedirectUrl to proceed with checkout. Please refer to Appendix A. Definition of Request Fields.

PayPassSendShoppingCart transaction object values

Table 1: PayPassSendShoppingCart transaction object mandatory values

Value	Type	Limits	Set Method
Subtotal			`paypassSendShoppingCart .SetSubtotal(subtotal);`
Suppress shipping address			`paypassSendShoppingCart .SetSuppressShippingAddress (suppress_shipping_address);`

Table 2: PayPassSendShoppingCart transaction object optional values

Value	Type	Limits	Set Method
Merchant callback URL			`paypassSendShoppingCart.SetMerchantCallbackUrl ("");`
Merchant card list			`paypassSendShoppingCart.SetMerchantCardList("");`

Sample PayPassSendShoppingCart - CA

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

public class TestPaypassSendShoppingCart
{
public static void Main(string[] args)
{
string host = "esqa.moneris.com";
string store_id = "moneris";
string api_token = "hurgle";
string subtotal = "1.00";
string suppress_shipping_address = "false";
PaypassSendShoppingCart paypassSendShoppingCart = new PaypassSendShoppingCart(subtotal,
suppress_shipping_address);
/*Optional*/
paypassSendShoppingCart.SetMerchantCallbackUrl("");
paypassSendShoppingCart.SetMerchantCardList("");
paypassSendShoppingCart.SetMerchantCallbackUrl("");

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

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

try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
```

Sample PayPassSendShoppingCart - CA

```
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("MPRequestToken = " + receipt.GetMPRequestToken());
Console.WriteLine("MPRedirectUrl = " + receipt.GetMPRedirectUrl());
//PayPassInfo
Console.WriteLine("\nCardBrandId = " + receipt.GetCardBrandId());
Console.WriteLine("CardBrandName = " + receipt.GetCardBrandName());
Console.WriteLine("CardBillingAddressCity = " + receipt.GetCardBillingAddressCity());
Console.WriteLine("CardBillingAddressCountry = " +
receipt.GetCardBillingAddressCountry());
Console.WriteLine("CardBillingAddressCountrySubdivision = " +
receipt.GetCardBillingAddressCountrySubdivision());
Console.WriteLine("CardBillingAddressLine1 = " + receipt.GetCardBillingAddressLine1());
Console.WriteLine("CardBillingAddressLine2 = " + receipt.GetCardBillingAddressLine2());
Console.WriteLine("CardBillingAddressPostalCode = " +
receipt.GetCardBillingAddressPostalCode());
Console.WriteLine("CardCardHolderName = " + receipt.GetCardCardHolderName());
Console.WriteLine("CardExpiryMonth = " + receipt.GetCardExpiryMonth());
Console.WriteLine("CardExpiryYear = " + receipt.GetCardExpiryYear());
Console.WriteLine("TransactionId = " + receipt.GetTransactionId());
Console.WriteLine("ContactEmailAddress = " + receipt.GetContactEmailAddress());
Console.WriteLine("ContactFirstName = " + receipt.GetContactFirstName());
Console.WriteLine("ContactLastName = " + receipt.GetContactLastName());
Console.WriteLine("ContactPhoneNumber = " + receipt.GetContactPhoneNumber());
Console.WriteLine("ShippingAddressCity = " + receipt.GetShippingAddressCity());
Console.WriteLine("ShippingAddressCountry = " + receipt.GetShippingAddressCountry());
Console.WriteLine("ShippingAddressCountrySubdivision = " +
receipt.GetShippingAddressCountrySubdivision());
Console.WriteLine("ShippingAddressLine1 = " + receipt.GetShippingAddressLine1());
Console.WriteLine("ShippingAddressLine2 = " + receipt.GetShippingAddressLine2());
Console.WriteLine("ShippingAddressPostalCode = " +
receipt.GetShippingAddressPostalCode());
Console.WriteLine("ShippingAddressRecipientName = " +
receipt.GetShippingAddressRecipientName());
Console.WriteLine("ShippingAddressRecipientPhoneNumber = " +
receipt.GetShippingAddressRecipientPhoneNumber());
Console.WriteLine("PayPassWalletIndicator = " + receipt.GetPayPassWalletIndicator());
Console.WriteLine("AuthenticationOptionsAuthenticateMethod = " +
receipt.GetAuthenticationOptionsAuthenticateMethod());
Console.WriteLine("AuthenticationOptionsCardEnrollmentMethod = " +
receipt.GetAuthenticationOptionsCardEnrollmentMethod());
Console.WriteLine("CardAccountNumber = " + receipt.GetCardAccountNumber());


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

Sample PayPassSendShoppingCart - CA

```
}
}
```

14.4 MasterPass Retrieve Checkout Data

PaypassRetrieveCheckoutData transaction object definition

```
PaypassRetrieveCheckoutData paypassRetrieveCheckoutData = new Pay-
passRetrieveCheckoutData();
```

HttpsPostRequest for PaypassRetrieveCheckoutData transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(paypassRetrieveCheckoutData);
```

PaypassRetrieveCheckoutData transaction is a mandatory call to Moneris in order to obtain customer profile details such as billing address, shipping address, masked card number, expiry date, customer contact information and cavv value. Please refer to Appendix A. Definition of Request Fields.

Table 1: PaypassRetrieveCheckoutData transaction object mandatory values

Value	Type	Limits	Set Method
Oauth token	String	alphanumeric	`paypassRetrieveCheckoutData.SetOauthToken (oauth_token);`
Oauth verifier	String	alphanumeric	`paypassRetrieveCheckoutData.SetOauthVerifier (oauth_verifier);`
Checkout resource URL			`paypassRetrieveCheckoutData.SetCheck-outResourceUrl(checkout_resource_url);`

Sample PaypassRetrieveCheckoutData - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaPaypassRetrieveCheckoutData
{
public static void Main(string[] args)
{
string store_id = "moneris";
string api_token = "hurgle";
string oauth_token = "407b74e40168289218f22265d8f7fe4c";
string oauth_verifier = "90a75e9269832a27eb50ad8a2f73ab9d";
```

Sample PaypassRetrieveCheckoutData - CA

```
string checkout_resource_url = "https://esqa.moneris.com";
string processing_country_code = "CA";

PaypassRetrieveCheckoutData paypassRetrieveCheckoutData = new PaypassRetrieveCheckoutData();
paypassRetrieveCheckoutData.SetOauthToken(oauth_token);
paypassRetrieveCheckoutData.SetOauthVerifier(oauth_verifier);
paypassRetrieveCheckoutData.SetCheckoutResourceUrl(checkout_resource_url);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(paypassRetrieveCheckoutData);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("MPRequestToken = " + receipt.GetMPRequestToken());
Console.WriteLine("MPRedirectUrl = " + receipt.GetMPRedirectUrl());
//PayPassInfo
Console.WriteLine("\nCardBrandId = " + receipt.GetCardBrandId());
Console.WriteLine("CardBrandName = " + receipt.GetCardBrandName());
Console.WriteLine("CardBillingAddressCity = " + receipt.GetCardBillingAddressCity());
Console.WriteLine("CardBillingAddressCountry = " + receipt.GetCardBillingAddressCountry());
Console.WriteLine("CardBillingAddressCountrySubdivision = " +
    receipt.GetCardBillingAddressCountrySubdivision());
Console.WriteLine("CardBillingAddressLine1 = " + receipt.GetCardBillingAddressLine1());
Console.WriteLine("CardBillingAddressLine2 = " + receipt.GetCardBillingAddressLine2());
Console.WriteLine("CardBillingAddressPostalCode = " + receipt.GetCardBillingAddressPostalCode());
Console.WriteLine("CardCardHolderName = " + receipt.GetCardCardHolderName());
Console.WriteLine("CardExpiryMonth = " + receipt.GetCardExpiryMonth());
Console.WriteLine("CardExpiryYear = " + receipt.GetCardExpiryYear());
Console.WriteLine("TransactionId = " + receipt.GetTransactionId());
Console.WriteLine("ContactEmailAddress = " + receipt.GetContactEmailAddress());
Console.WriteLine("ContactFirstName = " + receipt.GetContactFirstName());
Console.WriteLine("ContactLastName = " + receipt.GetContactLastName());
Console.WriteLine("ContactPhoneNumber = " + receipt.GetContactPhoneNumber());
Console.WriteLine("ShippingAddressCity = " + receipt.GetShippingAddressCity());
Console.WriteLine("ShippingAddressCountry = " + receipt.GetShippingAddressCountry());
Console.WriteLine("ShippingAddressCountrySubdivision = " +
    receipt.GetShippingAddressCountrySubdivision());
Console.WriteLine("ShippingAddressLine1 = " + receipt.GetShippingAddressLine1());
Console.WriteLine("ShippingAddressLine2 = " + receipt.GetShippingAddressLine2());
Console.WriteLine("ShippingAddressPostalCode = " + receipt.GetShippingAddressPostalCode());
```

Sample PaypassRetrieveCheckoutData - CA

```
Console.WriteLine("ShippingAddressRecipientName = " + receipt.GetShippingAddressRecipientName());
Console.WriteLine("ShippingAddressRecipientPhoneNumber = " +
    receipt.GetShippingAddressRecipientPhoneNumber());
Console.WriteLine("PayPassWalletIndicator = " + receipt.GetPayPassWalletIndicator());
Console.WriteLine("AuthenticationOptionsAuthenticateMethod = " +
    receipt.GetAuthenticationOptionsAuthenticateMethod());
Console.WriteLine("AuthenticationOptionsCardEnrollmentMethod = " +
    receipt.GetAuthenticationOptionsCardEnrollmentMethod());
Console.WriteLine("CardAccountNumber = " + receipt.GetCardAccountNumber());
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

14.5 MasterPass Purchase

PaypassPurchase transaction object definition

```
PaypassPurchase paypassPurchase = new PaypassPurchase();
```

HttpsPostRequest object for PaypassPurchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(paypassPurchase);
```

PaypassPurchase transaction is a call to Moneris to obtain funds from the MasterPass oauthtoken and ready them for deposit into the merchant account. PaypassPurchase requires some mandatory variables (store_id, api_token, order_id and mp_request_token). There are also a two optional variables such as cust_id and dynamic_descriptor available. This call can only made after making paypass_retreive_checkout_data call. Please refer to Appendix A. Definition of Request Fields

Table 1: PaypassPurchase transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`paypassPurchase.SetOrderId(order_id);`
Amount	String	9-character decimal	`paypassPurchase.SetAmount(amount);`
MP request token	String	255-character alpha-numeric	`paypassPurchase.SetMpRequestToken(mp_request_token);`
E-commerce indicator	String	1-character alpha-numeric	`paypassPurchase.SetCryptType(crypt);`

Table 2: PaypassPurchase transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`paypassPurchase.SetCustId(cust_id);`
Dynamic descriptor	String	20-character alpha-numeric	`paypassPurchase.SetDynamicDescriptor (dynamic_descriptor);`

Sample PaypassPurchase - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaPaypassPurchase
{
public static void Main(string[] args)
{
string store_id = "moneris";
string api_token = "hurgle";
string order_id = "Test" + DateTime.Now.ToString("yyyyMMddhhmmss");
string cust_id = "customer1";
string amount = "1.00";
string mp_request_token = "47edbc7b56cac2b7ed27ab4d62214407";
string crypt_type = "7";
string dynamic_descriptor = "paypass1";
string processing_country_code = "CA";
PaypassPurchase paypassPurchase = new PaypassPurchase();
paypassPurchase.SetOrderId(order_id);
paypassPurchase.SetCustId(cust_id);
paypassPurchase.SetAmount(amount);
paypassPurchase.SetMpRequestToken(mp_request_token);
paypassPurchase.SetCryptType(crypt_type);
paypassPurchase.SetDynamicDescriptor(dynamic_descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(paypassPurchase);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
```

Sample PaypassPurchase - CA

```
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("MPRequestToken = " + receipt.GetMPRequestToken());
Console.WriteLine("MPRedirectUrl = " + receipt.GetMPRedirectUrl());
//PayPassInfo
Console.WriteLine("\nCardBrandId = " + receipt.GetCardBrandId());
Console.WriteLine("CardBrandName = " + receipt.GetCardBrandName());
Console.WriteLine("CardBillingAddressCity = " + receipt.GetCardBillingAddressCity());
Console.WriteLine("CardBillingAddressCountry = " + receipt.GetCardBillingAddressCountry());
Console.WriteLine("CardBillingAddressCountrySubdivision = " +
    receipt.GetCardBillingAddressCountrySubdivision());
Console.WriteLine("CardBillingAddressLine1 = " + receipt.GetCardBillingAddressLine1());
Console.WriteLine("CardBillingAddressLine2 = " + receipt.GetCardBillingAddressLine2());
Console.WriteLine("CardBillingAddressPostalCode = " + receipt.GetCardBillingAddressPostalCode());
Console.WriteLine("CardCardHolderName = " + receipt.GetCardCardHolderName());
Console.WriteLine("CardExpiryMonth = " + receipt.GetCardExpiryMonth());
Console.WriteLine("CardExpiryYear = " + receipt.GetCardExpiryYear());
Console.WriteLine("TransactionId = " + receipt.GetTransactionId());
Console.WriteLine("ContactEmailAddress = " + receipt.GetContactEmailAddress());
Console.WriteLine("ContactFirstName = " + receipt.GetContactFirstName());
Console.WriteLine("ContactLastName = " + receipt.GetContactLastName());
Console.WriteLine("ContactPhoneNumber = " + receipt.GetContactPhoneNumber());
Console.WriteLine("ShippingAddressCity = " + receipt.GetShippingAddressCity());
Console.WriteLine("ShippingAddressCountry = " + receipt.GetShippingAddressCountry());
Console.WriteLine("ShippingAddressCountrySubdivision = " +
    receipt.GetShippingAddressCountrySubdivision());
Console.WriteLine("ShippingAddressLine1 = " + receipt.GetShippingAddressLine1());
Console.WriteLine("ShippingAddressLine2 = " + receipt.GetShippingAddressLine2());
Console.WriteLine("ShippingAddressPostalCode = " + receipt.GetShippingAddressPostalCode());
Console.WriteLine("ShippingAddressRecipientName = " + receipt.GetShippingAddressRecipientName());
Console.WriteLine("ShippingAddressRecipientPhoneNumber = " +
    receipt.GetShippingAddressRecipientPhoneNumber());
Console.WriteLine("PayPassWalletIndicator = " + receipt.GetPayPassWalletIndicator());
Console.WriteLine("AuthenticationOptionsAuthenticateMethod = " +
    receipt.GetAuthenticationOptionsAuthenticateMethod());
Console.WriteLine("AuthenticationOptionsCardEnrollmentMethod = " +
    receipt.GetAuthenticationOptionsCardEnrollmentMethod());
Console.WriteLine("CardAccountNumber = " + receipt.GetCardAccountNumber());
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

14.6 MasterPass PreAuth

PaypassPreAuth transaction object definition

```
PaypassPreauth paypassPreauth = new PaypassPreauth();
```

