BE PAYMENT READY

.NET - Moneris Gateway API - Credential on File

Version: 1.0.3

Applies to Canadian integrations only

Table of Contents

Getting Help

Moneris has help for you at every stage of the integration process.

Getting Started	During Development	Production
Contact our Client Integration Specialists: clientintegrations@moneris.com Hours: Monday – Friday, 8:30am to 8 pm ET	If you are already working with an integration specialist and need technical development assistance, contact our eProducts Technical Consultants: 1-866-319-7450 eproducts@moneris.com Hours: 8am to 8pm ET	If your application is already live and you need production support, contact Moneris Customer Service: onlinepayments@moneris.com 1-866-319-7450 Available 24/7

For additional support resources, you can also make use of our community forums at

http://community.moneris.com/product-forums/

1 About Credential on File

When storing customers' credit card credentials for use in future authorizations, or when using these credentials in subsequent transactions, card brands now require merchants to indicate this in the transaction request.

In the Moneris API, this is handled by the Moneris Gateway via the inclusion of the Credential on File info object and its variables in the transaction request.

While the requirements for handling Credential on File transactions relate to Visa, Mastercard and Discover only, in order to avoid confusion and prevent error, please implement these changes for all card types and the Moneris system will then correctly flow the relevant card data values as appropriate.

While in the testing phase, we recommend that you test with Visa cards because implementation for the other card brands is still in process.

NOTE: If either the first transaction or a Card Verification authorization is declined when attempting to store cardholder credentials, those credentials cannot be stored —therefore the merchant must not use the credential for any subsequent transactions.

2 Credential on File Info Object and Variables

The Credential on File Info object is nested within the request for the applicable transaction types.

Object:

cof

Variables in the cof object:

Payment Indicator
Payment Information
Issuer ID

For more information, see Definitions of Request Fields – Credential on File.

3 Initial Transactions in Credential on File

When sending an *initial* transaction with the Credential on File Info object, i.e., a transaction request where the cardholder's credentials are being stored for the *first* time, it is important to understand the following:

- You must send the cardholder's Card Verification Details (CVD)
- **Issuer ID** will be sent without a value on the initial transaction, because it is received in the response to that initial transaction; for all *subsequent* merchant-intiated transactions and all administrative transactions you send this **Issuer ID**
- The **payment information** field will always be a value of 0

4 Credential on File and Converting Temporary Tokens

In the event you decide to convert a temporary token representing cardholder credentials into a permanent token, these credentials become stored credentials, and therefore necessary to send Credential on File information.

For Temporary Token Add transactions where you subsequently decide to convert the temporary token into a permanent token (stored credentials):

1. Send a transaction request that includes the Credential on File Info object to get the Issuer ID; this can be a Card Verification, Purchase or Pre-Authorization request
2. After completing the transaction, send the Vault Add Token request with the Credential on File object(Issuer ID only) in order to convert the temporary token to a permanent one.

5 Credential on File Transaction Types

The Credential on File Info object applies to the following transaction types:

- Purchase
- Pre-Authorization
- Purchase with 3-D Secure – cavvPurchase
- Purchase with 3-D Secure and Recurring Billing
- Pre-Authorization with 3-D Secure – cavvPreauth
- Purchase with Vault – ResPurchaseCC
- Pre-Authorization with Vault – ResPreauthCC
- Card Verification with AVS and CVD
- Card Verification with Vault – ResCardVerificationCC
- Vault Add Credit Card – ResAddCC
- Vault Update Credit Card – ResUpdateCC
- Vault Add Token – ResAddToken
- Vault Tokenize Credit Card – ResTokenizeCC
- Recurring Billing transactions

NOTE: For the following transactions, the Credential on File Info object also applies, but Moneris sends the indicators on your behalf:

- Re-Authorization
- Level 2/3 transactions

5.1 Purchase

Purchase transaction object definition

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

HttpsPostRequest object for Purchase transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(purchase);
```

Purchase transaction values

Table 1: 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(expiry_date);`
E-commerce indicator	String	1-character alpha-numeric	`purchase.SetCryptType(crypt);`

Table 2: Purchase transaction object optional values

Value	Type	Limits	Set method
Status Check	Boolean	true/false	`mpgReq.SetStatusCheck (status_check);`
Customer information	Object	N/A	`purchase.SetCustInfo (customer);`
AVS	Object	N/A	`purchase.SetAvsInfo (avsCheck);`
CVD **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not**	Object	N/A	`purchase.SetCvdInfo (cvdCheck);`

Table 2: Purchase transaction object optional values

Value	Type	Limits	Set method
store CVD information.			
Convenience fee **NOTE:** This variable does not apply to Credential on File transactions.	Object	N/A	`purchase.SetConvFeeInfo (convFeeInfo);`
Recurring billing	Object	N/A	`purchase.SetRecur(recurring_ cycle);`
Dynamic descriptor	String	20-character alpha-numeric	`purchase .SetDynamicDescriptor (dynamic_descriptor);`
Wallet indicator[1]	String	3-character alpha-numeric	`purchase.SetWalletIndicator (wallet_indicator);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`purchase.SetCofInfo(cof);`

[1]Available to Canadian integrations only.

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated transactions following the first one; upon sending the first transaction, the Issuer ID value is received in the transaction response and then used in subsequent transaction requests (Issuer ID does not apply for Discover or Union Pay).	*String*	15-character alpha-numeric variable length	`cof.SetIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Indicator	*String*	1-character alphabetic	`cof.SetPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Information	*String*	1-character numeric	`cof.SetPaymentInformation ("PAYMENT_INFO_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File

Sample Purchase

