iTransact Gateway API Documentation

iTransact

Gateway API Documentation

iTransact

Table of Contents

List of Figures

List of Tables

Chapter 1. API Overview

The gateway API features allow merchants or resellers to submit server to server requests to perform actions like transaction submission, recurring setup or to retreive transaction histories. It also allows resellers to submit XML transaction requests on behalf of their merchants using one set of credentials. This is useful for those that provide hosted shopping cart services, or want to provide their own merchant control panel.

Authentication

Authentication is handled using the combination of a username and a HMAC digest. A HMAC digest verifies both the authenticity and data integrity for a request payload. Our implementation is based on HMAC-SHA-1 using a 160 bit key. You can read more about HMAC at Wikipedia [http://en.wikipedia.org/wiki/HMAC] or for the adventurous the RFC [http://www.faqs.org/rfcs/rfc2104.html] is available. By using a signature based authentication scheme we can validate who generated an API request and ensure that the request was not altered using a man-in-the-middle attack [http://en.wikipedia.org/wiki/Man-in-the-middle_attack]

Obtaining an API Key

The reseller or merchant interfaces can be used to enable API access and setup and revoke API credentials. Resellers should click on the 'API Access' tab and follow the instructions. Merchants can setup their API features using the merchant settings tool.

Interfaces

The XML API provides interfaces that provide transaction requests and recurring transaction management features.

Pricing

There is no cost to a reseller for using the API features, however there is still a per merchant charge for XML transaction services. This means that every merchant that you want to process transactions for has to have XML processing enabled.

Chapter 2. Payload Signature Generation

Introduction

All the API features require a signature value to be calculated and passed through in the request. Each API interface will require this in a different input field so this chapter is just providing a general discussion of how to generate the signature for a given payload value. The payload format is also a feature of the API interface being used, so refer to the API interface specific sections discussing what the payload is.

Generating The PayloadSignature Value.

Overview

Every module that supports the API username/key authentication requires that some part of the request method is signed and sent as a separate parameter of the request. Once the request is received by our servers we create the same signature from the request and compare the values. This type of authentication is used by many other companies and the hash functions are free and widely available. Many languages come with these hash functions as standard libraries. We have included examples for many popular programming languages below.

Java HMAC Signature Example

The following Java example shows an example of how you can generate the payloadSignature value.

Table 2.1. Java HMAC Signature Example

```
import java.util.*;

import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import org.apache.commons.codec.binary.Base64;
public class HMACSignature {
  private static final String EXPECTED_SIGNATURE = "4U1OXfCzwOG1lMWHOgaIplUcWiE=";

  public static void main(String[] args) throws Exception {
    java.security.Security.addProvider(new com.sun.crypto.provider.SunJCE());
   SecretKeySpec hmac = new SecretKeySpec( "12345678901234567890".getBytes("ASCII"),
     "HmacSHA1" );
    Mac mac = Mac.getInstance( hmac.getAlgorithm() );

    mac.init( hmac );
    String sigLoad = "<RecurDetails><OperationXID>12345</OperationXID></RecurDetails>";
    byte[] digest = mac.doFinal(sigLoad.getBytes("UTF8"));

     String actualSignature = new String(Base64.encodeBase64(digest), "ASCII");

    if (EXPECTED_SIGNATURE.equals(actualSignature)) {
      System.out.println("Success!");
      System.out.println("Signature: " + actualSignature);
    }
    else {
      System.out.println("Failure!");
      System.out.println("Expected Signature: " + EXPECTED_SIGNATURE);
      System.out.println("  Actual Signature: " + actualSignature);
    }
  }
```

```
}
```

Perl HMAC Signature Example

Table 2.2. Perl HMAC Signature Example

```
my $EXPECTED_SIGNATURE = '4U1OXfCzwOG1lMWHOgaIplUcWiE=';

my $payload = "<RecurDetails><OperationXID>12345</OperationXID></RecurDetails>";

use Digest::HMAC_SHA1;
my $hmac = Digest::HMAC_SHA1->new("12345678901234567890");

$hmac->add($payload);
# The Perl Digest lib doesn't provide the trailing '=' character
my $actual_signature = $hmac->b64digest . '=';

if ($EXPECTED_SIGNATURE eq $actual_signature) {
  print "Success!\n";
  print "Signature: '$actual_signature'\n";
} else {
  print "Failure!\n";
  print "Expected Signature: '$EXPECTED_SIGNATURE'\n";
  print "  Actual Signature: '$actual_signature'\n";
}
```