HttpsPostRequest object for PaypassPreAuth

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(paypassPreauth);
```

PaypassPreauth is virtually identical to the PaypassPurchase with the exception of the transaction type. It is 'PreAuth' instead of 'Purchase'. Like the PaypassPurchase example, PaypassPreauth's require some mandatory variables (store_id, api_token, order_id and mp_request_token). There are also a two optional variables such as cust_id and dynamic_descriptor available. Please refer to What Information do I need to include in a Transaction Request.

PaypassPreAuth Transaction Values

Table 1: PaypassPreAuth Mandatory Values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`paypassPreauth.SetOrderId(order_id);`
Transaction number	String	255-character alpha-numeric	`paypassPurchase.SetTxnNumber(txn_number);`
Amount	String	9-character decimal	`paypassPreauth.SetAmount(amount);`
E-commerce indicator	String	1-character alpha-numeric	`paypassPreauth.SetCryptType(crypt);`
MP request token	String	255-character alpha-numeric	`paypassPreauth.SetMpRequestToken(mp_request_token);`

Table 2: PaypassPreAuth Optional Values

Value	Type	Limits	Set Method
Dynamic descriptor	String	20-character alpha-numeric	`paypassPreauth.SetDynamicDescriptor(dynamic_descriptor);`
Customer ID	String	50-character alpha-numeric	`paypassPreauth.SetCustId(cust_id);`

Sample PaypassPreAuth - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaPaypassPreauth
{
public static void Main(string[] args)
{
string store_id = "moneris";
string api_token = "hurgle";
string order_id = "paypass_test1";
string cust_id = "customer1";
string amount = "1.00";
```

Sample PaypassPreAuth - CA

```
string mp_request_token = "47edbc7b56cac2b7ed27ab4d62214407";
string crypt_type = "7";
string dynamic_descriptor = "paypass1";
string processing_country_code = "CA";
PaypassPreauth paypassPreauth = new PaypassPreauth();
paypassPreauth.SetOrderId(order_id);
paypassPreauth.SetCustId(cust_id);
paypassPreauth.SetAmount(amount);
paypassPreauth.SetMpRequestToken(mp_request_token);
paypassPreauth.SetCryptType(crypt_type);
paypassPreauth.SetDynamicDescriptor(dynamic_descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(paypassPreauth);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("MPRequestToken = " + receipt.GetMPRequestToken());
Console.WriteLine("MPRedirectUrl = " + receipt.GetMPRedirectUrl());
//PayPassInfo
Console.WriteLine("\nCardBrandId = " + receipt.GetCardBrandId());
Console.WriteLine("CardBrandName = " + receipt.GetCardBrandName());
Console.WriteLine("CardBillingAddressCity = " + receipt.GetCardBillingAddressCity());
Console.WriteLine("CardBillingAddressCountry = " + receipt.GetCardBillingAddressCountry());
Console.WriteLine("CardBillingAddressCountrySubdivision = " +
    receipt.GetCardBillingAddressCountrySubdivision());
Console.WriteLine("CardBillingAddressLine1 = " + receipt.GetCardBillingAddressLine1());
Console.WriteLine("CardBillingAddressLine2 = " + receipt.GetCardBillingAddressLine2());
Console.WriteLine("CardBillingAddressPostalCode = " + receipt.GetCardBillingAddressPostalCode());
Console.WriteLine("CardCardHolderName = " + receipt.GetCardCardHolderName());
Console.WriteLine("CardExpiryMonth = " + receipt.GetCardExpiryMonth());
Console.WriteLine("CardExpiryYear = " + receipt.GetCardExpiryYear());
Console.WriteLine("TransactionId = " + receipt.GetTransactionId());
Console.WriteLine("ContactEmailAddress = " + receipt.GetContactEmailAddress());
Console.WriteLine("ContactFirstName = " + receipt.GetContactFirstName());
Console.WriteLine("ContactLastName = " + receipt.GetContactLastName());
Console.WriteLine("ContactPhoneNumber = " + receipt.GetContactPhoneNumber());
Console.WriteLine("ShippingAddressCity = " + receipt.GetShippingAddressCity());
Console.WriteLine("ShippingAddressCountry = " + receipt.GetShippingAddressCountry());
Console.WriteLine("ShippingAddressCountrySubdivision = " +
```

Sample PaypassPreAuth - CA

```
    receipt.GetShippingAddressCountrySubdivision());
Console.WriteLine("ShippingAddressLine1 = " + receipt.GetShippingAddressLine1());
Console.WriteLine("ShippingAddressLine2 = " + receipt.GetShippingAddressLine2());
Console.WriteLine("ShippingAddressPostalCode = " + receipt.GetShippingAddressPostalCode());
Console.WriteLine("ShippingAddressRecipientName = " + receipt.GetShippingAddressRecipientName());
Console.WriteLine("ShippingAddressRecipientPhoneNumber = " +
    receipt.GetShippingAddressRecipientPhoneNumber());
Console.WriteLine("PayPassWalletIndicator = " + receipt.GetPayPassWalletIndicator());
Console.WriteLine("AuthenticationOptionsAuthenticateMethod = " +
    receipt.GetAuthenticationOptionsAuthenticateMethod());
Console.WriteLine("AuthenticationOptionsCardEnrollmentMethod = " +
    receipt.GetAuthenticationOptionsCardEnrollmentMethod());
Console.WriteLine("CardAccountNumber = " + receipt.GetCardAccountNumber());
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

14.7 MasterPass Purchase with Cavv

PaypassCavvPurchase transaction object definition

```
PaypassCavvPurchase paypassPurchase = new PaypassCavvPurchase();
```

HttpsPostRequest for PaypassCavvPurchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(paypassPurchase);
```

PaypassCavvPurchase requires few mandatory variables (store_id, api_token, order_id and mp_request_token) and is a Verified by Visa or MasterCard SecureCode transaction call to retrieve the funds and ready them to be deposited into the merchant account. The PaypassCavvPurchase call can only be made after the PapassRetrieveCheckoutData. There are also two optional variables such as cust_id and dynamic_descriptor available. Please refer to Appendix A. Definition of Request Fields for variable definitions.

PaypassCavvPurchase transaction object values

Table 1: PaypassCavvPurchase transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`paypassPurchase.SetOrderId(order_id);`
Amount	String	9-character decimal	`paypassPurchase.SetAmount(amount);`

Value	Type	Limits	Set Method
CAVV	String	50-character alpha-numeric	`paypassPurchase.SetCavv(cavv);`
E-commerce indicator	String	1-character alpha-numeric	`paypassPurchase.SetCryptType(crypt);`
MP request token	String	255-character alpha-numeric	`paypassPurchase.SetMpRequestToken(mp_request_token);`

Table 2: PaypassCavvPurchase transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`paypassPurchase.SetCustId(cust_id);`
Dynamic descriptor	String	20-character alpha-numeric	`paypassPurchase.SetDynamicDescriptor(dynamic_descriptor);`

Sample PaypassCavvPurchase - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaPaypassCavvPurchase
{
public static void Main(string[] args)
{
string store_id = "moneris";
string api_token = "hurgle";
string order_id = "Test" + DateTime.Now.ToString("yyyyMMddhhmmss");
string cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA";
string cust_id = "customer1";
string amount = "1.00";
string mp_request_token = "47edbc7b56cac2b7ed27ab4d62214407";
string crypt_type = "7";
string dynamic_descriptor = "paypass1";
string processing_country_code = "CA";
PaypassCavvPurchase paypassPurchase = new PaypassCavvPurchase();
paypassPurchase.SetOrderId(order_id);
paypassPurchase.SetCavv(cavv);
paypassPurchase.SetCustId(cust_id);
paypassPurchase.SetAmount(amount);
paypassPurchase.SetMpRequestToken(mp_request_token);
paypassPurchase.SetCryptType(crypt_type);
paypassPurchase.SetDynamicDescriptor(dynamic_descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(paypassPurchase);
```

Sample PaypassCavvPurchase - CA

```
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("MPRequestToken = " + receipt.GetMPRequestToken());
Console.WriteLine("MPRedirectUrl = " + receipt.GetMPRedirectUrl());
//PayPassInfo
Console.WriteLine("\nCardBrandId = " + receipt.GetCardBrandId());
Console.WriteLine("CardBrandName = " + receipt.GetCardBrandName());
Console.WriteLine("CardBillingAddressCity = " + receipt.GetCardBillingAddressCity());
Console.WriteLine("CardBillingAddressCountry = " + receipt.GetCardBillingAddressCountry());
Console.WriteLine("CardBillingAddressCountrySubdivision = " +
    receipt.GetCardBillingAddressCountrySubdivision());
Console.WriteLine("CardBillingAddressLine1 = " + receipt.GetCardBillingAddressLine1());
Console.WriteLine("CardBillingAddressLine2 = " + receipt.GetCardBillingAddressLine2());
Console.WriteLine("CardBillingAddressPostalCode = " + receipt.GetCardBillingAddressPostalCode());
Console.WriteLine("CardCardHolderName = " + receipt.GetCardCardHolderName());
Console.WriteLine("CardExpiryMonth = " + receipt.GetCardExpiryMonth());
Console.WriteLine("CardExpiryYear = " + receipt.GetCardExpiryYear());
Console.WriteLine("TransactionId = " + receipt.GetTransactionId());
Console.WriteLine("ContactEmailAddress = " + receipt.GetContactEmailAddress());
Console.WriteLine("ContactFirstName = " + receipt.GetContactFirstName());
Console.WriteLine("ContactLastName = " + receipt.GetContactLastName());
Console.WriteLine("ContactPhoneNumber = " + receipt.GetContactPhoneNumber());
Console.WriteLine("ShippingAddressCity = " + receipt.GetShippingAddressCity());
Console.WriteLine("ShippingAddressCountry = " + receipt.GetShippingAddressCountry());
Console.WriteLine("ShippingAddressCountrySubdivision = " +
    receipt.GetShippingAddressCountrySubdivision());
Console.WriteLine("ShippingAddressLine1 = " + receipt.GetShippingAddressLine1());
Console.WriteLine("ShippingAddressLine2 = " + receipt.GetShippingAddressLine2());
Console.WriteLine("ShippingAddressPostalCode = " + receipt.GetShippingAddressPostalCode());
Console.WriteLine("ShippingAddressRecipientName = " + receipt.GetShippingAddressRecipientName());
Console.WriteLine("ShippingAddressRecipientPhoneNumber = " +
    receipt.GetShippingAddressRecipientPhoneNumber());
Console.WriteLine("PayPassWalletIndicator = " + receipt.GetPayPassWalletIndicator());
Console.WriteLine("AuthenticationOptionsAuthenticateMethod = " +
    receipt.GetAuthenticationOptionsAuthenticateMethod());
Console.WriteLine("AuthenticationOptionsCardEnrollmentMethod = " +
    receipt.GetAuthenticationOptionsCardEnrollmentMethod());
Console.WriteLine("CardAccountNumber = " + receipt.GetCardAccountNumber());
}
catch (Exception e)
{
Console.WriteLine(e);
```

Sample PaypassCavvPurchase - CA

```
    }
    }
    }
    }
```

14.8 MasterPass PreAuth with Cavv

PaypassCavvPreAuth transaction object definition

```
PaypassCavvPreauth paypassPreauth = new PaypassCavvPreauth();
```

HttpsPostRequest for PaypassCavvPreAuth transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(paypassPreauth);
```

PaypassCAvvPreauth is virtually identical to the PaypassCavvPurchase with the exception of the transaction type. It is a ‘Preauth’ instead of a ‘Purchase’. Like the PaypassCavvPurchase example, PaypassCavvPreauth requires few mandatory variables (store_id, api_token, order_id and mp_request_token) and is a Verified by Visa or MasterCard SecureCode transaction call to lock the funds for a specified amount of time, based on the card issuer. The PaypassCavvPreauth call can only be made after the PaypassRetrieveCheckoutData. A PaypassCompletion is required to be used to secure the funds locked by a PaypassCavvPreauth transaction

If the order could not be completed for reasons such as order is cancelled, made in error or not fulfillable, PaypassCavvPreauth transaction must be reversed within 72 hours. To reverse an authorization, perform PaypassCompletion transaction for $0.00 (zero dollars).

Table 1: PaypassCavvPreAuth transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`paypassPreauth.SetOrderId(order_id);`
CAVV	String	50-character alpha-numeric	`paypassPreauth.SetCavv(cavv);`
Amount	String	9-character decimal	`paypassPreauth.SetAmount(amount);`
MP request token	String	255-character alpha-numeric	`paypassPreauth.SetMpRequestToken(mp_request_token);`

Table 2: PaypassCavvPreAuth transaction object optional values

Value	Type	Limits	Set Method
Customer ID	String	50-character alpha-numeric	`paypassPreauth.SetCustId(cust_id);`
Dynamic descriptor	String	20-character alpha-numeric	`paypassPreauth.SetDynamicDescriptor (dynamic_descriptor);`

Sample PaypassCavvPreAuth - CA

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaCavvPaypassPreauth
{
public static void Main(string[] args)
{
string store_id = "moneris";
string api_token = "hurgle";
string order_id = "Test" + DateTime.Now.ToString("yyyyMMddhhmmss");
string cavv = "AAABBJg0VhI0VniQEjRWAAAAAAA";
string cust_id = "customer1";
string amount = "1.00";
string mp_request_token = "47edbc7b56cac2b7ed27ab4d62214407";
string crypt_type = "7";
string dynamic_descriptor = "paypass1";
string processing_country_code = "CA";
PaypassCavvPreauth paypassPreauth = new PaypassCavvPreauth();
paypassPreauth.SetOrderId(order_id);
paypassPreauth.SetCavv(cavv);
paypassPreauth.SetCustId(cust_id);
paypassPreauth.SetAmount(amount);
paypassPreauth.SetMpRequestToken(mp_request_token);
paypassPreauth.SetCryptType(crypt_type);
paypassPreauth.SetDynamicDescriptor(dynamic_descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(paypassPreauth);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
```

Sample PaypassCavvPreAuth - CA

```
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("MPRequestToken = " + receipt.GetMPRequestToken());
Console.WriteLine("MPRedirectUrl = " + receipt.GetMPRedirectUrl());
//PayPassInfo
Console.WriteLine("\nCardBrandId = " + receipt.GetCardBrandId());
Console.WriteLine("CardBrandName = " + receipt.GetCardBrandName());
Console.WriteLine("CardBillingAddressCity = " + receipt.GetCardBillingAddressCity());
Console.WriteLine("CardBillingAddressCountry = " + receipt.GetCardBillingAddressCountry());
Console.WriteLine("CardBillingAddressCountrySubdivision = " +
    receipt.GetCardBillingAddressCountrySubdivision());
Console.WriteLine("CardBillingAddressLine1 = " + receipt.GetCardBillingAddressLine1());
Console.WriteLine("CardBillingAddressLine2 = " + receipt.GetCardBillingAddressLine2());
Console.WriteLine("CardBillingAddressPostalCode = " + receipt.GetCardBillingAddressPostalCode());
Console.WriteLine("CardCardHolderName = " + receipt.GetCardCardHolderName());
Console.WriteLine("CardExpiryMonth = " + receipt.GetCardExpiryMonth());
Console.WriteLine("CardExpiryYear = " + receipt.GetCardExpiryYear());
Console.WriteLine("TransactionId = " + receipt.GetTransactionId());
Console.WriteLine("ContactEmailAddress = " + receipt.GetContactEmailAddress());
Console.WriteLine("ContactFirstName = " + receipt.GetContactFirstName());
Console.WriteLine("ContactLastName = " + receipt.GetContactLastName());
Console.WriteLine("ContactPhoneNumber = " + receipt.GetContactPhoneNumber());
Console.WriteLine("ShippingAddressCity = " + receipt.GetShippingAddressCity());
Console.WriteLine("ShippingAddressCountry = " + receipt.GetShippingAddressCountry());
Console.WriteLine("ShippingAddressCountrySubdivision = " +
    receipt.GetShippingAddressCountrySubdivision());
Console.WriteLine("ShippingAddressLine1 = " + receipt.GetShipAddressLine1());
Console.WriteLine("ShippingAddressLine2 = " + receipt.GetShippingAddressLine2());
Console.WriteLine("ShippingAddressPostalCode = " + receipt.GetShippingAddressPostalCode());
Console.WriteLine("ShippingAddressRecipientName = " + receipt.GetShippingAddressRecipientName());
Console.WriteLine("ShippingAddressRecipientPhoneNumber = " +
    receipt.GetShippingAddressRecipientPhoneNumber());
Console.WriteLine("PayPassWalletIndicator = " + receipt.GetPayPassWalletIndicator());
Console.WriteLine("AuthenticationOptionsAuthenticateMethod = " +
    receipt.GetAuthenticationOptionsAuthenticateMethod());
Console.WriteLine("AuthenticationOptionsCardEnrollmentMethod = " +
    receipt.GetAuthenticationOptionsCardEnrollmentMethod());
Console.WriteLine("CardAccountNumber = " + receipt.GetCardAccountNumber());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