```
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" + DateTime.Now.ToString("yyyyMMddhhmmss");
string store_id = "store5";
string api_token = "yesguy";
```

Sample Purchase

```
string amount = "5.00";
string pan = "4242424242424242";
string expdate = "1901"; //YYMM format
string crypt = "7";
string processing_country_code = "CA";
bool status_check = false;
CofInfo cof = new CofInfo();
cof.SetPaymentIndicator("U");
cof.SetPaymentInformation("2");
cof.SetIssuerId("12345678901234");
Purchase purchase = new Purchase();
purchase.SetOrderId(order_id);
purchase.SetAmount(amount);
purchase.SetPan(pan);
purchase.SetExpDate("2011");
purchase.SetCryptType(crypt);
purchase.SetDynamicDescriptor("2134565");
//purchase.SetWalletIndicator(""); //Refer to documentation for details
purchase.SetCofInfo(cof);

//Optional - Set for Multi-Currency only
//setAmount must be 0.00 when using multi-currency
//purchase.SetMCPAmount("500"); //penny value amount 1.25 = 125
//purchase.SetMCPCurrencyCode("840"); //ISO-4217 country currency 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(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 = " + receipt.GetTicket());
Console.WriteLine("TimedOut = " + receipt.GetTimedOut());
Console.WriteLine("IsVisaDebit = " + receipt.GetIsVisaDebit());
Console.WriteLine("HostId = " + receipt.GetHostId());
Console.WriteLine("MCPAmount = " + receipt.GetMCPAmount());
Console.WriteLine("MCPCurrencyCode = " + receipt.GetMCPCurrencyCode());
Console.WriteLine("IssuerId = " + receipt.GetIssuerId());
Console.ReadLine();
}
catch (Exception e)
```

Sample Purchase

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

5.2 Pre-Authorization

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

Table 3: Pre-Authorization object required values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`preauth.SetOrderId(order_id);`
Amount	String	9-character decimal	`preauth.SetAmount(amount);`
Credit card number	String	20-character numeric	`preauth.SetPan(pan);`
Expiry date	String	4-character numeric	`preauth.SetExpdate(expiry_date);`
E-Commerce indicator	String	1-character alpha-numeric	`preauth.SetCryptType(crypt);`

Table 4: Pre-Authorization object optional values

Value	Type	Limits	Set method
Status Check	Boolean	true/false	`mpgReq.SetStatusCheck (status_check);`
Dynamic descriptor	String	20-character alpha-numeric	`preauth.SetDynamicDescriptor (dynamic_descriptor);`
Customer inform-ation	Object	N/A	`preauth.SetCustInfo (customer);`
AVS	Object	N/A	`preauth.SetAvsInfo (avsCheck);`
CVD **NOTE:** When storing credentials on the ini-tial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	Object	N/A	`preauth.SetCvdInfo (cvdCheck);`

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`preauth.SetCustId(cust_id);`
Wallet indicator[1]	String	3-character alpha-numeric	`preauth.SetWalletIndicator (wallet_indicator);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`cof.SetCofInfo(cof);`

[1]Available to Canadian integrations only.

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated transactions following the first one; upon sending the first transaction, the Issuer ID value is received in the transaction response and then used in subsequent transaction requests (Issuer ID does not apply for Discover or Union Pay).	*String*	15-character alpha-numeric variable length	`cof.SetIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Indicator	*String*	1-character alphabetic	`cof.SetPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Information	*String*	1-character numeric	`cof.SetPaymentInformation ("PAYMENT_INFO_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File

Sample Pre-Authorization

```
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");
```

Sample Pre-Authorization

```
string amount = "5.00";
string pan = "4242424242424242";
string expdate = "0412";
string crypt = "7";
string processing_country_code = "CA";
bool status_check = false;
CofInfo cof = new CofInfo();
cof.SetPaymentIndicator("U");
cof.SetPaymentInformation("2");
cof.SetIssuerId("12345678901234");
PreAuth preauth = new PreAuth();
preauth.SetOrderId(order_id);
preauth.SetAmount(amount);
preauth.SetPan(pan);
preauth.SetExpDate(expdate);
preauth.SetCryptType(crypt);
//preauth.SetWalletIndicator(""); //Refer to documentation for details
preauth.SetCofInfo(cof);
//Optional - Set for Multi-Currency only
//setAmount must be 0.00 when using multi-currency
//preauth.SetMCPAmount("500"); //penny value amount 1.25 = 125
//preauth.SetMCPCurrencyCode("840"); //ISO-4217 country currency 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(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());
//Console.WriteLine("StatusMessage = " + receipt.GetStatusMessage());
Console.WriteLine("MCPAmount = " + receipt.GetMCPAmount());
Console.WriteLine("MCPCurrencyCode = " + receipt.GetMCPCurrencyCode());
Console.WriteLine("IssuerId = " + receipt.GetIssuerId());
Console.ReadLine();
}
catch (Exception e)
{
```

Sample Pre-Authorization

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

5.3 Purchase with Vault – ResPurchaseCC

Purchase with Vault transaction object definition

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

HttpsPostRequest object for Purchase with Vault transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resPurchaseCC);
```