Ruby HMAC Signature Example

Table 2.3. Ruby HMAC Signature Example

```
require 'openssl'
require 'base64'

EXPECTED_SIGNATURE = '4U1OXfCzwOG1lMWHOgaIplUcWiE='

payload = "<RecurDetails><OperationXID>12345</OperationXID></RecurDetails>"
key = "12345678901234567890"

digest=OpenSSL::HMAC.digest(OpenSSL::Digest::SHA1.new(key), key, payload)
actual_signature = Base64.b64encode(digest)

# For some reason, we end up with a new line character in the actual_signature...
actual_signature.chomp!

if EXPECTED_SIGNATURE.eql?(actual_signature)
  puts 'Success!'
  puts 'Signature: ' + actual_signature
else
  puts 'Failure!'
  puts 'Expected Signature: ' + EXPECTED_SIGNATURE
  puts '  Actual Signature: ' + actual_signature
end
```

ColdFusion HMAC Signature Example

Table 2.4. ColdFusion HMAC Signature Example

```
<cfset expectedSignature = "4U1OXfCzwOG1lMWHOgaIplUcWiE=">
<cfset payload = "<RecurDetails><OperationXID>12345</OperationXID></RecurDetails>">
<cfset key = "12345678901234567890">
<cfset generatedSignature = HMAC_SHA1(key = key, payload = payload)>

<cfoutput>
Payload:<br>
#payload#
<hr>
Expected Signature: #expectedSignature#<br>
Generated Signature: #generatedSignature#
</cfoutput>
```

PHP HMAC Signature Example

Table 2.5. PHP HMAC Signature Example

```
<?php

$expected = '4U1OXfCzwOG1lMWHOgaIplUcWiE=';

$payload = "<RecurDetails><OperationXID>12345</OperationXID></RecurDetails>";
$key = "12345678901234567890";

#Using built in PHP5 functions
$digest = hash_hmac('sha1', $payload, $key, true);
$actual_signature = base64_encode($digest);

#Using PEAR module
#require_once 'Crypt/HMAC.php';
#$hmac = new Crypt_HMAC($key,"sha1");
#$digest = pack("H40", $hmac->hash(trim($payload)));
#$actual_signature = base64_encode($digest);

if($expected == $actual_signature) {
  echo "Success!\n";
  echo "Signature: $actual_signature\n";
} else {
  echo "Failure!\n";
  echo "Expected Signature: $expected\n";
  echo "  Actual Signature: $actual_signature\n";
}

?>
```

.NET/C# HMAC Signature Example

Table 2.6. .NET/C# HMAC Signature Example

```
using System;
using System.Security.Cryptography;
using Text;
```

```
string expected_signature = "4U1OXfCzwOG1lMWHOgaIplUcWiE=";

string payload = "<RecurDetails><OperationXID>12345</OperationXID></RecurDetails>";

string key = "12345678901234567890";

Encoding encoding = new UTF8Encoding();
HMACSHA1 signature = new HMACSHA1(encoding.GetBytes(key));
string actual_signature = Convert.ToBase64String(
  signature.ComputeHash(encoding.GetBytes(
    payload.ToCharArray() ) )
);

if(expected_signature == actual_signature) {
  Console.WriteLine("Success!");
  Console.WriteLine("  Signature:  {0}", actual_signature);
} else {
  Console.WriteLine("Failure!");
  Console.WriteLine("Expected Signature:  {0}", expected_signature);
  Console.WriteLine("  Actual Signature:  {0}", actual_signature);
}
```

Testing Your Process

If you would like to test your signature process without having to hit our servers use the signatures generated in the examples above as a test base. If you use the same set of parameters your code should generate the same signature values that are shown in the examples.