14.9 MasterPass Completion

PaypassCompletion transaction object definition

HttpsPostRequest for PaypassCompletion transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();
```

```
mpgReq.SetTransaction(TRANSACTION-NAME-TO-COME);
```

PaypassCompletion transaction object values

Table 1: PaypassCompletion transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`TRANSACTION-NAME-TO-COME.SetOrderId (order_id);`
Completion amount	String	9-character decimal	`TRANSACTION-NAME-TO-COME.SetCompAmount (amount);`
Transaction number	String	255-character alpha-numeric	`TRANSACTION-NAME-TO-COME.SetTxnNumber (txn_number);`
E-commerce indicator	String	1-character alpha-numeric	`TRANSACTION-NAME-TO-COME.SetCryptType (crypt);`

Table 2: PaypassCompletion transaction object optional values

Value	Type	Limits	Set Method
Dynamic descriptor	String	20-character alphanumeric	`TRANSACTION-NAME-TO-COME.SetDynamicDescriptor (dynamic_descriptor);`

Sample PaypassCompletion - CA

14.10 MasterPass Refund

PaypassRefund transaction object definition

HttpsPostRequest for PaypassRefund transaction

PaypassRefund transaction object values

Table 1: PaypassRefund transaction object mandatory values

Value	Type	Limits	Set Method
Order ID	String	50-character alpha-numeric	`TRANSACTION-NAME-TO-COME.SetOrderId (order_id);`
Amount	String	9-character decimal	`TRANSACTION-NAME-TO-COME.SetAmount (amount);`

Value	Type	Limits	Set Method
Transaction number	String	255-character alphanumeric	`TRANSACTION-NAME-TO-COME.SetTxnNumber (txn_number);`
E-commerce indicator	String	1-character alphanumeric	`TRANSACTION-NAME-TO-COME.SetCryptType (crypt);`

Table 2: PaypassRefund transaction object optional values

Value	Type	Limits	Set Method
Dynamic descriptor	String	20-character alphanumeric	`TRANSACTION-NAME-TO-COME.SetDynamicDescriptor (dynamic_descriptor);`

Sample PaypassRefund - CA

```
namespace Moneris
{
using System;
public class TestCanadaRefund
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string amount = "1.00";
string crypt = "7";
string dynamic_descriptor = "123456";
string custid = "mycust9";
string order_id = "mvt3230836758";
string txn_number = "21964-0_10";
string processing_country_code = "CA";
bool status_check = false;
Refund refund = new Refund();
refund.SetTxnNumber(txn_number);
refund.SetOrderId(order_id);
refund.SetAmount(amount);
refund.SetCryptType(crypt);
refund.SetCustId(custid);
refund.SetDynamicDescriptor(dynamic_descriptor);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.SetProcCountryCode(processing_country_code);
mpgReq.SetTestMode(true); //false or comment out this line for production transactions
mpgReq.SetStoreId(store_id);
mpgReq.SetApiToken(api_token);
mpgReq.SetTransaction(refund);
mpgReq.SetStatusCheck(status_check);
mpgReq.Send();
try
{
Receipt receipt = mpgReq.GetReceipt();
Console.WriteLine("CardType = " + receipt.GetCardType());
Console.WriteLine("TransAmount = " + receipt.GetTransAmount());
Console.WriteLine("TxnNumber = " + receipt.GetTxnNumber());
Console.WriteLine("ReceiptId = " + receipt.GetReceiptId());
Console.WriteLine("TransType = " + receipt.GetTransType());
Console.WriteLine("ReferenceNum = " + receipt.GetReferenceNum());
```

Sample PaypassRefund - CA

```
Console.WriteLine("ResponseCode = " + receipt.GetResponseCode());
Console.WriteLine("ISO = " + receipt.GetISO());
Console.WriteLine("BankTotals = " + receipt.GetBankTotals());
Console.WriteLine("Message = " + receipt.GetMessage());
Console.WriteLine("AuthCode = " + receipt.GetAuthCode());
Console.WriteLine("Complete = " + receipt.GetComplete());
Console.WriteLine("TransDate = " + receipt.GetTransDate());
Console.WriteLine("TransTime = " + receipt.GetTransTime());
Console.WriteLine("Ticket = " + receipt.GetTicket());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

14.11 MasterPass Transaction

PaypassTxn transaction object definition

HttpsPostRequest for PaypassTxn transaction

PaypassTxn transaction object values

Table 1: PaypassTxn transaction object mandatory values

Value	Type	Limits	Set Method

Table 2: PaypassTxn transaction object optional values

Value	Type	Limits	Set Method

Sample PaypassTxn - CA

15 Encorporating All Available Fraud Tools

To minimize fraudulent activity in online transactions, Moneris recommends that you implement all of the fraud tools available through the Moneris Payment Gateway. These are explained below:

Address Verification Service (AVS)
Verifies the cardholder's billing address information.

Verified by Visa and MasterCard Secure Code (VBV/SecureCode)
Authenticates the cardholder at the time of an online transaction.

Card Validation Digit (CVD)
Validates that cardholder is in possession of a genuine credit card during the transaction.

Note that all responses that are returned from these verification methods are intended to provide added security and fraud prevention. The response itself does not affect the completion of a transaction. Upon receiving a response, the choice to proceed with a transaction is left entirely to the merchant.

15.1 Implementation Options

Option A

Process a Transaction Risk Management Tool query and obtain the response. You can then decide whether to continue with the transaction, abort the transaction, or use additional efraud features.

If you want to use additional efraud features, perform one or both of the following to help make your decision about whether to continue with the transaction or abort it"

- Process a VBV/SecureCode transaction and obtain the response. The merchant then makes the decision whether to continue with the transaction or to abort it.
- Process a financial transaction including AVS/CVD details and obtain the response. The merchant then makes a decision whether to continue with the transaction or to abort it.

Option B

1. Process a Transaction Risk Management Tool query and obtain the response.
2. Process a VBV/SecureCode transaction and obtain the response.
3. Process a financial transaction including AVS/CVD details and obtain the response.
4. Merchant then makes a one-time decision based on the responses received from the eFraud tools.

15.2 Implementation Checklist

The following checklists provide high-level tasks that are required as part of your implementation of the Transaction Risk Management Tool. Because each organization has certain project requirements for implementing system and process changes, this list is only a guideline, and does not cover all aspects of your project.

Download and review all of the applicable APIs and Integration Guides

Please review the sections outlined within this document that refers to the following feature

Table 114: API documentation

Document/API	Use the document if you are....
Transaction Risk Management Tool Integration Guide (Section #)	Implementing or updating your integration for the Transaction Risk Management Tool
Moneris MPI – Verified by Visa/MasterCard SecureCode – Java API Integration Guide Section #	Implementing or updating Verified by Visa and MasterCard SecureCode
Basic transaction with VS and CVD (Section#)	Implementing or updating transaction processing, AVS or CVD

Design your transaction flow and business processes

When designing your transaction flow, think about which scenarios you would like to have automated, and which scenarios you would like to have handled manually by your employees.

The “Understand Transaction Risk Management Transaction Flow” and "Handling Response Information" (page 200) sections can help you work through the design of your transaction and process flows.

Things to consider when designing your process flows:

- Processes for notifying people within your organization when there is scheduled maintenance for Moneris Payment Gateway.
- Handling refunds, canceled orders and so on.
- Communicating with customers when you will not be shipping the goods because of suspected fraud, back-ordered goods and so on.

Complete your development and testing

- The North American API - Integration Guide provides the technical details required for the development and testing. Ensure that you follow the testing instructions and data provided.

If you are an integrator

- Ensure that your solution meets the requirements for PCI-DSS/PA-DSS as applicable.
- Send an email to eproducts@moneris.com with the subject line “Certification Request”.
- Develop material to set up your customers as quickly as possible with your solution and a Moneris account. Include information such as:
    - Steps they must take to enter their store ID or API token information into your solution.
    - Any optional services that you support via Moneris Payment Gateway (such as TRMT, AVS, CVD, VBV/SecureCode and so on) so that customers can request these features.

15.3 Making a Decision

Depending on your business policies and processes, the information obtained from the fraud tools (such as AVS, CVD, VBV/SecureCode and TRMT) can help you make an informed decision about whether to accept a transaction or deny it because it is potentially fraudulent.

If you do not want to continue with a likely fraudulent transaction, you must inform the customer that you are not proceeding with their transaction.

If you are attempting to do further authentication by using the available fraud tools, but you have received an approval response instead, cancel the financial transaction by doing one of the following:

- If the original transaction is a Purchase, use a Purchase Correction or Refund transaction. You will need the original order ID and transaction number.
- If the original transaction is a Pre-Authorization, use a Completion transaction for $0.00.

Appendix A Definition of Request Fields

This appendix deals with values that belong to transaction objects. For information on values that belong to the (HttpsPostRequest) connection object, see "HttpsPostRequest Object" on page 25.

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

Note that the values listed in Table 115 are not mandatory for **every** transaction. Check the transaction definition. If it says that a value is mandatory, a further description is found here.

Table 115: Mandatory request fields

Value	Type	Limits	Sample code variable definition
	Description		
General transaction values			
Order ID	Alphanumeric	50 characters	`String order_id`
	Merchant-defined transaction identifier that must be unique for every Purchase, PreAuth and Independent Refund transaction. No two transactions of these types may have the same order ID. For Refund, Completion and Purchase Correction transactions, the order ID must be the same as that of the original transaction. **Canada**: The last 10 characters of the order ID are displayed in the “Invoice Number” field on the Merchant Direct Reports. However only letters, numbers and spaces are sent to Merchant Direct. A minimum of 3 and a maximum of 10 valid characters are sent to Merchant Direct. Only the last characters beginning after any invalid characters are sent. For example, if the order ID is **1234-567890**, only **567890** is sent to Merchant Direct. **US**: The last 32 characters of the order ID are sent on to the Client Line settlement reports. For either countries, If the order ID has fewer than 3 characters, it may display a blank or 0000000000 in the Invoice Number field.		
Amount	Decimal	9 characters	`String amount;`
	Transaction amount. Used in a number of transactions. Note that this is different from the amount used in a Completion transaction, which is an alphanumeric value. This must contain at least 3 digits, two of which are penny values. The minimum allowable value is $0.01, and the maximum allowable value is 999 999.99. Transaction amounts of $0.00 are not allowed.		

Table 115: Mandatory request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Credit card number	Numeric	20 characters (no spaces or dashes)	`String pan;`
	Most credit card numbers today are 16 digits, but some 13-digit numbers are still accepted by some issuers. This field has been intentionally expanded to 20 digits in consideration for future expansion and potential support of private label card ranges.		
Expiry date	Numeric	4 characters (YYMM format)	`String expdate;`
	Note: This is the reverse of the date displayed on the physical card, which is MMYY.		
E-Commerce indicator	Alphanumeric	1 character	`String crypt;`
	1: Mail Order / Telephone Order—Single 2: Mail Order / Telephone Order—Recurring 3: Mail Order / Telephone Order—Instalment 4: Mail Order / Telephone Order—Unknown classification 5: Authenticated e-commerce transaction (VBV) 6: Non-authenticated e-commerce transaction (VBV) 7: SSL-enabled merchant 8: Non-secure transaction (web- or email-based) 9: SET non-authenticated transaction		
Completion Amount	Decimal	9 characters	`String comp_amount;`
	Amount of a Completion transaction. This may not be equal to the amount value (described on page 258), which appeared in the original Pre-Authorization transaction.		

Table 115: Mandatory request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Transaction number	Variable characters	255 characters	`String txn_number;`
	Used when performing follow-on transactions. (That is, Completion, Purchase Correction or Refund.) This must be the value that was returned as the transaction number in the response of the original transaction. When performing a Completion, this value must reference the Pre-Authorization. When performing a Refund or a Purchase Correction, this value must reference the Completion or the Purchase.		
Authorization code	Alphanumeric	8 characters	`String auth_code;`
	Authorization code provided in the transaction response from the issuing bank. This is required for Force Post transactions.		
ECR number	String	TBD	`String ecr_no;`
	Electronic cash register number.		
MPI transaction values			
XID	Alphanumeric	20 characters	`String xid;`
	Can also be used as your order ID when using Moneris Payment Gateway.		
MD	String	1024-character alphanumeric	`String MD;`
	Information to be echoed back in the response.		
Merchant URL	String	TBD	`String merchantUrl;`
	URL to which the MPI response is to be sent.		
Accept	String		`String accept;`
	MIME types that the browser accepts		
User Agent	String		`String userAgent;`
	Browser details		
PARes	String	Variable	(Not shown)
	Value passed back to the API during the TXN, and returned to the MPI when an ACS request is made.		

Table 115: Mandatory request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Cardholder Authentication Verification Value	Alphanumeric	50 characters	`String cavv;`
	Value provided by the Moneris MPI or by a third-party MPI. It is part of a VBV/MCSC transaction.		
ACH transaction values			
Routing number	Numeric	9 characters	`String routing_num;`
	TBD		
Vault transaction values			
Data key	Alphanumeric	25-character	`String data_key;`
	Profile identifier that all future financial Vault transactions (that is, they occur after the profile was registered by a ResAddCC or ResTokenizeCC transaction) will use to associate with the saved information. The data key is generated by Moneris, and is returned to the merchant (via the Receipt object) when the profile is first registered.		
Duration	String	3-numeric	`String duration;`
	Amount of time the temporary token should be available, up to 900 seconds.		
Mag Swipe transaction values			
POS code	Numeric	2 characters	`String pos_code;`
	Under normal presentment situations, the value is `00`. If a Pre-Authorization transaction was card-present and keyed-in[1], then the POS code for the corresponding Completion transaction is `71`. In an unmanned kiosk environment where the card is present, the value is `27`. If the solution is not "merchant and cardholder present", contact Moneris for the proper POS code.		
Track2 data	Alphanumeric	40 characters	`String track2;`
	Retrieved from the mag stripe of a credit card by swiping it through a card reader, **or** the "fund guarantee" value returned by the INTERAC® Online Payment system (Canada only).		

[1]That is, a Track2Preauth transaction was submitted where the credit card number and expiry date values were sent, but track2 was left blank.

Table 115: Mandatory request fields (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Encrypted track2 data	Alphanumeric		`String enc_track2;`
	String that is retrieved by swiping or keying in a credit card number through a Moneris-provided encrypted mag swipe card reader. It is part of an encrypted keyed or swiped transaction only. This string must be retrieved by a specific device. (See below for the list of current available devices.)		
Device type	Alphanumeric	30 characters	`String device_type;`
	Type of encrypted mag swipe reader that was read the credit card. This must be a Moneris-provided device so that the values are properly encrypted and decrypted. This field is case-sensitive. Available values are: "idtech_bdk" (Canada only) "idtech" (US only).		