Purchase with Vault transaction values

Table 5: Purchase with Vault transaction object mandatory values

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

Value	Type	Limits	Set method
E-commerce indicator	String	1-character alpha-numeric	`resPurchaseCC.SetCryptType (crypt);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`cof.SetCofInfo(cof);`

Table 6: Purchase with Vault transaction optional values

Value	Type	Limits	Set method
Status Check	Boolean	true/false	`mpgReq.SetStatusCheck (status_check);`
Expiry date	String	4-character numeric YYMM format. (Note that this is reversed from the date displayed on the card, which is MMYY)	`resPurchaseCC.SetExpdate (expiry_date);`
Customer ID	String	50-character alpha-numeric	`resPurchaseCC.SetCustId (cust_id);`
Dynamic descriptor	String	20-character alpha-numeric	`resPurchaseCC .SetDynamicDescriptor (dynamic_descriptor);`
Customer information	Object	N/A	`resPurchaseCC.SetCustInfo (customer);`

Value	Type	Limits	Set method
AVS information	Object	N/A	`resPurchaseCC.SetAvsInfo (avsCheck);`
CVD information **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	Object	N/A	`resPurchaseCC.SetCvdInfo (cvdCheck);`
Recurring billing	Object	N/A	`resPurchaseCC.SetRecur (recurring_cycle);`

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated transactions following the first one; upon sending the first transaction, the Issuer ID value is received in the transaction response and then used in subsequent transaction requests (Issuer ID does not apply for Discover or Union Pay).	*String*	15-character alpha-numeric variable length	`cof.SetIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Indicator	*String*	1-character alphabetic	`cof.SetPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Information	*String*	1-character numeric	`cof.SetPaymentInformation ("PAYMENT_INFO_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File

Sample Purchase with Vault

```
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";
```

Sample Purchase with Vault

```
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";
string processing_country_code = "CA";
bool status_check = false;
CofInfo cof = new CofInfo();
cof.SetPaymentIndicator("U");
cof.SetPaymentInformation("2");
cof.SetIssuerId("12345678901234");
ResPurchaseCC resPurchaseCC = new ResPurchaseCC();
resPurchaseCC.SetDataKey(data_key);
resPurchaseCC.SetOrderId(order_id);
resPurchaseCC.SetCustId(cust_id);
resPurchaseCC.SetAmount(amount);
resPurchaseCC.SetCryptType(crypt_type);
resPurchaseCC.SetDynamicDescriptor(descriptor);
resPurchaseCC.SetCofInfo(cof);
//resPurchaseCC.SetExpDate("1511"); //optional - use for temp token only
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 = " + receipt.GetIsVisaDebit());
Console.WriteLine("IssuerId = " + receipt.GetIssuerId());

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());
```

Sample Purchase with Vault

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

5.4 Pre-Authorization with Vault – ResPreauthCC

Pre-Authorization with Vault transaction object definition

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

HttpsPostRequest object for Pre-Authorization with Vault transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resPreauthCC);
```

Pre-Authorization with Vault transaction values

Table 7: Pre-Authorization with Vault transaction object mandatory values

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

Table 7: Pre-Authorization with Vault transaction object mandatory values (continued)

Value	Type	Limits	Set method
E-commerce indicator	String	1-character alpha-numeric	`resPreauthCC.SetCryptType (crypt);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`resPreauthCC.SetCofInfo (cof);`

Table 8: Pre-Authorization with Vault transaction optional values

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

Value	Type	Limits	Set method
Customer information	Object	N/A	`resPreauthCC.SetCustInfo (customer);`
AVS information	Object	N/A	`resPreauthCC.SetAvsInfo (avsCheck);`
CVD information **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	Object	N/A	`resPreauthCC.SetCvdInfo (cvdCheck);`

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated transactions following the first one; upon sending the first transaction, the Issuer ID value is received in the transaction response and then used in subsequent transaction requests (Issuer ID does not apply for Discover or Union Pay).	*String*	15-character alpha-numeric variable length	`cof.SetIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Indicator	*String*	1-character alphabetic	`cof.SetPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Information	*String*	1-character numeric	`cof.SetPaymentInformation ("PAYMENT_INFO_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File

Sample Pre-Authorization with Vault

```
namespace Moneris
{
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";
```

Sample Pre-Authorization with Vault

```
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;
CofInfo cof = new CofInfo();
cof.SetPaymentIndicator("U");
cof.SetPaymentInformation("2");
cof.SetIssuerId("12345678901234");
ResPreauthCC resPreauthCC = new ResPreauthCC();
resPreauthCC.SetDataKey(data_key);
resPreauthCC.SetOrderId(order_id);
resPreauthCC.SetCustId(cust_id);
resPreauthCC.SetAmount(amount);
resPreauthCC.SetCryptType(crypt_type);
resPreauthCC.SetDynamicDescriptor(dynamic_descriptor);
resPreauthCC.SetCofInfo(cof);
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 = " + 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("IssuerId = " + receipt.GetIssuerId());