Chapter 3. Interfaces

The XMLTrans2.cgi Module

The use of this module allows a reseller or merchant to submit Check and Card transactions as well as other requests such as recurring transaction modification.

https://secure.itransact.com/cgi-bin/rc/xmltrans2.cgi

Mime Type Information

The above cgi is accessed with an HTTP POST and requires a CONTENT_TYPE header to be specified. Either 'application/x-www-form-urlencoded' or 'text/xml' must be used. If 'application/x-www-form-urlencoded' is sent then the HTTP body must contain valid form markup. See WW3 Form Spec for details at w3.org. If 'text/xml' is used then the HTTP body should only contain the XML request. If the incorrect mime type is used, the following response will be sent back to your server:

Table 3.1. XMLTrans2.cgi MIME Type Error Example

```
<?xml version="1.0" standalone="yes"?>
  <GatewayFailureResponse>
    <Status>FAILED</Status>
      <ErrorCategory>REQUEST_FORMAT</ErrorCategory>
      <ErrorMessage>Unexpected mime type: </ErrorMessage>
  </GatewayFailureResponse>
```

Testing Tools for XMLTrans2.cgi Module

You can run all requests through xmltrans2 in test mode in order to help with the integration process. There are three different ways to run a XML transaction in test mode:

1. Enable the TestMode checkbox in your merchant account settings. Please remember this turns on TestMode for all transactions.
2. Setup the Test First Name field in your merchant account settings. Any xmltrans2 request that contains a BillingFirstName tag with this same value will be run as a test transaction. This is only applicable to a couple transactions such as CreditTransaction.
3. Send through a TestMode tag with the value TRUE. The TestMode tag which is in the TransactionControl structure always overrides the other test settings. This means that sending through TestMode equal to FALSE will cause a transaction to always be run as a live transaction.

API Payload

The API Payload for which a signature is generated is simply the whole XML structure of the action underneath the GatewayInterface tag. Let's look at the full XML structure for a request you might be using.

Table 3.2. XMLTrans2.cgi API Payload Full Request

```
<?xml version="1.0"?>
 <GatewayInterface>
   <APICredentials>
     <Username>username</Username>
     <PayloadSignature>signature</PayloadSignature>
     <TargetGateway>12345</TargetGateway>
   </APICredentials>
   <RecurDetails>
     <OperationXID>12345</OperationXID>
   </RecurDetails>
 </GatewayInterface>
```

For this request the action structure is the RecurDetails structure so you would calculate the payload based on the following text.

Table 3.3. XMLTrans2.cgi API Payload Full Request

```
   <RecurDetails>
     <OperationXID>12345</OperationXID>
   </RecurDetails>
```

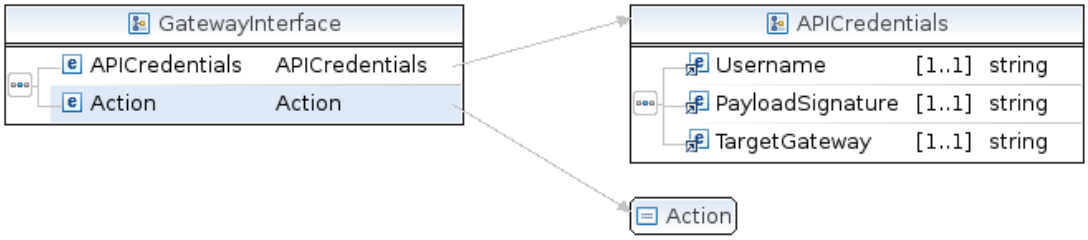
The payload passed into your signature generation routines should start with the first < character in <RecurDetails> and should end with the last > character in </RecurDetails>. We do not strip out space or carriage return/newline characters when validating the signature on our end. If you are having problems retrieving the XML sub structure without affecting the spacing that is present compared to how we receive it, you might want to disable XML pretty printing. This will ensure that there are no whitespace characters in your payload.

Request Structure

NOTE: The APICredential values are made up of the following:

- **Username** - This value is found (and can be reset) in the Account Settings of the Control Panel on an API enabled account.
- **PayloadSignature** - This is generated by signing the unique payload of each transaction witht the API key, which is found (and can be reset) in the Account Settings of the Control Panel on an API enabled account.
- **TargetGateway** - This is the five digit gateway ID number of the account. This field is designed for use with the iTransact Reseller API, but it is an optional field for the Gateway API.