Note that the values listed in Table 116 are not supported by **every** transaction. Check the transaction definition. If it says that a value is optional, a further description is found here.

Table 116: Optional transaction values

Value	Type	Limits	Sample code variable definition
	Description		
General transaction values			
Customer ID	Alphanumeric	50 characters	String cust_id;
	This can be used for policy number, membership number, student ID, invoice number and so on. This field is searchable from the Moneris Merchant Resource Centre.		
Status Check	Boolean	true/false	`String status_check;`
	See "Status Check" on page 280.		
Dynamic descriptor	Alphanumeric	20 characters. Combined with merchant's business name cannot exceed 25 characters.	`String dynamic_descriptor;`
	Merchant-defined description sent on a per-transaction basis that will appear on the credit card statement appended to the merchant's business name.		
Commercial card invoice	Alphanumeric	17 characters	`String commcard_invoice;`
	(**US only**) Level 2 Invoice Number of the transaction used for Corporate Credit Card transactions (Commercial Purchasing Cards). Characters allowed for commcard_invoice: a-z, A-Z, 0-9, spaces		
Commercial card tax amount	Decimal	9 characters. Must contain at least 3 digits, two of which must be penny values. 0.00-999999.99	`String commcard_tax_amount;`
	(**US only**) Level 2 Tax Amount of the transaction used for Corporate Credit Card transactions (Commercial Purchasing Cards).		
Vault transaction values			
Phone number	Alphanumeric	30 characters	`String phone;`
	Phone number of the customer. Can be sent in when creating or updating a Vault profile.		
Email address	Alphanumeric	30 characters	`String email;`
	Email address of the customer. Can be sent in when creating or updating a Vault profile.		

Table 116: Optional transaction values (continued)

Value	Type	Limits	Sample code variable definition
	Description		
Additional notes	Alphanumeric	30 characters	`String note;`
	This optional field can be used for supplementary information to be sent in with the transaction. This field can be sent in when creating or updating a Vault profile.		

Appendix B Definition of Response Fields

- General response fields, Appendix B Definition of Response Fields
- Recurring Billing response fields, Appendix B Definition of Response Fields
- Status Check response fields, Appendix B Definition of Response Fields
- AVS response fields, AVS response fields (see Appendix E, page 288)
- CVD response fields, CVD response fields (see Appendix F, page 294)
- MPI response fields, page 271
- Vault response fields, Vault response fields (see 9, page 108)
- Mag Swipe response fields, Mag Swipe response fields (see 10, page 163)
- Convenience Fee response fields, Convenience Fee response fields (see Appendix H, page 304)

Table 117: Receipt object response values

Value	Type	Limits	Get Method
	Description		
General response fields			
Card type	String	2-character alphabetic (min. 1)	`receipt.GetCardType());`
	Represents the type of card in the transaction, e.g., Visa, Mastercard. Possible values: V = Visa, M = Mastercard, AX = American Express , DC = Diner's Card, NO = Novus/Discover in (Canada only), DS= Discover (US only), C = JCB (US only), SE = Sears (Canada only), CQ = ACH (US only), P = Pin Debit (US only), D = Debit (canada only), C1 = JCB (Canada only)		
Card level result	String	3-alphanumeric	`receipt.getCardLevelResult();`
	TBD		
Transaction amount	String	9-character decimal	`receipt.GetTransAmount());`
	Transaction amount that was processed.		
Transaction num-ber	String	20-character alphanumeric	`receipt.GetTxnNumber());`
	Gateway Transaction identifier often needed for follow-on transactions (such as Refund and Purchase Correction) to reference the originally processed transaction.		
Receipt ID	String	50-character alphanumeric	`receipt.GetReceiptId());`
	Order ID that was specified in the transaction request.		

Table 117: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Transaction type	String	2-character alphanumeric	`receipt.GetTransType());`
	• 0 = Purchase • 1 = PreAuth • 2 = Completion • 4 = Refund • 11 = Void		
Reference number	String	18-character numeric	`receipt.GetReferenceNum());`
	Terminal used to process the transaction as well as the shift, batch and sequence number. This data is typically used to reference transactions on the host systems, and must be displayed on any receipt presented to the customer. This information is to be stored by the merchant. Example: 660123450010690030 • 66012345: Terminal ID • 001: Shift number • 069: Batch number • 003: Transaction number within the batch.		
Response code	String	3-character numeric?	`receipt.GetResponseCode());`
	• < 50: Transaction approved • ≥ 50: Transaction declined • Null: Transaction incomplete. For further details on the response codes that are returned, see the Response Codes document at https://developer.moneris.com.		
ISO	String	2-character numeric	`receipt.GetISO());`
	ISO response code		
Bank totals	Object		`receipt.getBankTotals();`
	Response data returned in a Batch Close and Open Totals request. See "Definition of Response Fields" on the previous page.		

Table 117: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Message	String	100-character alphanumeric	`receipt.GetMessage());`
	Response description returned from issuer. The message returned from the issuer is intended for merchant information only, and is **not** intended for customer receipts.		
Authorization code	String	8-character alphanumeric	`receipt.GetAuthCode());`
	Authorization code returned from the issuing institution.		
Complete		true/false	`receipt.GetComplete());`
	Transaction was sent to authorization host and a response was received		
Transaction date	String	Format: yyyy-mm-dd	`receipt.GetTransDate());`
	Processing host date stamp		
Transaction time	String	Format: ##:##:##	`receipt.GetTransTime());`
	Processing host time stamp		
Ticket	String	N/A	`receipt.GetTicket());`
	Reserved field.		
Timed out		true/false	`receipt.GetTimedOut());`
	Transaction failed due to a process timing out.		
Is Visa Debit		true/false	`receipt.GetIsVisaDebit());`
	(**Canada only**) Indicates whether the card processed is a Visa Debit.		
Batch Close/Open Totals response fields (see)			
Processed card types	String Array	N/A	`receipt.GetCreditCards(ecr));`
	Returns all of the processed card types in the current batch for the terminal ID/ECR Number from the request.		
Terminal IDs	String	8-character alpha-numeric	`receipt.getTerminalIDs();`
	Returns the terminal ID/ECR Number from the request.		

Table 117: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Purchase count	String	4-character numeric	`receipt.GetPurchaseCount(ecr, cardType);`
	Indicates the # of Purchase, ACH debit, Pre-Authorization Completion and Force Post transactions processed. If none were processed in the batch, then the value returned will be 0000.		
Purchase amount	String	11-character alpha-numeric	`receipt.GetPurchaseAmount(ecr, cardType));`
	Indicates the dollar amount processed for Purchase, ACH debit, Pre-Authorization Completion or Force Post transactions. This field begins with a + and is followed by 10 numbers, the first 8 indicate the amount and the last 2 indicate the penny value. Example, +0000000000 = 0.00 and +0000041625 = 416.25		
Refund count	String	4-character numeric	`receipt.GetRefundCount(ecr, cardType);`
	Indicates the # of Refund, Independent Refund or ACH Credit transactions processed. If none were processed in the batch, then the value returned will be 0000.		
Refund amount	String	11-character alpha-numeric	`receipt.GetRefundAmount(ecr, cardType));`
	Indicates the dollar amount processed for Refund, Independent Refund or ACH Credit transactions. This field begins with a + and is followed by 10 numbers, the first 8 indicate the amount and the last 2 indicate the penny value. Example, +0000000000 = 0.00 and +0000041625 = 416.25		
Correction count	String	4-character numeric	`receipt.GetCorrectionCount(ecr, cardType);`
	Indicates the # of Purchase Correction or ACH Reversal transactions processed. If none were processed in the batch, then the value returned will be 0000.		
Correction amount	String	11-character alpha-numeric	`receipt.GetCorrectionAmount(ecr,-cardType));`
	Indicates the dollar amount processed for Purchase Correction or ACH Reversal transactions. This field begins with a + and is followed by 10 numbers, the first 8 indicate the amount and the last 2 indicate the penny value. Example, +0000000000 = 0.00 and +0000041625 = 416.25		
Recurring Billing Response Fields (see Appendix G, page 297)			

Table 117: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Recurring billing success	String	true/false	`receipt.GetRecurSuccess();`
	Indicates whether the recurring billing transaction has been successfully set up for future billing.		
Recur update success	String	true/false	`receipt.GetRecurUpdateSuccess();`
	Indicates recur update success.		
Next recur date	String	yyyy-mm-dd	`receipt.GetNextRecurDate();`
	Indicates next recur billing date.		
Recur end date	String	yyyy-mm-dd	`receipt.GetRecurEndDate();`
	Indicates final recur billing date.		
Status Check response fields (see Appendix C, page 280)			
Status code	String	3-character alpha-numeric	`receipt.GetStatusCode();`
	• < 50: Transaction found and successful • ≥ 50: Transaction not found and not successful Note that the status code is only populated if the connection object's Status Check property is set to **true**.		
Status message	String	found or not found	`receipt.GetStatusMessage();`
	• Found: 0 ≤ Status Code ≤ 49 • Not Found or null: 50 ≤ Status Code ≤ 999. Note that The status message is only populated if the connection object's Status Check property is set to **true**.		
AVS response fields (see Appendix E, page 288)			
AVS result code	String	1-character alpha-numeric	`receipt.GetAvsResultCode();`
	Indicates the address verification result. For a full list of possible response codes refer to Section Appendix B.		
CVD response fields (see Appendix F, page 294)			

Table 117: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
CVD result code	String	2-character alpha-numeric	`receipt.GetCvdResultCode());`
	Indicates the CVD validation result. The first byte is the numeric CVD indicator sent in the request; the second byte is the response code. Possible response codes are shown in Appendix B		
MPI response fields (see "MPI" on page 60)			
Type	String	99-character alpha-numeric	
	VERes, PARes or error defines what type of response you are receiving .		
Success	Boolean	true/false	`receipt.GetMpiSuccess());`
	True if attempt was successful, false if attempt was unsuccessful.		
Message	String	100-character alphabetic	`receipt.GetMpiMessage());`
	MPI TXN transactions can produce the following values: • Y: Create VBV verification form popup window. • N: Send purchase or preauth with crypt type 6 • U: Send purchase or preauth with crypt type 7. MPI ACS transactions can produce the following values: • Y or A: (Also `receipt.getMpiSuccess()=true`) Proceed with cavv purchase or cavv preauth. • N: Authentication failed or high-risk transaction. It is recommended that you do not to proceed with the transaction. Depending on a merchant’s risk tolerance and results from other methods of fraud detection, transaction may proceed with crypt type 7. • U or time out: Send purchase or preauth as crypt type 7.		
Term URL	String	255-character alphanumeric	
	URL to which the PARes is returned		
MD	String	10024-character alphanumeric	
	Merchant-defined data that was echoed back		

Table 117: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
ACS URL	String	255-character alphanumeric	
	URL that will be for the generated pop-up		
MPI CAVV	String	28-character alpha-numeric	`receipt.GetMpiCavv());`
	Visa/MasterCard authentication data		
CAVV result code	String	1-character alpha-numeric	`receipt.GetCavvResultCode());`
	Indicates the Visa CAVV result. "Cavv Result Codes" on page 78. 0 = CAVV authentication results invalid 1 = CAVV failed validation; authentication 2 = CAVV passed validation; authentication 3 = CAVV passed validation; attempt 4 = CAVV failed validation; attempt 7 = CAVV failed validation; attempt (US issued cards only) 8 = CAVV passed validation; attempt (US issued cards only) The CAVV result code indicates the result of the CAVV validation.		
MPI inline form			`receipt.GetInLineForm());`
Vault response fields (see 9, page 108)			
Data key	String	25-character alpha-numeric	`receipt.GetDataKey());`
	This field is created when the ResAddCC transaction or ResTokenizeCC transaction is sent. (That is, when the profile is created.) It is a unique profile identifier, and is a required value for for all future Vault transactions.		
Payment type	String	cc/ach	`receipt.GetPaymentType();`
	Indicates the payment type associated with a Vault profile		
Masked PAN	String	20-character numeric	`receipt.GetResDataMaskedPan());`
	Returns the first 4 and/or last 4 of the card number saved in the profile.		

Table 117: Receipt object response values (continued)

Value	Type	Limits	Get Method / Description
Expired card count	String		
	Total number of profiles (minus 1) that have a credit card that is expiring in the current or next calendar month. This value is returned by the ResGetExpiring transaction.		
Vault success	String	true/false	`receipt.GetResSuccess());`
	Indicates whether Vault transaction was successful.		
Vault customer ID	String	30-character alphanumeric	`receipt.GetResDataCustId());`
	Returns the customer ID saved in the profile.		
Vault phone number	String	30-character alphanumeric	`receipt.GetResDataPhone());`
	Returns the phone number saved in the profile.		
Vault email address	String	30-character alphanumeric	`receipt.GetResDataEmail());`
	Returns the email address saved in the profile.		
Vault note	String	30-character alphanumeric	`receipt.GetResDataNote());`
	Returns the note saved in the profile.		
Vault expiry date	String	4-character numeric	`receipt.GetResDataExpdate());`
	Returns the expiry date of the card number saved in the profile. YYMM format.		
E-commerce indicator	String	1-character numeric	`receipt.GetResDataCryptType());`
	Returns the e-commerce indicator saved in the profile.		
Vault AVS street number	String	19-character alphanumeric	`.GetResDataAvsStreetNumber());`
	Returns the AVS street number saved in the profile. If no other AVS street number is passed in the transaction request, this value will be submitted along with the financial transaction to the issuer.		

Table 117: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Vault AVS street name	String	19-character alpha-numeric	`receipt.GetResDataAvsStreetName());`
	Returns the AVS street name saved in the profile. If no other AVS street number is passed in the transaction request, this value will be submitted along with the financial transaction to the issuer.		
Vault AVS ZIP code	String	9-character alpha-numeric	`receipt.GetResDataAvsZipcode());`
	Returns the AVS zip/postal code saved in the profile. If no other AVS street number is passed in the transaction request, this value will be submitted along with the financial transaction to the issuer.		
Vault customer first name	String	50-character alpha-numeric	`receipt.GetResDataCustFirstName());`
	(**US ACH only**) Returns the customer first name saved in the profile.		
Vault customer last name	String	50-character alpha-numeric	`receipt.GetResDataCustLastName());`
	(**US ACH only**) Returns the customer last name saved in the profile.		
Vault customer address 1	String	50-character alpha-numeric	`receipt.GetResDataCustAddress1());`
	(**US ACH only**) Returns the customer address line 1 saved in the profile.		
Vault customer address 2	String	50-character alpha-numeric	`receipt.GetResDataCustAddress2());`
	(**US ACH only**) Returns the customer address line 2 saved in the profile.		
Vault customer city	String	50-character alpha-numeric	`receipt.GetResDataCustCity());`
	US ACH only Returns the customer city saved in the profile.		
Vault customer state	String	2-character alpha-numeric	`receipt.GetResDataCustState());`
	US ACH only Returns the customer state code saved in the profile.		
Vault customer ZIP code	String	10-character numeric	`receipt.GetResDataCustZip());`
	US ACH only Returns the customer zip code saved in the profile.		