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());
```

Sample Pre-Authorization with Vault

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

5.5 Vault Tokenize Credit Card and Credential on File

When you want to store cardholder credentials from previous transactions into the Vault, you use the Vault Tokenize Credit Card transaction request. Credential on File rules require that only previous transactions with the Credential on File Info object can be tokenized to the Vault.

For more information about this transaction, see 5.5.1 Vault Tokenize Credit Card – ResTokenizeCC.

5.5.1 Vault Tokenize Credit Card – 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. Previous transactions to be tokenized must have included the Credential on File Info object.

The Issuer ID received in the previous transaction response is sent in the Vault Tokenize Credit Card request to reference that this is a stored credential.

Basic transactions that can be tokenized are:

- Purchase
- Pre-Authorization
- Card Verification

The tokenization process is outlined below :

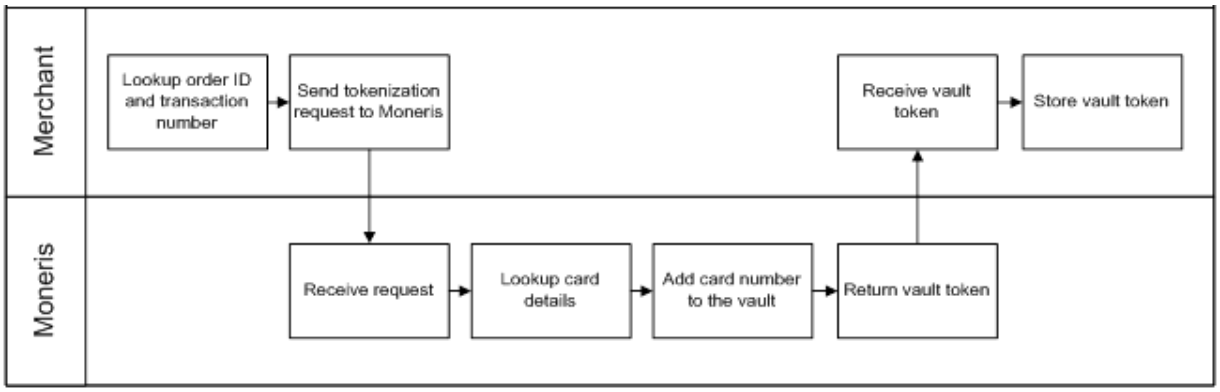

Figure 1: Tokenize process diagram

Vault Tokenize Credit Card transaction object definition

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

HttpsPostRequest object for Vault Tokenize Credit Card transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resTokenizeCC);
```

Vault Tokenize Credit Card transaction values

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

Table 9: Vault Tokenize Credit Card transaction object mandatory values

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

Table 10: Vault Tokenize Credit Card transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`resTokenizeCC.SetCustId (cust_id);`
Email address	String	30-character alpha-numeric	`resTokenizeCC.SetEmail (email);`
Phone number	String	30-character alpha-numeric	`resTokenizeCC.SetPhone (phone);`
Note	String	30-character alpha-numeric	`resTokenizeCC.SetNote(note);`

Value	Type	Limits	Set method
AVS information	Object	N/A	`resTokenizeCC.SetAvsInfo (avsCheck);`
Data key format[1]	String	2-character alpha-numeric	`resTokenizeCC .SetDataKeyFormat(data_key_ format)`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	*Object*	N/A	`resTokenizeCC.SetCofInfo (cof);`

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated transactions following the first one; upon sending the first transaction, the Issuer ID value is received in the transaction response and then used in subsequent transaction requests (Issuer ID does not apply for Discover or Union Pay).	*String*	15-character alpha-numeric variable length	`cof.SetIssuerId("VALUE_FOR_ ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File

[1]Available to Canadian integrations only.

Any field that is not set in the tokenize request is not stored with the transaction. That is, Moneris Gateway does not automatically take the optional information that was part of the original transaction.

The ResolveData that is returned in the response fields indicates what values were registered for this profile.

Sample Vault Tokenize Credit Card

```
namespace Moneris
{
using System;
using System.Text;
using System.Collections;
public class TestCanadaResTokenizeCC
{
public static void Main(string[] args)
{
string store_id = "store1";
string api_token = "yesguy";
string order_id = "1000189096";
string txn_number = "880416-0_10";
string phone = "0000000000";
string email = "bob@smith.com";
string note = "my note";
string cust_id = "customer1";
string data_key_format = "0";
string processing_country_code = "CA";
bool status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
CofInfo cof = new CofInfo();
cof.SetIssuerId("168451306048014");
ResTokenizeCC resTokenizeCC = new ResTokenizeCC();
resTokenizeCC.SetOrderId(order_id);
resTokenizeCC.SetTxnNumber(txn_number);
resTokenizeCC.SetCustId(cust_id);
resTokenizeCC.SetPhone(phone);
resTokenizeCC.SetEmail(email);
resTokenizeCC.SetNote(note);
resTokenizeCC.SetAvsInfo(avsCheck);
resTokenizeCC.SetCofInfo(cof);
//resTokenizeCC.SetDataKeyFormat(data_key_format); //optional
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(resTokenizeCC);
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 Vault Tokenize Credit Card

```
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());
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 Definition of Response Fields (page 1).