Table 117: Receipt object response values (continued)

Value	Type	Limits	Get Method / Description
Vault check routing number	String	9-character numeric	`receipt.GetResDataRoutingNum());`
	US ACH only Returns the customer check routing number saved in the profile.		
Vault masked account number	String	15-character alphanumeric	`receipt.GetResDataMaskedAccountNum());`
	US ACH only Returns the masked first 4 and last 4 digits of the account number saved in the profile.		
Vault check number	String	16-character numeric	`receipt.GetResDataCheckNum());`
	US ACH only Returns the check number saved in the profile.		
Vault account type	String	savings/checking	`receipt.GetResDataAccountType());`
	US ACH only Returns the type of account saved in the profile.		
Vault SEC code	String	3-character alphanumeric	`receipt.GetResDataSec());`
	US ACH only Returns the ACH SEC code saved in the profile.		
Vault credit card number	String		
Expiring customer ID	String		
Expiring customer's phone number	String		
Expiring customer's email address	String		
Expiring customer note	String		`receipt.getExpEmail(index)`
Expired payment type	String		`receipt.GetExpPaymentType(dataKey));`

Table 117: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Masked expiring credit card number	String		`receipt.getExpMaskedPan(index)`
Expiry date of expiring credit card	String		`receipt.GetExpExpdate(dataKey));`
E-commerce type of expiring credit card	String		`receipt.GetExpCryptType(dataKey));`
AVS street number of expiring credit card	String		`receiptreceipt.GetExpAvsStreetNumber (dataKey));`
AVS street name of expiring credit card	String		`receipt.GetExpAvsStreetName(dataKey));`
AVS ZIP code of expiring credit card	String		`receipt.GetExpAvsZipCode(dataKey));`
	TBD		
Presentation type of expiring credit card	String		`receipt.GetExpPresentationType (dataKey));`
P Account number of expiring credit card?	String		`receipt.GetExpPAccountNumber (dataKey));`
Corporate card		true/false	`receipt.GetCorporateCard());`
	Indicates whether the card associated with the Vault profile is a corporate card.		
Mag Swipe response fields (see 10, page 163)			
Masked credit card number	String		`receipt.GetResDataMaskedPan());`
Convenience Fee response fields (see Appendix H, page 304)			
Convenience fee success		true/false	
	Indicates whether the Convenience Fee transaction processed successfully.		

Table 117: Receipt object response values (continued)

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

Table 117: Receipt object response values (continued)

Value	Type	Limits	Get Method
	Description		
Other			
ITD Response	String	1-character alpha-numeric	`receipt.GetITDResponse());`

	The ITD (Internet Transaction Data) reviews several methods for performing a credit card transaction online. The ITDReponse indicates the AmEx ITD validation results. Applicable for AmEx and JCB only.
	
	Y = data matches
	N = data does not match
	U = data not checked
	R = retry
	S = Service not allowed
	[space] = data not sent		

RuleName			
	The names of rules verified from the selected policy that have triggered. Each rule name is returned as a separate name/value pair.		
RuleCode			
	The codes of the rules verified from the selected policy that have triggered. Each rule code is returned as a separate name/value pair.		
RuleMessageEn			
	An English message description of the rule returned.		
RuleMessageFr			
	A French message description of the rule returned.		
CorporateCard	Boolean string	true/ false	`receipt.GetCorporateCard());`
	Indicates whether the card associated with the vault profile is a corporate card or not.		

Table 118: Financial transaction response codes

Code	Description
< 50	Transaction approved
≥ 50	Transaction declined
NULL	Transaction was not sent for authorization

For more details on the response codes that are returned, see the Response Codes document available at https://developer.moneris.com

Table 119: Vault Admin Responses

Code	Description
001	Successfully registered CC details. Successfully updated CC details. Successfully deleted CC details. Successfully located CC details. Successfully located # expiring cards. (NOTE: # = the number of cards located)
983	Cannot find previous
986	Incomplete: timed out
987	Invalid transaction
988	Cannot find expiring cards
Null	Error: Malformed XML

Appendix C Status Check

Status Check is a connection object value that allows merchants to verify whether a previously sent transaction was processed successfully.

To submit a Status Check request, resend the original transaction with all the same parameter values, but set the status check value to either `true` or `false`.

Once set to "true", the gateway will check the status of a transaction that has an order_id that matches the one passed.

- If the transaction is found, the gateway will respond with the specifics of that transaction.
- If the transaction is not found, the gateway will respond with a not found message.

Once it is set to "false", the transaction will process as a new transaction.

For example, if you send a Purchase transaction with Status Check, include the same values as the original Purchase such as the order ID and the amount.

The feature must be enabled in your merchant profile. To have it enabled, contact Moneris.

Things to consider:

- The Status Check request should only be used once and immediately (within 2 minutes) after the last transaction that had failed.
- The Status Check request should not be used to check openTotals & batchClose requests.
- Do not resend the Status Check request if it has timed out. Additional investigation is required.

C.1 Using Status Check Response Fields

After you have used the connection object to send a Status Check request, you can use the Receipt object to obtain the information you want regarding the success of the original transaction.

The status response fields related to the status check are Status Code and Status Message.

Possible Status Code response values:

- 0-49: successful transaction
- 50-999: unsuccessful transaction.

Possible Status Message response values:

- Found: Status code is 0-49
- Not found or Null: Status code is 50-999)

If the Status Message is `Found`, all other response fields are the same as those from the original transaction.

If the Status Message is `Not found`, all other response fields will be Null.

Sample Purchase transaction with Status Check

```
public class TestCanadaPurchase
{
```

Sample Purchase transaction with Status Check

```
        public static void main(String[] args)
        {
            boolean status_check = false;
            Purchase purchase = new Purchase();

            HttpsPostRequest mpgReq = new HttpsPostRequest();
            mpgReq.setTransaction(purchase);
            mpgReq.setStatusCheck(status_check);
            mpgReq.send();
            try
            {
                Receipt receipt = mpgReq.getReceipt();
                System.out.println("StatusCode = " + receipt.getStatusCode());
                System.out.println("StatusMessage = " + receipt.getStatusMessage());
            }
            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
```

Appendix D Customer Information

An optional add-on to a number of transactions the Customer Information object. The Customer Information object offers a number of fields to be submitted as part of the financial transaction, and stored by Moneris. These details may be viewed in the future in the Merchant Resource Center.

The following transactions support the Customer Information object :

- Purchase (Basic, Interac Debit and Vault)
- Pre-Authorization (Basic and Vault)
- Re-Authorization (Basic)
- ACH Debit

The Customer Information object holds three types of information:

- Miscellaneous customer information properties (page 283)
- Billing/Shipping information (page 283)
- Item information (page 285).

Things to consider:

- If you send characters that are not included in the allowed list, these extra transaction details may not be stored.
- All fields are alphanumeric and allow the following characters: a-z A-Z 0-9 _ - : . @ $ = /
- All French accents should be encoded as HTML entities, such as &eacute.
- The data sent in Billing and Shipping Address fields will not be used for any address verification.

D.1 Using the CustInfo object

In addition to instantiating a transaction object and a connection object (as you would for a normal transaction), you must instantiate a CustInfo object.

Any transaction that supports CustInfo has a setCustInfo method. This is used to write the customer information to the transaction object before writing the transaction object to the connection object.

CustInfo object definition

```
CustInfo customer = new CustInfo();
```

Transaction object set method

```
<transaction>.setCustInfo(customer);
```

D.1.1 Miscellaneous Properties

While most of the customer information data is organized into objects, there are some values that are properties of the CustInfo object itself. They are explained in Table 120

Table 120: CustInfo object miscellaneous properties

Value	Type	Limits	Set method
Email Address	String	60-character alphanumeric	`customer.setEmail("nick@widget.com");`
Instructions	String	100-character alphanumeric	`customer.setInstructions("Rush!");`

D.1.2 Billing/Shipping information

Billing and shipping information is stored as part of the CustInfo object. They can be written to the object in one of two ways:

- Using set methods
- Using hash tables.

Whichever method you use, you will be writing the information found in Table 121 for both the billing information and the shipping information.

All values are alphanumeric strings. Their maximum lengths are given in the Limit column.

Table 121: Billing and shipping information values

Value	Limit	Hash table key
First name	30	"first_name"
Last name	30	"last_name"
Company name	50	"company_name"
Address	70	"address"
City	30	"city"
Province/State	30	"province"
Postal/Zip code	30	"postal_code"
Country	30	"country"
Phone number (voice)	30	"phone"
Fax number	30	"fax"
Federal tax	10	"tax1"

Table 121: Billing and shipping information values (continued)

Value	Limit	Hash table key
Provincial/State tax	10	"tax2"
County/Local/Specialty tax	10	"tax3"
Shipping cost	10	"shipping_cost"

D.1.2.1 Set Methods

The billing information and the shipping information for a given CustInfo object are written by using the `customer.setBilling()` and `customer.setShipping()` methods respectively:

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

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

Both of these methods have the same set of mandatory arguments. They are explained in Table 121 (page 283) .

For sample code, see D.2 (page 285).

D.1.2.2 Hash Tables

Writing billing or shipping information using hash tables is done as follows:

1. Instantiate a CustInfo object.
2. Instantiate a Hashtable object. (The sample code uses a different hash table for billing and shipping for clarity purposes. However, the skillful developer can re-use the same one.)
3. Build the hashtable using put methods with the hash table keys in Table 121 (page 283).
4. Call the CustInfo object's setBilling/setShipping method to pass the hashtable information to the CustInfo object
5. Call the transaction object's setCustInfo method to write the CustInfo object (with the billing/-shipping information to the transaction object.

For sample code, see D.2 (page 285).

D.1.3 Item Information

The CustInfo object can hold information about multiple items. For each item, the values in Table 122 can be written.

All values are strings, but note the guidelines in the Limits column.

Table 122: Item information values

Value	Limits	Hash table key
Item name	45-character alphanumeric	"name"
Item quantity	5-character numeric	"quantity"
Item product code	20-character alphanumeric	"product_code"
Item extended amount	9-character decimal with at least 3 digits and 2 penny values. 0.01-999999.99	"extended_ amount"

One way of representing multiple items is with four arrays. This is the method used in the sample code. However, there are two ways to write the item information to the CustInfo object:

- Set methods
- Hash tables.

D.1.3.1 Set Methods

All the item information in Table 122 is written to the CustInfo in one instruction for a given item. Such as:

```
customer.setItem(item_description, item_quantity, item_product_code, item_
extended_amount);
```

For sample code (showing how to use arrays to write information about two items), see D.2 (page 285).

D.1.3.2 Hash Tables

Writing item information using hash tables is done as follows:

1. Instantiate a CustInfo object.
2. Instantiate a Hashtable object. (The sample code uses a different hash table for each item for clarity purposes. However, the skillful developer can re-use the same one.)
3. Build the hashtable using put methods with the hash table keys in Table 121 (page 283).
4. Call the CustInfo object's setItem method to pass the hashtable information to the CustInfo object
5. Call the transaction object's setCustInfo method to write the CustInfo object (with the item information to the transaction object.

For sample code (showing how to use arrays to write information about two items), see D.2 (page 285).

D.2 Customer Information Sample Code

Below are 2 examples of a Basic Purchase Transaction with Customer Information. Both samples start by declaring the same variables. Therefore, that part will only be shown once. Values that are not involved in the Customer Information feature are not shown.

Note that the two items ordered are represented by four arrays, and the billing and shipping details are the same.

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

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

Sample Purchase with Customer Information—Set method version

```
CustInfo customer = new CustInfo();

/*************** Miscellaneous Customer Information Methods *******************/
customer.setEmail("nick@widget.com");
customer.setInstructions("Make it fast!");

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

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

/**************************** Order Line Items *****************************/
customer.setItem(item_description[0], item_quantity[0], item_product_code[0], item_extended_amount
    [0]);
customer.setItem(item_description[1], item_quantity[1], item_product_code[1], item_extended_amount
    [1]);

Purchase purchase = new Purchase();
purchase.setCustInfo(customer);

HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setTransaction(purchase);
mpgReq.send();
```

Sample Purchase with Customer Information—Hash table version

```
CustInfo customer2 = new CustInfo();
/*************** Miscellaneous Customer Information Methods *******************/
customer.setEmail("nick@widget.com");
customer.setInstructions("Make it fast!");
```

Sample Purchase with Customer Information—Hash table version

```
/******************************* Billing Hashtable **************************/
Hashtable<String, String> b = new Hashtable<String, String>(); //billing hashtable
b.put("first_name", first_name);
b.put("last_name", last_name);
b.put("company_name", company_name);
b.put("address", address);
b.put("city", city);
b.put("province", province);
b.put("postal_code", postal_code);
b.put("country", country);
b.put("phone", phone);
b.put("fax", fax);
b.put("tax1", tax1); //federal tax
b.put("tax2", tax2); //prov tax
b.put("tax3", tax3); //luxury tax
b.put("shipping_cost", shipping_cost); //shipping cost
customer2.setBilling(b);
/****************************** Shipping Hashtable **************************/
Hashtable<String, String> s = new Hashtable<String, String>(); //shipping hashtable
s.put("first_name", first_name);
s.put("last_name", last_name);
s.put("company_name", company_name);
s.put("address", address);
s.put("city", city);
s.put("province", province);
s.put("postal_code", postal_code);
s.put("country", country);
s.put("phone", phone);
s.put("fax", fax);
s.put("tax1", tax1); //federal tax
s.put("tax2", tax2); //prov tax
s.put("tax3", tax3); //luxury tax
s.put("shipping_cost", shipping_cost); //shipping cost
customer2.setShipping(s);
/************************* Order Line Item1 Hashtable ***********************/
Hashtable<String, String> i1 = new Hashtable<String, String>(); //item hashtable #1
i1.put("name", item_description[0]);
i1.put("quantity", item_quantity[0]);
i1.put("product_code", item_product_code[0]);
i1.put("extended_amount", item_extended_amount[0]);
customer2.setItem(i1);
/************************* Order Line Item2 Hashtable *************************/
Hashtable<String, String> i2 = new Hashtable<String, String>(); //item hashtable #2
i2.put("name", "item2's name");
i2.put("quantity", "7");
i2.put("product_code", "item2's product code");
i2.put("extended_amount", "5.01");
customer2.setItem(i2);

Purchase purchase = new Purchase();
purchase.setCustInfo(customer);
HttpsPostRequest mpgReq = new HttpsPostRequest();
mpgReq.setTransaction(purchase);
mpgReq.send();
```

Appendix E Address Verification Service

Address Verification Service (AVS) is an optional fraud-prevention tool offered by issuing banks whereby a cardholder's address is submitted as part of the transaction authorization. The AVS address is then compared to the address kept on file at the issuing bank. AVS checks whether the street number, street name and zip/postal code match. The issuing bank returns an AVS result code indicating whether the data was matched successfully. Regardless of the AVS result code returned, the credit card is authorized by the issuing bank.

The response that is received from AVS verification is intended to provide added security and fraud prevention, but the response itself does not affect the completion of a transaction. Upon receiving a response, the choice to proceed with a transaction is left entirely to the merchant. The responses is **not** a strict guideline of whether a transaction will be approved or declined.

The following transactions support AVS:

- Purchase (Basic and Mag Swipe)
- Pre-Authorization (Basic)
- Re-Authorization (Basic)
- ResAddCC (Vault)
- ResUpdateCC (Vault)

Things to consider:

- AVS is only supported by Visa, MasterCard, Discover and American Express.
- When testing AVS, you must **only** use the Visa test card numbers 4242424242424242 or 4005554444444403, and the amounts described in the Simulator eFraud Response Codes document available at the Moneris developer portal (https://developer.moneris.com).
- Store ID “store5” is set up to support AVS testing.