5.6 Card Verification and Credential on File Transactions

In certain cases, some Credential on File transactions require the prior use of a Card Verification with AVS and CVD transaction.

In the absence of a Purchase or Pre-Authorization, a Card Verification transaction is used to get the unique Issuer ID value that is used in subsequent Credential on File transactions. Issuer ID is a variable included in the nested Credential on File Info object.

For all first-time transactions, including Card Verification transactions, you must also request the card-holder's Card Verification Details (CVD). For more on CVD, see 1 Card Validation Digits (CVD).

For a complete list of these variables, see each transaction type or Definitions of Request Fields – Credential on File

The Card Verification request, including the Credential on File Info object, must be sent immediately prior to storing cardholder credentials.

For information about Card Verification, see 5.6.6 Card Verification with AVS and CVD.

5.6.1 When to Use Card Verification With COF

If you are not sending a Purchase or Pre-Authorization transaction (i.e., you are not charging the customer immediately), you must use Card Verification (or in the case of Vault Add Token, Card Verification with Vault) first before running the transaction in order to get the Issuer ID.

Transactions this applies to:

Vault Add Credit Card – ResAddCC
Vault Update Credit Card – ResUpdateCC
Vault Add Token – ResAddToken
Recurring Billing transactions (first in series), if:

- the first transaction does not begin immediately

5.6.2 Credential on File and Vault Add Token

For Vault Add Token transactions:

1. Send Card Verification with Vault transaction request including the Credential on File object to get the Issuer ID
2. Send the Vault Add Token request including the Credential on File object (with Issuer ID only; other fields are not applicable)

For more on this transaction type, see 5.6.2.1 Vault Add Token – ResAddToken.

5.6.2.1 Vault Add Token – ResAddToken

Things to Consider:

- This transaction is used to convert a temporary token into a permanent token for storage in the Moneris Vault
- If you intend to store the token for use in future transactions (i.e., Credential on File transactions), **first** you must send either a Vault financial transaction (Purchase with Vault or Pre-Authorization with Vault) or a Card Verification with Vault in order to get the Issuer ID
- The Vault Add Token request uses the Issuer ID to indicate that it is referencing stored credentials

Vault Add Token transaction object definition

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

HttpsPostRequest object for Vault Add Token transaction

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

```
mpgReq.SetTransaction(resAddToken);
```

Vault Add Token transaction values

Table 11: Vault Add Token transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	28-character alpha-numeric	`resAddToken.SetData(data_key);`
E-commerce indicator	String	1-character alpha-numeric	`resAddToken.SetCryptType(crypt);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`resaddcc.SetCofInfo(cof);`

Table 12: Vault Add Token transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`resAddToken.SetCustId(cust_id);`
AVS information	Object	N/A	`resAddToken.SetAvsInfo(avsCheck);`
Email address	String	30-character alpha-numeric	`resAddToken.SetEmail(email);`

Value	Type	Limits	Set method
Phone number	String	30-character alpha-numeric	`resAddToken.SetPhone(phone);`
Note	String	30-character alpha-numeric	`resAddToken.SetNote(note);`
Data key format[1]	String	2-character alpha-numeric	`resAddToken.SetDataKeyFormat (data_key_format)`

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated transactions following the first one; upon sending the first transaction, the Issuer ID value is received in the transaction response and then used in subsequent transaction requests (Issuer ID does not apply for Discover or Union Pay).	*String*	15-character numeric variable length	`cof.SetIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File

Sample Vault Add Token

```
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";
```

[1]Available to Canadian integrations only.

Sample Vault Add Token

```
string email = "bob@smith.com";
string note = "my note";
string cust_id = "customer1";
string crypt_type = "7";
string data_key_format = "0";
string processing_country_code = "CA";
bool status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
CofInfo cof = new CofInfo();
cof.SetIssuerId("168451306048014");
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);
resAddToken.SetCofInfo(cof);
//resAddToken.SetDataKeyFormat(data_key_format); //optional
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 = " + receipt.GetResDataAvsStreetName());
Console.WriteLine("Avs Zipcode = " + receipt.GetResDataAvsZipcode());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

5.6.3 Credential on File and Vault Update Credit Card

For Vault Update Credit Card transactions where you are updating the credit card number:

1. Send Card Verification transaction request including the Credential on File object to get the Issuer ID
2. Send the Vault Update Credit Card request including the Credential on File Info object (Issuer ID only).

For more on this transaction type, see 5.6.3.1 Vault Update Credit Card – ResUpdateCC.

5.6.3.1 Vault Update Credit Card – ResUpdateCC

Things to Consider:

- 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.
- This will update a profile to contain Credit Card information by referencing the profile's unique data_key. If the profile which is being updated was already a Credit Card profile, all information contained within it will simply be updated as indicated by the submitted fields. This means that all fields are optional, and only those fields that are submitted will be updated.
- To update a specific field on the profile, only set that specific element using the corresponding set method.

Vault Update Credit Card transaction object definition

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

HttpsPostRequest object for Vault Update Credit Card transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resUpdateCC);
```

Vault Update Credit Card transaction values

Table 13: Vault Update Credit Card transaction object mandatory values

Value	Type	Limits	Set method
Data key	String	25-character alpha-numeric	`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 variables in the optional values table below 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.

> **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 14: Vault Update Credit Card transaction optional values

Value	Type	Limits	Set method
Credit card number	String	20-character alpha-numeric	`resUpdateCC.SetPan(pan);`
Expiry date	String	4-character alpha-numeric (YYMM format)	`resUpdateCC.SetExpdate (expiry_date);`
E-commerce indicator	String	1-character alpha-numeric	`resUpdateCC.SetCryptType (crypt);`
Customer ID	String	50-character alpha-numeric	`resUpdateCC.SetCustId(cust_ id);`
AVS information	Object	n/a	`resUpdateCC.SetAvsInfo (avsCheck);`
Email address	String	30-character alpha-numeric	`resUpdateCC.SetEmail(email);`

Value	Type	Limits	Set method
Phone number	String	30-character alpha-numeric	`resUpdateCC.SetPhone(phone);`
Note	String	30-character alpha-numeric	`resUpdateCC.SetNote(note);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`resUpdateCC.SetCofInfo(cof);`

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated transactions following the first one; upon sending the first transaction, the Issuer ID value is received in the transaction response and then used in subsequent transaction requests (Issuer ID does not apply for Discover or Union Pay).	*String*	15-character alpha-numeric variable length	`cof.SetIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File

Sample Vault Update Credit Card

```
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");
CofInfo cof = new CofInfo();
cof.SetIssuerId("168451306048014");
ResUpdateCC resUpdateCC = new ResUpdateCC();
resUpdateCC.SetDataKey(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);
resUpdateCC.SetCofInfo(cof);
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);
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());
```

Sample Vault Update Credit Card

```
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);
}
}
}
}
```

5.6.4 Credential on File and Vault Add Credit Card

For Vault Add Credit Card transactions:

1. Send Card Verification transaction request including the Credential on File object to get the Issuer ID
2. Send the Vault Add Credit Card request including the Credential on File Info object (Issuer ID only)

For more on this transaction type, see 5.6.4.1 Vault Add Credit Card – ResAddCC.

5.6.4.1 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

Table 15: Vault Add Credit Card transaction object mandatory values

Value	Type	Limits	Set method
Credit card number	String	20-character alpha-numeric	`resaddcc.SetPan(pan);`
Expiry date	String	4-character alpha-numeric (YYMM format)	`resaddcc.SetExpdate(expiry_date);`
E-commerce indicator	String	1-character alpha-numeric	`resaddcc.SetCryptType (crypt);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`resaddcc.SetCofInfo(cof);`

Table 16: Vault Add Credit Card transaction optional values

Value	Type	Limits	Set method
Customer ID	String	50-character alpha-numeric	`resaddcc.SetCustId(cust_id);`
AVS information	Object	N/A	`resaddcc.SetAvsInfo (avsCheck);`
Email address	String	30-character alpha-numeric	`resaddcc.SetEmail(email);`
Phone number	String	30-character alpha-	`resaddcc.SetPhone(phone);`

Table 16: Vault Add Credit Card transaction optional values

Value	Type	Limits	Set method
		numeric	
Note	String	30-character alpha-numeric	`resaddcc.SetNote(note);`
Data key format[1]	String	2-character alpha-numeric	`resaddcc.SetDataKeyFormat (data_key_format)`

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated transactions following the first one; upon sending the first transaction, the Issuer ID value is received in the transaction response and then used in subsequent transaction requests (Issuer ID does not apply for Discover or Union Pay).	*String*	15-character alpha-numeric variable length	`cof.SetIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File

Sample Vault Add Credit Card

```
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";
```

[1]Available to Canadian integrations only.

Sample Vault Add Credit Card

```
string expdate = "1912";
string phone = "0000000000";
string email = "bob@smith.com";
string note = "my note";
string cust_id = "customer1";
string crypt_type = "7";
string data_key_format = "0";
string processing_country_code = "CA";
bool status_check = false;
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");
CofInfo cof = new CofInfo();
cof.SetIssuerId("168451306048014");
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(data_key_format); //optional
resaddcc.SetCofInfo(cof);
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());
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("IssuerId = " + receipt.GetIssuerId());
Console.ReadLine();
```

Sample Vault Add Credit Card

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

5.6.5 Credential on File and Recurring Billing

> **NOTE:** The value of the **payment indicator** field must be **R** when sending Recurring Billing transactions.

For Recurring Billing transactions which are set to start **immediately**:

- Send a Purchase transaction request with both the Recurring Billing and Credential on File info objects.