E.1 Using AVS

In addition to instantiating a transaction object and a connection object (as you would for a normal transaction), you must instantiate an AvsInfo object. This object has a number of mandatory values that must be set (Table 123, page 289) and optional values that may be set (Table 124, page 289).

Any transaction that supports AVS has a setAvsInfo method. This is used to write the AVS information to the transaction object before writing the transaction object to the connection object.

AVSInfo object definition

```
AvsInfo avsCheck = new AvsInfo();
```

Transaction object set method

```
<transaction>.setAvsInfo(avsCheck);
```

E.2 AVS Request Fields

Table 123: AvsInfo object mandatory values

Value	Type	Limits	Set method
	Description		
AVS street number	String	19-character alphanumeric[1]	`avsCheck.setAvsStreetNumber("212");`
	Cardholder street number.		
AVS street name	String	See AVS street number	`avsCheck.setAvsStreetName("Payton Street");`
	Cardholder street name.		
AVS zip/ postal code	String	9-character alphanumeric	`avsCheck.setAvsZipCode("M1M1M1");`
	Cardholder zip/postal code.		

Table 124: AvsInfo object optional values

Value	Type	Limits	Set method
	Description		
AVS email address	String	60-character alphanumeric	`avsCheck.setAvsEmail ("test@host.com");`
	Email address provided by the customer at the point of sale. Applicable for American Express and JCB only.		
AVS host name	String	60-character alphanumeric	`avsCheck.setAvsHostname("host-name");`
	Applicable for American Express and JCB only.		
AVS browser type	String	60-character alphabetic	`avsCheck.setAvsBrowser("Moz-illa");`
	Web browser used to make the purchase. Applicable for American Express and JCB only.		
AVS ship-to-country code	String	3-character alphabetic	`avsCheck.setAvsShiptoCountry ("CAN");`
	Applicable for AmEx and JCB only.		

[1] 19 characters is the combined limit between AVS street number and AVS street name.

Table 124: AvsInfo object optional values (continued)

Value	Type	Limits	Set method
	Description		
AVS Shipping Method	String	X-character alphanumeric	`avsCheck.setAvsShipMethod ("G");`
Merchant product SKU	String	15-character alphanumeric	`avsCheck.setAvsMerchProdSku ("123456");`
	For multiple items, the SKU of the most expensive item should be entered. Applicable for AmEx and JCB only.		
AVS customer's IP address	String	15-character alphanumeric	`avsCheck.setAvsCustIp ("192.168.0.1");`
	IP address of device from which transaction is being sent. Applicable for AmEx and JCB only.		
AVS customer's phone number	String	10-character numeric	`avsCheck.setAvsCustPhone ("5556667777");`
	Telephone number provided at point of sale. Applicable for American Express and JCB only.		

E.3 AVS Result Codes

Below is a full list of possible AVS response codes. These can be returned when you call the `receipt.-getAvsResultCode()` method .

Table 125: AVS result codes

Value	Visa	MasterCard/Discover	Amex/JCB
A	Street address matches, zip/postal code does not. Acquirer rights not implied.	Address matches, zip/-postal code does not.	Billing address matches, zip/postal code does not.
B	Street address matches. Zip/Postal code not verified due to incompatible formats. (Acquirer sent both street address and zip/-postal code.)	N/A	N/A
C	Street address not verified due to incom-patible formats. (Acquirer sent both street address and zip/postal code.)	N/A	N/A

Table 125: AVS result codes (continued)

Value	Visa	MasterCard/Discover	Amex/JCB
D	Street address and zip/postal code match.	N/A	Customer name incorrect, zip/postal code matches
E	N/A	N/A	Customer name incorrect, billing address and zip/postal code match
F	(Applies to UK only) Street address and zip/postal code match.	N/A	Customer name incorrect, billing address matches.
G	Address information not verified for international transaction. Any of the following may be true: • Issuer is not an AVS participant. • AVS data was present in the request, but issuer did not return an AVS result. • Visa performs AVS on behalf of the issuer and there was no address record on file for this account.	N/A	N/A
I	Address information not verified.	N/A	N/A
K	N/A	N/A	Customer name matches.
L	N/A	N/A	Customer name and postal code match.
N/A	N/A	Customer name and zip/postal code match.	
M	Street address and zip/postal code match.	N/A	Customer name, billing address, and zip/postal code match.
N	No match. Also used when acquirer requests AVS but sends no AVS data.	Neither address nor postal code matches.	Billing address and postal code do not match.
O	N/A	N/A	Customer name and billing address match

Table 125: AVS result codes (continued)

Value	Visa	MasterCard/Discover	Amex/JCB
P	Postal code matches. Acquirer sent both postal code and street address, but street address not verified due to incompatible formats.	N/A	N/A
R	Retry: System unavailable or timed out. Issuer ordinarily performs AVS, but was unavailable. The code R is used by Visa when issuers are unavailable. Issuers should refrain from using this code.	Retry. System unable to process.	Retry. System unavail-able.
S	N/A	AVS currently not sup-ported.	AVS currently not sup-ported.
T	N/A	Nine-digit zip/postal code matches, address does not match.	N/A
U	Address not verified for domestic trans-action. One of the following is true: • Issuer is not an AVS participant • AVS data was present in the request, but issuer did not return an AVS res-ult • Visa performs AVS on behalf of the issuer and there was no address record on file for this account.	No data from Issuer/Authorization system.	Information is unavail-able.
W	Not applicable. If present, replaced with ‘Z’ by Visa. Available for U.S. issuers only.	For US Addresses, nine-digit zip/postal code matches, address does not. For addresses out-side the US, zip/postal code matches, address does not.	Customer name, billing address, and zip/postal code are all correct.
X	N/A	For US addresses, nine-digit zip/postal code and address match. For addresses outside the US,zip/postal code and address match.	N/A
Y	Street address and zip/postal code match.	For US addresses, five-digit zip/postal code and address match.	Billing address and zip/-postal code match.

Table 125: AVS result codes (continued)

Value	Visa	MasterCard/Discover	Amex/JCB
Z	Zip/postal code matches, but street address either does not match or street address was not included in request.	For U.S. addresses, five-digit zip code matches, address does not match.	Postal code matches, billing address does not match.

E.4 AVS Sample Code

This is a sample of .NET code illustrating how AVS is implemented with a Purchase transaction. Purchase object information that is not relevant to AVS has been removed.

Sample Purchase with AVS information

```
AvsInfo avsCheck = new AvsInfo();
avsCheck.setAvsStreetNumber("212");
avsCheck.setAvsStreetName("Payton Street");
avsCheck.setAvsZipCode("M1M1M1");
avsCheck.setAvsEmail("test@host.com");
avsCheck.setAvsHostname("hostname");
avsCheck.setAvsBrowser("Mozilla");
avsCheck.setAvsShiptoCountry("CAN");
avsCheck.setAvsShipMethod("G");
avsCheck.setAvsMerchProdSku("123456");
avsCheck.setAvsCustIp("192.168.0.1");
avsCheck.setAvsCustPhone("5556667777");

Purchase purchase = new Purchase();
purchase.setAvsInfo(avsCheck);
```

Appendix F Card Validation Digits

The Card Validation Digits (CVD) value refers to the numbers appearing on the back of the credit card rather than the numbers imprinted on the front[1]. It is an optional fraud prevention tool that enables merchants to verify data provided by the cardholder at transaction time. This data is submitted along with the transaction to the issuing bank, which provides a response indicating whether the data is a match.

The response that is received from CVD verification is intended to provide added security and fraud prevention, but the response itself does not affect the completion of a transaction. Upon receiving a response, the choice whether to proceed with a transaction is left entirely to the merchant. The responses is **not** a strict guideline of which transaction will approve or decline.

The following transactions support CVD:

- Purchase (Basic, Vault and Mag Swipe)
- Pre-Authorization (Basic and Vault)
- Re-Authorization

Things to consider:

- CVD is only supported by Visa, MasterCard and American Express.
- When testing CVD, you must **only** use the Visa test card numbers 4242424242424242 or 4005554444444403, and the amounts described in the Simulator eFraud Response Codes document available at the Moneris developer portal (https://developer.moneris.com).
- Test store_id “store5” is set up to support CVD testing.
- To have CVD for American Express added to your profile, contact American Express directly.

F.1 Using CVD

Security

The CVD value must only be passed to the payment gateway. Under **no** circumstances may it be stored for subsequent uses or displayed as part of the receipt information.

In addition to instantiating a transaction object and a connection object (as you would for a normal transaction), you must instantiate an CVDInfo object. This object has a number of mandatory values that must be set (Table 126, page 295) .

Any transaction that supports CVD has a setCvdInfo method. This is used to write the CVD information to the transaction object before writing the transaction object to the connection object.

CvdInfo object definition

```
CvdInfo cvdCheck = new CvdInfo();
```

[1]The exception to this rule is with American Express cards, which have the CVD printed on the front.

Transaction object set method

```
transaction.setCvdInfo(cvdCheck);
```

F.2 CVD Request Fields

Security

The CVD value must only be passed to the payment gateway. Under **no** circumstances may it be stored for subsequent uses or displayed as part of the receipt information.

Table 126: CvdInfo object mandatory values

Value	Type	Limits	Set method
	Description		
CVD indicator	String	1-character numeric	`cvdCheck.setCvdIndicator("1");`
	CVD presence indicator: 0: CVD value is deliberately bypassed or is not provided by the merchant. 1: CVD value is present. 2: CVD value is on the card, but is illegible. 9: Cardholder states that the card has no CVD imprint.		
CVD value	String	4-character numeric	`cvdCheck.setCvdValue("099");`
	CVD value located on credit card. The CVD value (supplied by the cardholder) must only be passed to the payment gateway. Under **no** circumstances may it be stored for subsequent use or displayed as part of the receipt information.		

F.3 CVD Result Definitions

Table 127: CVD result definitions

Value	Definition
M	Match
N	No Match
P	Not Processed
S	CVD should be on the card, but Merchant has indicated that CVD is not present.
U	Issuer is not a CVD participant

Value	Definition
Y	Match for AmEx/JCB only
D	Invalid security code for AmEx/JCB
Other	Invalid response code

F.4 CVD Sample Code

This is a sample of .NET code illustrating how CVD is implemented with a Purchase transaction. Purchase object information that is not relevant to CVD has been removed.

Sample purchase with CVD information

```
CvdInfo cvdCheck = new CvdInfo();
cvdCheck.setCvdIndicator("1");
cvdCheck.setCvdValue("099");

Purchase purchase = new Purchase();
purchase.setCvdInfo(cvdCheck);
```

Appendix G Recurring Billing

- G.1 Setting up a new recurring payment
- G.2 Updating a Recurring Payment
- Appendix A Recurring Billing Response Fields and Codes, page 1

Recurring Billing allows you to set up payments whereby Moneris automatically processes the transactions and bills customers on your behalf based on the billing cycle information you provide.

Section 1.1 outlines how to set up a new recurring payment when you submit a Purchase transaction (for various features), and Section 1.2 outlines how to update the details of a previously registered recurring payment by using the Recur Update transaction.

In addition to Recur Update, the features that support Purchase transactions with recurring billing are:

- Basic
- ACH (referred to as ACH Debit)
- Vault

Things to consider:

- To avoid shifting, do not set the `start_date` after the 28th if the `recur_unit` is `month`. To set the billing date for the last day of the month, set `recur_unit` to `eom`.
- When completing the update recurring billing portion please keep in mind that the recur bill dates cannot be changed to have an end date greater than 10 years from today and cannot be changed to have an end date end today or earlier.

G.1 Setting up a new recurring payment

In addition to instantiating a transaction object and a connection object (as you would for a normal transaction), you must instantiate a Recur object. This object has a number of mandatory properties that must be set (Table 128, page 298).

Any transaction that supports Recurring Billing has a setRecur method. This is used to write the Recurring Billing information to the transaction object before writing the transaction object to the connection object.

Recur Object Definition

```
Recur recurring_cycle = new Recur(recur_unit, start_now, start_date, num_
recurs, period, recur_amount);
```

For an explanation of these fields, see Table 128 (page 298).

Transaction object set method

```
<transaction>.setRecur(recurring_cycle);
```

For Recurring Billing response fields, see page 1.

Table 128: Recur object mandatory arguments

Value	Type	Limits	Argument name in example
	Description		
Recur unit	String	`day`, `week`, `month` or `eom`	`recur_unit`
	Unit to be used as a basis for the interval. This can be set as day, week, month or the end of the month. Works in conjunction with the period argument (see below) to define the billing frequency.		
Start Now	String	`true/false`	`start_now`
	If a single charge is to be made against the card immediately, set this value to `true`. The amount to be billed immediately may differ from the amount billed on a regular basis thereafter. If the billing is to start in the future, set this value to `false`.		
Start Date	String	YYYY/MM/DD format	`start_date`
	Date of the first future recurring billing transaction. This value **must** be a date in the future. If an additional charge is to be made immediately, the `start_now` argument must be set to `true`.		
Number of Recurs	String	numeric 1-99	`num_recurs`
	The number of times that the transaction must recur.		
Period	String	numeric 1-999	`period`
	Number of recur units that must pass between recurring billings.		
Recurring Amount	String	9-character decimal 0.01-9999999.99.	`recur_amount`
	Amount of the recurring transaction. This must contain at least three digits, two of which are penny values. This is the amount that will be billed on the `start_date`, and then billed repeatedly based on the interval defined by `period` and `recur_unit`.		

Recurring billing examples

```
Recur recurring_cycle = new Recur(recur_unit, start_now, start_date, num_
recurs, period, recur_amount);
```

Given a Recur object with the above syntax, Table 129 shows how the transaction is interpreted for different argument values.

Table 129: Recurring Billing examples

Argument	Values	Description
recur_unit	"month";	The first transaction occurs on January 2, 2030 (because start_now="false"). The card is billed $30.00 every 2 months on the 2nd of each month. The card will be billed a total of 12 times. This includes the transaction on January 2, 2030
start_date	"2030/01/02"	
num_recurs	"12"	
start_now	"false"	
period	"2"	
recur_amount	"30.00"	
recur_unit	"week";	The first charge is billed immediately (because start_now=true). The initial charge is $15.00. Beginning on January 2, 2030 the credit card will be billed $30.00 every 2 weeks for 26 recurring charges. Therefore, the card will be billed a total of 27 times. (1 immediate and 26 recurring.)
start_date	"2030/01/02"	
num_recurs	"26"	
start_now	"true"	
period	"2"	
recur_amount	"30.00"	

Sample Purchase with Recurring Billing

```
public class TestPurchaseRecur
{
    public static void main(String[] args)
    {
        /**Purchase transaction arguments removed for space

        /************************* Recur Variables **********************************/
        String recur_unit = "month"; //eom = end of month
        String start_now = "true";
        String start_date = "2016/07/28";
        String num_recurs = "12";
        String period = "1";
        String recur_amount = "30.00";
        /************************* Recur Object Option1 ******************************/
        Recur recurring_cycle = new Recur(recur_unit, start_now, start_date, num_recurs, period,
            recur_amount);
        /************************* Recur Object Option2 ******************************/
        Hashtable<String, String> recur_hash = new Hashtable<String, String>();
        recur_hash.put("recur_unit", recur_unit);
        recur_hash.put("start_now", start_now);
        recur_hash.put("start_date", start_date);
        recur_hash.put("num_recurs", num_recurs);
        recur_hash.put("period", period);
        recur_hash.put("recur_amount", recur_amount);
        /************************ Transactional Object *******************************/
        Purchase purchase = new Purchase();
        /**Purchase transaction arguments removed for space
        /******************************* Set Recur ***********************************/
```

Sample Purchase with Recurring Billing

```
          purchase.setRecur(recurring_cycle);
          /*************************** Https Post Request ***************************/
          HttpsPostRequest mpgReq = new HttpsPostRequest();
          /**Connection object arguments removed for space

          mpgReq.send();
          catch (Exception e)
       }
   }
```

G.2 Updating a Recurring Payment

After you have set up a Recurring Billing transaction, you can change the details of it. The RecurUpdate transaction object works like any of the basic transactions. That is, you must instantiate the RecurUpdate object, instantiate a connection object, update the connection object with the Recur Update transaction object, invoke the connection object's send method.