For Recurring Billing transactions which are set to start on a **future** date:

1. Send Card Verification transaction request including the Credential on File info object to get the Issuer ID
2. Send Purchase transaction request with the Recur and Credential on File info objects included

For updating a Recurring Billing series where you are updating the cardholder credentials (does not apply if you are only modifying the schedule or amount in a recurring series):

1. Send Card Verification request including the Credential on File info object to get the Issuer ID
2. Send a Recurring Billing Update transaction

For more information about the Recurring Billing object, see Definition of Request Fields – Recurring.

5.6.6 Card Verification with AVS and CVD

> **Things to Consider:**
> - The Card Verification transaction is only supported by Visa, MasterCard and Discover

- For some Credential on File transactions, Card Verification with AVS and CVD is used as a prior step to get the Issuer ID used in the subsequent transaction
- 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

Table 17: Card Verification transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`cardVerification.SetOrderId (order_id);`
Credit card number	String	20-character alpha-numeric	`cardVerification.SetPan (pan);`
Expiry date	String	4-character alpha-numeric (YYMM format)	`cardVerification.SetExpdate (expiry_date);`

Table 17: Card Verification transaction object mandatory values

Value	Type	Limits	Set method
E-commerce indicator	String	1-character alpha-numeric	`cardVerification .SetCryptType(crypt);`
AVS	Object	N/A	`cardVerification.SetAvsInfo (avsCheck);`
CVD **NOTE:** When storing credentials on the initial transaction, the CVD object must be sent; for subsequent transactions using stored credentials, CVD can be sent with cardholder-initiated transactions only—**merchants must not store CVD information**.	Object	N/A	`cardVerification.SetCvdInfo (cvdCheck);`

Table 18: Basic Card Verification transaction object optional values

Value	Type	Limits	Set Method
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`cardVerification.SetCofInfo (cof);`

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated transactions following the first one; upon sending the first transaction, the Issuer ID value is received in the transaction response and then used in subsequent transaction requests (Issuer ID does not apply for Discover or Union Pay).	*String*	15-character alpha-numeric variable length	`cof.SetIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Indicator	*String*	1-character alphabetic	`cof.SetPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Information	*String*	1-character numeric	`cof.SetPaymentInformation ("PAYMENT_INFO_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File

Sample Card Verification

```
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" + DateTime.Now.ToString("yyyyMMddhhmmss");
string pan = "4242424242424242";
string expdate = "1901"; //YYMM format
string crypt = "7";
```

Sample Card Verification

```
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");
CofInfo cof = new CofInfo();
cof.SetPaymentIndicator("U");
cof.SetPaymentInformation("2");
cof.SetIssuerId("12345678901234");
CardVerification cardVerification = new CardVerification();
cardVerification.SetOrderId(order_id);
cardVerification.SetPan(pan);
cardVerification.SetExpDate(expdate);
cardVerification.SetCryptType(crypt);
cardVerification.SetAvsInfo(avsCheck);
cardVerification.SetCvdInfo(cvdCheck);
cardVerification.SetCofInfo(cof);
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 = " + 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("IssuerId = " + receipt.GetIssuerId());
Console.ReadLine();
}
catch (Exception e)
{
Console.WriteLine(e);
}
}
}
}
```

5.6.7 Card Verification with Vault – ResCardVerificationCC

Things to Consider:

- This transaction type only applies to Visa, Mastercard and Discover transactions
- This transaction is also known as an "account status inquiry"
- The card number and expiry date for this transaction are passed using a token, as represented by the data key value
- When using a temporary token (e.g., such as with Hosted Tokenization) **and** you intend to store the cardholder credentials, this transaction must be run prior to running the Vault Add Token transaction

Card Verification object definition

```
CardVerification resCardVerificationCC = new CardVerification();
```

HttpsPostRequest object for Card Verification transaction

```
HttpsPostRequest mpgReq = new HttpsPostRequest();

mpgReq.SetTransaction(resCardVerificationCC);
```

Card Verification transaction values

Table 19: Card Verification with Vault transaction object mandatory values

Value	Type	Limits	Set method
Order ID	String	50-character alpha-numeric	`resCardVerificationCC .SetOrderId(order_id);`
Data key	String	25-character alpha-numeric	`resCardVerificationCC .SetDataKeyFormat(data_key_ format)`
E-commerce indicator	String	1-character alpha-numeric	`resCardVerificationCC .SetCryptType(crypt);`

Table 19: Card Verification with Vault transaction object mandatory values

Value	Type	Limits	Set method
AVS	Object	N/A	`resCardVerificationCC .SetAvsInfo(avsCheck);`
CVD	Object	N/A	`resCardVerificationCC .SetCvdInfo(cvdCheck);`
Credential on File Info `cof` **NOTE:** This is a nested object within the transaction, and required when storing or using the customer's stored credentials. The Credential on File Info object has its own request variables, listed in blue in the table below, "Credential on File Object Request Variables".	Object	N/A	`resCardVerificationCC .SetCofInfo(cof);`

Credential on File Transaction Object Request Fields

Value	Type	Limits	Set Method
Issuer ID **NOTE:** This variable is required for all merchant-intiated transactions following the first one; upon sending the first transaction, the Issuer ID value is received in the transaction response and then used in subsequent transaction requests (Issuer ID does not apply for Discover or Union Pay).	*String*	15-character alpha-numeric variable length	`cof.SetIssuerId("VALUE_FOR_ISSUER_ID");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Indicator	*String*	1-character alphabetic	`cof.SetPaymentIndicator ("PAYMENT_INDICATOR_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File
Payment Information	*String*	1-character numeric	`cof.SetPaymentInformation ("PAYMENT_INFO_VALUE");` **NOTE:** For a list and explanation of the possible values to send for this variable, see Definitions of Request Fields – Credential on File

Sample Card Verification with Vault

```
namespace Moneris
{
using System;
public class TestResCardVerificationCC
{
public static void Main(string[] args)
{
string store_id = "store5";
string api_token = "yesguy";
string data_key = "V6F9PJKdXQj6vKiCMNrWbsyJ2";
string order_id = "Test_P_033333_6";
string cust_id = "Customer1";
string crypt = "7";
```

Sample Card Verification with Vault

```
string processing_country_code = "CA";
bool status_check = false;
/*************** Address Verification Service *********************/
AvsInfo avsCheck = new AvsInfo();
avsCheck.SetAvsStreetNumber("212");
avsCheck.SetAvsStreetName("Payton Street");
avsCheck.SetAvsZipCode("M1M1M1");

/****************** Card Validation Digits ************************/
CvdInfo cvdCheck = new CvdInfo();
cvdCheck.SetCvdIndicator("1");
cvdCheck.SetCvdValue("099");
/*************** Credential on File ************************************/
CofInfo cof = new CofInfo();
cof.SetPaymentIndicator("U");
cof.SetPaymentInformation("2");
cof.SetIssuerId("12345678901234");
ResCardVerificationCC rescardverify = new ResCardVerificationCC();
rescardverify.SetDataKey(data_key);
rescardverify.SetOrderId(order_id);
rescardverify.SetCustId(cust_id);
//rescardverify.SetExpDate("1612"); //for use with Temp Tokens only
rescardverify.SetCryptType(crypt);
rescardverify.SetAvsInfo(avsCheck);
rescardverify.SetCvdInfo(cvdCheck);
rescardverify.SetCofInfo(cof);

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(rescardverify);
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("IssuerId = " + receipt.GetIssuerId());
Console.ReadLine();
}
catch (Exception e)
```

Sample Card Verification with Vault

```
{
Console.WriteLine(e);
}
}
} // end TestResCardVerificationCC
}
```

Appendix A Definitions of Request Fields – Credential on File

Variable Name	Type	Limits	Description
Issuer ID **NOTE:** This variable is required for all merchant-intiated transactions following the first one; upon sending the first transaction, the Issuer ID value is received in the transaction response and then used in subsequent transaction requests (Issuer ID does not apply for Discover or Union Pay).	String	15-character alpha-numeric Variable length	Unique identifier for the cardholder's stored credentials Sent back in the response from the card brand when processing a Credential on File transaction If the cardholder's credentials are being stored for the first time, you must save the Issuer ID on your system to use in subsequent Credential on File transactions (applies to merchant-initiated transactions only)
Payment Indicator	String	1-character alphabetic	Indicates the intended or current use of the credentials Possible values for first transactions: C - unscheduled credential on file (first transaction only) R - recurring Possible values for subsequent transactions: R - recurring U - unscheduled merchant-initiated transaction Z - unscheduled cardholder-initiated transaction
Payment Information	String	1-character numeric	Describes whether the transaction is the first or subsequent in the series Possible values are: 0 - first transaction in a series (storing payment details provided by the cardholder)

Variable Name	Type	Limits	Description
			2 - subsequent transactions (using previously stored payment details)

Appendix B Definition of Request Fields – Recurring

Recurring Billing Info Object Request Fields

Variable and Field Name	Type and Limits	Description
Number of Recurs `num_recurs`	*String* numeric, 1-99	The number of times that the transaction must recur
Period `period`	*String* numeric, 1-999	Number of recur units that must pass between recurring billings
Start Date `start_date`	*String* YYYY/MM/DD	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 value of Start Now must be set to true
Start Now `start_now`	*String* true/false	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 When set to false, use Card Verification prior to sending the Purchase with Recur and Credential on File objects
Recurring Amount `recur_amount`	*String* 9-character decimal; Up to 6 digits (dollars) + decimal point + 2 digits (cents) after the decimal point	Amount of the recurring transaction This is the amount that will be billed on the Start Date and then billed repeatedly based on the interval defined by Period

Variable and Field Name	Type and Limits	Description
	EXAMPLE: 123456.78	and Recur Unit
Recur Unit `recur_unit`	*String* day, week, month or eom	Unit to be used as a basis for the interval Works in conjunction with Period to define the billing frequency Possible values are: day week month eom (end of month)

Appendix C Definition of Response Fields – Credential on File

Value	Type	Size	Get Method / Description
Issuer ID	String	15-character alpha-numeric	`receipt.GetIssuerId());` Returned when processing a transaction where the cardholder's credentials are being stored for the first time , and is used as the value for Issuer ID in the requests for subsequent transactions **NOTE:** For Discover and Union Pay transactions, Issuer ID is not returned in the response