RecurUpdate transaction object definition

```
RecurUpdate recurUpdate = new RecurUpdate();
```

HttpsPostRequest object for recurring billing update transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.setTransaction(recurUpdate);
```

Table 130: RecurUpdate transaction object mandatory values

Value	Type	Limits	Set method
	Description		
Order ID	String	50-character alphanumeric	`recurUpdate.setOrderId(order_id);`
	Order ID of the previously registered recurring billing transaction.		

With the exception of Status Check, the values/actions in Table 131 are optional because they are the values that were specified in the original Recurring Billing transaction that you may now update. You can update any or all of them.

Status Check is used to determine whether a previous Recur Update transaction was properly processed.

Table 131: RecurUpdate transaction optional values

Value/Action	Type	Limits	Set method
	Description (if any)		
Non-recurring billing values (see "Definition of Request Fields" on page 258 for more details).			

Table 131: RecurUpdate transaction optional values (continued)

Value/Action	Type	Limits	Set method
	Description (if any)		
Customer ID	String	50-character alphanumeric	`recurUpdate.setCustId(cust_id);`
Credit card number	String	20-character alphanumeric	`recurUpdate.setPan(pan);`
Credit card expiry date	String	4-character alphanumeric (YYMM format)	`recurUpdate.setExpdate(expiry_date);`
Status Check[1]	Boolean	`true/false`	`mpgReq.setStatusCheck(status_check);`
Recurring billing values			
Recurring amount	String	9-character decimal At least 3 digits with two penny values. (0.01-9999999.99).	`recurUpdate.setRecurAmount(recur_amount);`
	Changes the amount that is billed recurrently. The change takes effect on the next charge.		
Add number of recurs	String	Numeric 1-999	`recurUpdate.setAddNumRecurs(add_num);`
	Adds to the given number of recurring transactions to the current (remaining) number. This can be used if a customer decides to extend a membership/subscription. However, because this must be a positive number, it cannot be used to decrease the current number of recurring transactions. For that, use the setTotalNumRecurs method below.		
Change number of recurs	String	Numeric 1-999	`recurUpdate.setTotalNumRecurs(total_num);`
	Replaces the current (remaining) number of recurring transactions. Note how this differs from the setAddNumRecurs method above.		

[1]For more information, see Appendix C (page 280).

Table 131: RecurUpdate transaction optional values (continued)

Value/Action	Type	Limits	Set method
	Description (if any)		
Hold recurring billing	String	TBD	`recurUpdate.setHold(hold);`
	Temporarily pauses recurring billing. While a transaction is on hold, it is not billed for the recurring amount. However, the number of remaining recurs continues to be decremented during that time.		
Terminate recurring transaction	String	TBD	`recurUpdate.setTerminate(terminate);`
	Terminates recurring billing. Note: After it has been terminated, a recurring transaction **cannot** be reactivated. A new purchase transaction with recurring billing must be submitted.		

Sample Purchase with Recurring Billing

```
public class TestCanadaRecurUpdate
{
    public static void main(String[] args)
    {
        String store_id = "store5";
        String api_token = "yesguy";
        String order_id = "Test155409282";
        String cust_id = "antonio";
        String recur_amount = "1.50";
        String pan = "4242424242424242";
        String expiry_date = "1902";
        //String add_num = "";
        //String total_num = "";
        //String hold = "";
        //String terminate = "";
        String processing_country_code = "CA";
        boolean status_check = false;

        RecurUpdate recurUpdate = new RecurUpdate();
        recurUpdate.setOrderId(order_id);
        recurUpdate.setCustId(cust_id);
        recurUpdate.setRecurAmount(recur_amount);
        recurUpdate.setPan(pan);
        recurUpdate.setExpdate(expiry_date);
        //recurUpdate.setAddNumRecurs(add_num);
        //recurUpdate.setTotalNumRecurs(total_num);
        //recurUpdate.setHold(hold);
        //recurUpdate.setTerminate(terminate);

        HttpsPostRequest mpgReq = new HttpsPostRequest();
        mpgReq.setProcCountryCode(processing_country_code);
        mpgReq.setTestMode(true); //false or comment out this line for production transactions
        mpgReq.setStoreId(store_id);
        mpgReq.setApiToken(api_token);
        mpgReq.setTransaction(recurUpdate);
```

Sample Purchase with Recurring Billing

```
            mpgReq.setStatusCheck(status_check);
            mpgReq.send();

            catch (Exception e)
            {
                e.printStackTrace();
            }
        }
    }
```

Appendix H Convenience Fee

The Convenience Fee program allows merchants to apply an additional charge to a customer's bill (with their consent) for the convenience of being able to pay for goods and services using an alternative payment channel. This applies only when providing a true convenience in the form of a channel outside the merchant's customary face-to-face payment channels.

The convenience fee is a charge in addition to what the consumer is paying for the provided goods/services. This charge appears as a separate line item on the consumer's statement.

The Convenience Fee program provides several benefits. It may allow you an opportunity to reduce or eliminate credit card processing fees and improve customer satisfaction.

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

It is supported by the following transactions:

- Basic Purchase
- CAVV Purchase
- ACH Debit.

H.1 Using Convenience Fee

In addition to instantiating a transaction object and a connection object (as you would for a normal transaction), you must instantiate a ConvFeeInfo object. This object has one mandatory value that must be set (Table 132, page 305).

Any transaction that supports Convenience Fee has a setConvFeeInfo method. This is used to write the Convenience Fee information to the transaction object before writing the transaction object to the connection object.

ConvFeeInfo object definition

```
ConvFeeInfo convFeeInfo = new ConvFeeInfo();
```

Transaction object set method

```
<transaction>.setConvFeeInfo(convFeeInfo);
```

H.2 Convenience Fee Request Fields

Table 132: ConvFeeInfo object mandatory values

Value	Type	Limits	Set method
	Description		
Convenience fee amount	Decimal	9 characters	`convFeeInfo.setConvenienceFee ("5.00");`
	Amount customer is being charged as a convenience fee.		

H.3 Convenience Fee Sample Code

This is a sample of .NET code illustrating how the Convenience Fee option is implemented with a Purchase transaction. Purchase object information that is not relevant to Convenience Fee has been removed.

Sample Purchase with Convenience Fee information

```
Purchase purchase = new Purchase();

ConvFeeInfo convFeeInfo = new ConvFeeInfo();
convFeeInfo.setConvenienceFee("5.00");
purchase.setConvFeeInfo(convFeeInfo);
```

Appendix I Error Messages

Error messages that are returned if the gateway is unreachable

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. The majority of the time, the explanation is contained within the Message field.

When a 'NULL' response is returned, it can indicate that the issuer, the credit card host, or the gateway is unavailable. This may be because they are offline or because you are unable to connect to the internet.

A 'NULL' can also be returned when a transaction message is improperly formatted.

Error messages that are returned in the Message field of the response

XML Parse Error in Request: <System specific detail>
An improper XML document was sent from the API to the servlet.

XML Parse Error in Response: <System specific detail>
An improper XML document was sent back from the servlet.

Transaction Not Completed Timed Out
Transaction timed out before the host responds to the gateway.

Request was not allowed at this time
The host is disconnected.

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

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

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

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

Vault error messages

Can not find previous
Data key provided was not found in our records or profile is no longer active.

Invalid Transaction
Transaction cannot be performed because improper data was sent.

or

Mandatory field is missing or an invalid SEC code was sent.

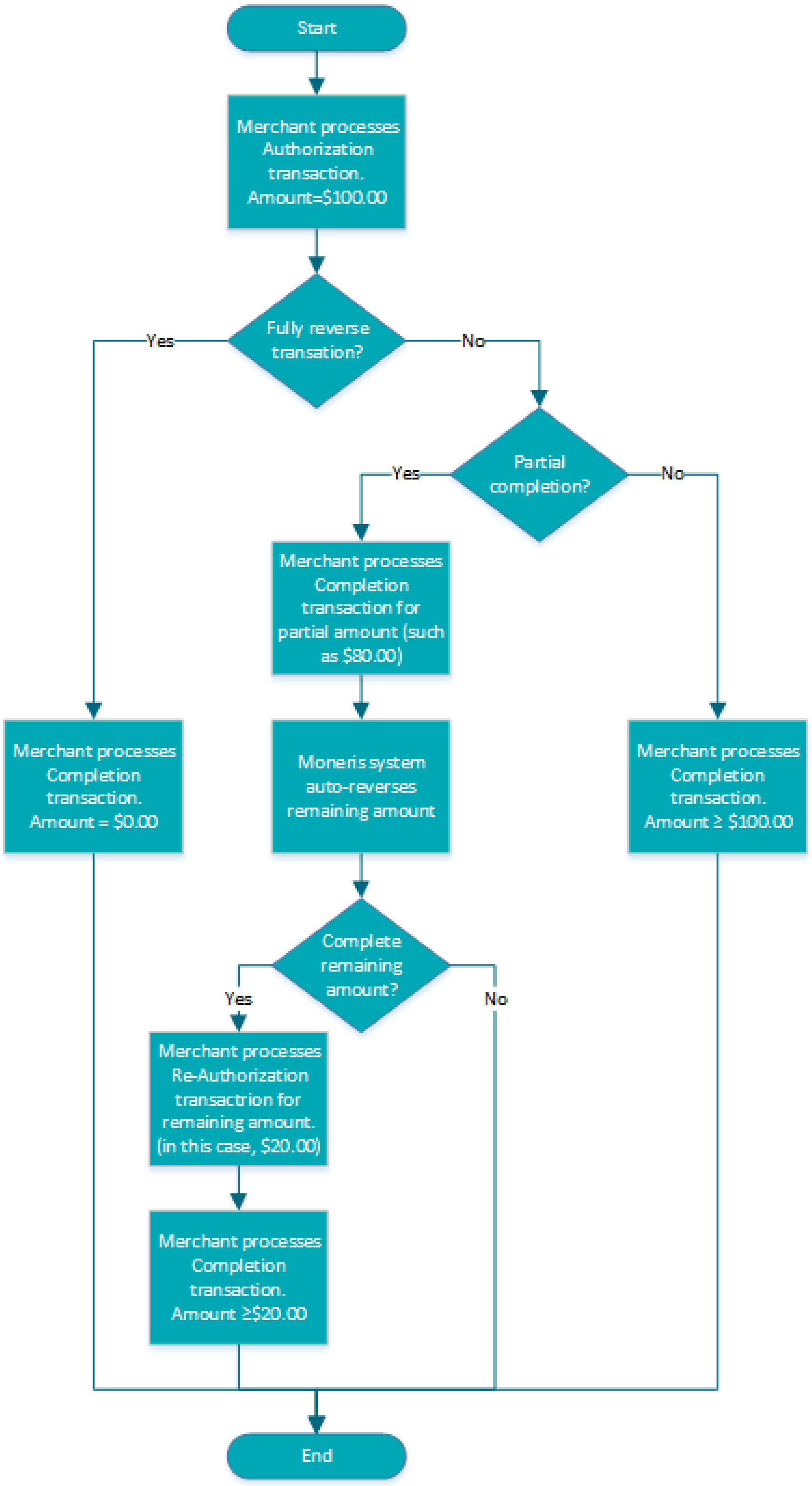
Malformed XML
Parse error.

Incomplete
Timed out.

or

Cannot find expiring cards.

Appendix J Process Flow for Basic PreAuth, ReAuth and Completion Transactions


Start
Merchant processes Authorization transaction. Amount=$100.00
Fully reverse transation?
Yes
No
Partial completion?
Yes
No
Merchant processes Completion transaction for partial amount (such as $80.00)
Merchant processes Completion transaction. Amount = $0.00
Moneris system auto-reverses remaining amount
Merchant processes Completion transaction. Amount ≥ $100.00
Complete remaining amount?
Yes
No
Merchant processes Re-Authorization transactrion for remaining amount. (in this case, $20.00)
Merchant processes Completion transaction. Amount ≥$20.00
End

Appendix K Merchant Checklists for INTERAC® Online Payment Certification Testing

Merchant Information

Name and URL	Merchant Name (English)	
	Homepage URL (English)	
	Merchant Name (French)	
	Homepage URL (French)	
Number	Merchant Number	
Transaction fee category (Circle one)	Government Education General	

Checklist for Front-End Tests

Case #	Date Completed	Remarks
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		

Case #	Date Completed	Remarks
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		
30		
31		
32		
33		
34		
35		
36		
37		
38		
39		

Merchant Requirements

Table 133: Checklist for web display requirements

Done	Requirement
Checkout page	
	Displays the INTERAC Online design (logo), wordmark (text "INTERAC Online) or both

Table 133: Checklist for web display requirements (continued)

Done	Requirement
Design and Wordmark Requirements (any page)	
	Other payment option logos: • Displays the INTERAC Online design (logo) if the merchant displays the trademarks or logos of other payment options. • Design is equal in size and no less prominent than other payment option trademarks.
	INTERAC wordmark: • INTERAC is always either in capital letters or italics (as in "the INTERAC Online service") • In the first use of the INTERAC Online wordmark, INTERAC is followed by the ® notation in superscript. For example, "*Interac*®" (English) or <<*Interac*MD>> (French). • On the same page as the first occurence of the wordmark, the following language-appropriate footnote appears: • ® Trademark of Interac Inc. Used under licence" • MD Marque de commerce d'Interac Inc. Utilisée sous licence
Version of design	
	Uses the two-colour design on the web: • Horizontal version—height no shorter than 25 pixels (width-to-height ratio of 2:37:1) • Vertical version—width no narrower than 30 pixels (widteh-to-height ratio of 1:1:37)
"Learn more" information	
	Provides consumers with a link to www.interaconline.com/learn (preferably on the checkout page)
Confirmation page	
	States that the transaction is successful
	Displays the financial institution's name and confirmation number
	Provides ability to print
Error page	
	Indicates that payment was unsuccsessful
	States that the order is cancelled or displays other payment options
Timeout message	
	Is displayed if consumer has less than 30 minutes to complete payment
Payment	
	Displays the total in Canadian dollars

Table 134: Checklist for security/privacy requirements

Done	Requirement
Merchant	
	Uses no less than 128-bit SSL encryption when collecting personal information
	Protects consumer information in accordance with applicable federal and provincial privacy legislation
	Adheres to the Canadian Code of Practice for Consumer Protection in Electronic Commerce
Provided screenshots	
	Checkout page (where customer selects INTERAC Online option)
	Confirmation page (one of the test case 1, 2, or 3)
	Error page (test case 4)

Appendix L Third-Party Service Provider Checklists for INTERAC® Online Payment Certification Testing

Third-Party Service Provider Information

Name	English	
	French	
Merchant Web Application	Solution Name	
	Version	
Acquirer		

Interaconline.com/Interacenlgne.com Web Site Listing Information

See http://www.interaconline.com/merchants_thirdparty.php for examples.

English contact inform-ation	5 lines maximum. 35 characters/line maximum. For example, contact name and title, department, telephone, web site, email.
English logo	File type: PNG. Maximum size: 120x120 pixels.
French contact inform-ation	5 lines maximum. 35 characters/line maximum. For example, contact name and title, department, telephone, web site, email.
French logo	File type: PNG. Maximum size: 120x120 pixels.

Table 135: Checklist for front-end tests

Case #	Date Completed	Remarks
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		

Table 135: Checklist for front-end tests

Case #	Date Completed	Remarks
30		
31		
32		
33		
34		
35		
36		
37		
38		
39		

Merchant Requirements

Table 136: Checklist for web display requirements

Done	Requirement
Checkout page	
	Displays the INTERAC Online design (logo), wordmark (text "INTERAC Online) or both
Design and Wordmark Requirements (any page)	
	Other payment option logos: • Displays the INTERAC Online design (logo) if the merchant displays the trademarks or logos of other payment options. • Design is equal in size and no less prominent than other payment option trademarks.
	INTERAC wordmark: • INTERAC is always either in capital letters or italics (as in "the INTERAC Online service") • In the first use of the INTERAC Online wordmark, INTERAC is followed by the ® notation in superscript. For example, "*Interac*®" (English) or <<*Interac*^{MD}>> (French). • On the same page as the first occurence of the wordmark, the following language-appropriate footnote appears: • ® Trademark of Interac Inc. Used under licence" • ^{MD} Marque de commerce d'Interac Inc. Utilisée sous licence
Version of design	

Table 136: Checklist for web display requirements (continued)

Done	Requirement
	Uses the two-colour design on the web: • Horizontal version—height no shorter than 25 pixels (width-to-height ratio of 2:37:1) • Vertical version—width no narrower than 30 pixels (widteh-to-height ratio of 1:1:37)
"Learn more" information	
	Provides consumers with a link to www.interaconline.com/learn (preferably on the checkout page)
Confirmation page	
	States that the transaction is successful
	Displays the financial institution's name and confirmation number
	Provides the ability to print
Error page	
	Indicates that payment was unsuccsessful
	States that the order is cancelled or displays other payment options
Timeout message	
	Is displayed if consumer has less than 30 minutes to complete payment
Payment	
	Displays the total in Canadian dollars

Table 137: Checklist for security/privacy requirements

Done	Requirement
Merchant	
	Uses no less than 128-bit SSL encryption when collecting personal information
	Protects consumer information in accordance with applicable federal and provincial privacy legislation
	Adheres to the Canadian Code of Practice for Consumer Protection in Electronic Commerce

Table 138: Checklist for required screenshots

Done	Requirement
Provided screenshots	
	Checkout page (where customer selects INTERAC Online option)
	Confirmation page (one of the test case 1, 2, or 3)
	Error page (test case 4)

Appendix M Merchant Checklists for INTERAC® Online Payment Certification

Merchant Information

Name and URL	Merchant Name (English)	
	Homepage URL (English)	
	Merchant Name (French)	
	Homepage URL (French)	
Number	Merchant Number	
Transaction fee category (Circle one)	Government Education General	
Third-party service provider	Company name	
Service provider's merchant web application	Solution name	
	Version	

Merchant Requirements

Table 139: Checklist for web display requirements

Done	Requirement
	Checkout page
	Displays the INTERAC Online design (logo), wordmark (text "INTERAC Online) or both
	Design and Wordmark Requirements (any page)
	Other payment option logos: • Displays the INTERAC Online design (logo) if the merchant displays the trademarks or logos of other payment options. • Design is equal in size and no less prominent than other payment option trademarks.

Table 139: Checklist for web display requirements (continued)

Done	Requirement
	INTERAC wordmark: • INTERAC is always either in capital letters or italics (as in "the INTERAC Online service") • In the first use of the INTERAC Online wordmark, INTERAC is followed by the ® notation in superscript. For example, "*Interac*®" (English) or <<*Interac*^{MD}>> (French). • On the same page as the first occurence of the wordmark, the following language-appropriate footnote appears: • ® Trademark of Interac Inc. Used under licence" • ^{MD} Marque de commerce d'Interac Inc. Utilisée sous licence
Version of design	
	Uses the two-colour design on the web: • Horizontal version—height no shorter than 25 pixels (width-to-height ratio of 2:37:1) • Vertical version—width no narrower than 30 pixels (widteh-to-height ratio of 1:1:37)
"Learn more" information	
	Provides consumers with a link to www.interaconline.com/learn (preferably on the checkout page)
Confirmation page	
	States that the transaction is successful
	Displays the financial institution's name and confirmation number
	Provides ability to print
Error page	
	Indicates that payment was unsuccsessful
	States that the order is cancelled or displays other payment options
Timeout message	
	Is displayed if consumer has less than 30 minutes to complete payment
Payment	
	Displays the total in Canadian dollars

Table 140: Checklist for security/privacy requirements

Done	Requirement
Merchant	
	Uses no less than 128-bit SSL encryption when collecting personal information

Done	Requirement
	Protects consumer information in accordance with applicable federal and provincial privacy legislation
	Adheres to the Canadian Code of Practice for Consumer Protection in Electronic Commerce
	Provided screenshots
	Checkout page (where customer selects INTERAC Online option)
	Confirmation page (one of the test case 1, 2, or 3)
	Error page (test case 4)

Appendix N INTERAC® Online Payment Certification Test Case Detail

N.1 Common Validations

The Merchant sends a request to the INTERAC Online Merchant Test Tool, which validates the fields as follows:

- All mandatory fields are present.
- All fields are valid according to their definition in the *INTERAC Online Functional Specifications* (including field lengths, valid characters and so on).
- Merchant number is that of a valid registered merchant.
- Funded URL matches one of the merchant's registered funded URLs that were provided during merchant registration.
- The not funded URL matches one of the merchant's registered Not Funded URLs that were provided during merchant registration.
- No additional fields are present.

N.2 Test Cases

Table 141: Cases 1-3

Objective	To test that the merchant can do all of the following: • Send a valid request to the Gateway page • Receive a valid confirmation of funding from the Issuer Online Banking application • Issue a request for purchase completion to the acquirer • Receive an approved response from the acquirer.
Pre-requisites	None
Configuration	Merchant sends form posts to the Merchant Test Tool, which in turn responds to either the Funded or Not Funded URL. The Merchant is connected to an acquirer emulator, which can be set to confirm any request for payment confirmation. (That is, the back-end process of sending a 0200 Message to the issuer is emulated to always accept the purchase request).
Special tools required	None

Table 141: Cases 1-3 (continued)

Input data requirements	Acquirer must have registered the merchant using the administration system, and have supplied the following: • IDEBIT_FUNDEDURL(S) • IDEBIT_NOTFUNDEDURL(S) • HTTP REFERERURL(S) Data will be provided by the Merchant Test Tool.
Execution strategy	Initiate a payment at the merchant. The two least significant digits of the dollar amount must be equal to the test case number. For example, if you are executing test case 3, the format of the amount must be ### ### #03.##.
Expected out-come	The merchant indicates to the customer that the purchase was completed and presents a confirmation screen that includes (depending on the test case) the correct amount, the issuer name and the issuer confirmation number. Test case 1 • Issuer name: 123Bank • Issuer confirmation number: CONF#123 Test case 2 • Issuer name: Bank Éàêëï#$.,-/=?@' • Issuer confirmation number: #$.,-/=?@'UPdn9 Test case 3 • Issuer name: B • Issuer confirmation number: C
Applicable logs	• Merchant Test Tool logs • Screen capture of the merchant's confirmation page.

Table 142: Case 4

Objective	To test that the merchant handles a rejection in response to the acquirer
Pre-requisites	None
Configuration	Same as test cases 1-3 except that the acquirer emulator must be set to decline the request for mayment confirmation. (That is, to emulate the scenario in which an issuer sends a delcine in the 0210 response to the acquirer's 0200 message.)
Special tools required	None

Table 142: Case 4 (continued)

Input data requirements	Acquirer must have registered the merchant using the administration system, and have supplied the following: • IDEBIT_FUNDEDURL(S) • IDEBIT_NOTFUNDEDURL(S) • HTTP REFERERURL(S) Data will be provided by the Merchant Test Tool.
Execution strategy	Initiate a payment at the merchant for any amount where the two least significant dollar digits are 04. (That is, of the form ### ### #04.##.)
Expected outcome	The merchant indicates to the customer that the purchase was declined. Neither the issuer name nor the issuer confirmation number are displayed.
Applicable logs	Merchant Test Tool logs

Table 143: Cases 5-22

Objective	To test that a merchant safely handles redirections to the Funded URL with invalid data, and treats the transaction as funded.
Pre-requisites	None
Configuration	None. The acquirer emulator is not needed because the merchant does not submit any requests for payment confirmation.
Special tools required	None
Input data requirements	Acquirer must have registered the merchant using the administration system, and have supplied the following: • IDEBIT_FUNDEDURL(S) • IDEBIT_NOTFUNDEDURL(S) • HTTP REFERERURL(S) Data will be provided by the Merchant Test Tool.
Execution strategy	Initiate a payment at the merchant. The two least significant digits of the dollar amount must be equal to the test case number. For example, if you are executing test case 13, the format of the amount must be ### ### #13.##.
Expected outcome	The merchant indicates to the customer that the purchase was declined. Neither the issuer name nor the issuer confirmation number are displayed.
Applicable logs	Merchant Test Tool logs

Table 144: Case 23

Objective	To test that a merchant can receive a valid redirection from the issuer that indicates the payment was not funded.
Pre-requisites	None
Configuration	None. The acquirer emulator is not needed because the merchant does not submit any requests for payment confirmation.
Special tools required	None
Input data requirements	Acquirer must have registered the merchant using the administration system, and have supplied the following: • IDEBIT_FUNDEDURL(S) • IDEBIT_NOTFUNDEDURL(S) • HTTP REFERERURL(S) Data is provided by the Merchant Test Tool.
Execution strategy	Initiate a payment at the merchant for any amount where the two least significant dollar digits are 23. (That is, of the form ### ### #23.##.)
Expected outcome	The merchant indicates to the customer that the purchase was declined. Neither the issuer name nor the issuer confirmation number are displayed.
Applicable logs	Merchant Test Tool logs

Table 145: Cases 24-39

Objective	To test that a merchant safely handles redirections to the Not Funded URL with invalid data, and treats the transaction as not funded.
Pre-requisites	None
Configuration	None. The acquirer emulator is not needed because the merchant does not submit any requests for payment confirmation.
Special tools required	None

Table 145: Cases 24-39 (continued)

Input data requirements	Acquirer must have registered the merchant using the administration system, and have supplied the following: • IDEBIT_FUNDEDURL(S) • IDEBIT_NOTFUNDEDURL(S) • HTTP REFERERURL(S) Data is provided by the Merchant Test Tool.
Execution strategy	Initiate a payment at the merchant. The two least significant digits of the dollar amount must be equal to the test case number. For example, if you are executing test case 27, the format of the amount must be ### ### #27.##.
Expected out-come	The merchant indicates to the customer that the purchase was declined. Neither the issuer name nor the issuer confirmation number are displayed.
Applicable logs	Merchant Test Tool logs

N.3 Merchant front-end test case values

These values are automatically sent by the INTERAC Online Merchant Test Tool. They are provided here for reference only.

Table 146: Test cases 1 and 4—Funded URL

Redirection URL	Funded
ISSLANG	en
TRACK2	3728024906540591206=12010123456789XYZ
ISSCONF	CONF#123
ISSNAME	123Bank
INVOICE	(Same as supplied by merchant)
MERCHDATA	(Same as supplied by merchant)
VERSION	1

Table 147: Test case 2—Funded URL

Redirection URL	Funded
ISSLANG	en
TRACK2	5268051119993326=29129999999999999000
ISSCONF	#$.,-/=?@'UPdn9
ISSNAME	987Bank Éàêëï#$.,-/=?@'Àôùûüÿç

Table 147: Test case 2—Funded URL

INVOICE	(Same as supplied by merchant)
MERCHDATA	(Same as supplied by merchant)
VERSION	1

Table 148: Test case 3—Funded URL

Redirection URL	Funded
ISSLANG	fr
TRACK2	453781122255=1001ABC11223344550000000
ISSCONF	C
ISSNAME	B
INVOICE	(Same as supplied by merchant)
MERCHDATA	(Same as supplied by merchant)
VERSION	123

Table 149: Test cases 5-22—invalid fields, Funded URL

Test case	Purpose	Field	Value
5	missing field	IDEBIT_INVOICE	(missing)
6	missing field	IDEBIT_MERCHDATA	(missing)
7	missing field	IDEBIT_ISSLANG	(missing)
8	missing field	IDEBIT_TRACK2	(missing)
9	missing field	IDEBIT_ISSCONF	(missing)
10	missing field	IDEBIT_ISSNAME	(missing)
11	missing field	IDEBIT_VERSION	(missing)
12	missing field	IDEBIT_TRACK2, IDEBIT_ ISSCONF, IDEBIT_ISSNAME	(missing)
13	wrong value	IDEBIT_INVOICE	XXX
14	wrong value	IDEBIT_MERCHDATA	XXX
15	invalid value	IDEBIT_ISSLANG	de
16	value too long	IDEBIT_TRACK2	3728024906540591206=12010123456789XYZA
17	invalid check digit	IDEBIT_TRACK2	3728024906540591207=12010123456789XYZ

Table 149: Test cases 5-22—invalid fields, Funded URL (continued)

Test case	Purpose	Field	Value
18	field too long	IDEBIT_ISSCONF	Too long confirm
19	invalid character	IDEBIT_ISSCONF	CONF<123
20	field too long	IDEBIT_ISSNAME	Very, very, very long issuer name
21	invalid character	IDEBIT_ISSNAME	123<Bank
22	invalid value	IDEBIT_VERSION	2

Table 150: Test case 23—valid data, Not Funded URL

Redirection URL	Not funded
ISSLANG	en
INVOICE	(Same as supplied by merchant)
MERCHDATA	(Same as supplied by merchant)
VERSION	1

Table 151: Test cases 5-22—invalid fields, Funded URL

Test case	Purpose	Field	Value
24	missing field	IDEBIT_INVOICE	(missing)
25	missing field	IDEBIT_MERCHDATA	(missing)
26	missing field	IDEBIT_ISSLANG	(missing)
27	IDEBIT_TRACK2 is present and valid	IDEBIT_TRACK2	3728024906540591206=12010123456789XYZ
28	IDEBIT_ISSCONF is present and valid	IDEBIT_ISSCONF	CONF#123
29	IDEBIT_ISSNAME is present and valid	IDEBIT_ISSNAME	12Bank
30	missing field	IDEBIT_VERSION	(missing)
31	wrong value	IDEBIT_INVOICE	XXX
32	invalid value	IDEBIT_INVOICE	invalid </html> tricky data
33	wrong value	IDEBIT_MERCHDATA	XXX

Table 151: Test cases 5-22—invalid fields, Funded URL (continued)

Test case	Purpose	Field	Value
34	invalid value	IDEBIT_MERCHDATA	<2000 characters in the range hex 20-7E
35	invalid value	IDEBIT_ISSLANG	de
36	invalid IDEBIT_ TRACK2 is present	IDEBIT_TRACK2	INVALIDTRACK2, incorrect format and too long
37	invalid IDEBIT_ ISSCONF is present	IDEBIT_ISSCONF	Too long confirm
38	invalid IDEBIT_ ISSNAME is present	IDEBIT_ISSNAME	Very, very, very long issuer name
39	invalid value	IDEBIT_VERSION	2