Figure 3.1. GatewayInterface Diagram



Actions

Supported Actions

Table 3.4. Supported Actions

Action Name	Response	Description
AuthTransaction	TransactionResponse	Performs a Force, PreAuth or Sale request.
CreditTransaction	TransactionResponse	Performs a credit transaction without referencing a previous transaction.
PostAuthTransaction	TransactionResponse	This will generate a postauth (capture) for a previously run Preauth transaction.
RecurDetails	RecurDetailsResponse	this request will return the number of remaining repetitions, the recipe name and total for an existing recurring transaction.
RecurUpdate	RecurUpdateResponse	This request allows you to modify billing information, recurring status and customer info for an existing recurring transaction.
TranCredTransaction	TransactionResponse	Performs a credit transaction using the billing and customer data from a previous transaction.
TranForceTransaction	TransactionResponse	Performs a force transaction using the billing and customer data from a previous transaction.
TranRetryTransaction	TransactionResponse	Performs a sale transaction using the billing and customer data from a previous transaction. Previous transaction can be a sale, pre-auth or force.
TransactionStatus	TransactionResponse	Returns transaction information based on VendorData elements which can be used to determine the status of a transaction in an unknown state.
VoidTransaction	TransactionResponse	Voids an auth transaction that exists in the open batch. If a void cannot be performed because of

Action Name	Response	Description
		batch closure then a TranCredTransaction should be used instead.

AuthTransaction

The AuthTransaction can be used to perform three different authorization requests. A sale request is the default authorization type performed which is an authorization that will automatically be captured during the settlement process. A pre-auth request is performed when the PreAuth field is included. This performs an authorization that will not be captured during the settlement process until a post-auth transaction is run. A force transaction is run when an AuthCode field is included in the request. A force transaction can be run when you have been provided an authorization code over the phone by a processing network.

Figure 3.2. AuthTransaction Diagram

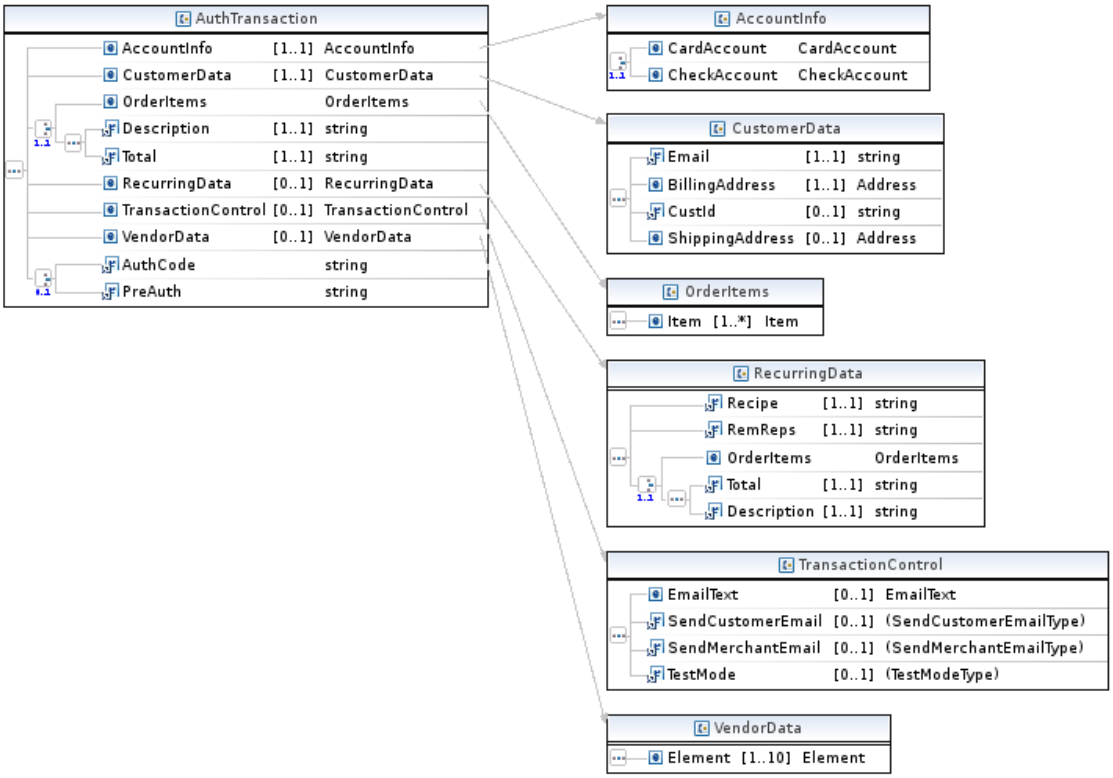


Table 3.5. AuthTransaction Fields

Field	Description	Data Type	Required
AccountInfo	Contains either card or check information	pc:AccountInfo	Yes
AuthCode	Initiates a Force transaction	xs:string	No (2)
CustomerData	Customer Information	pc:CustomerData	Yes

Field	Description	Data Type	Required
Description	Transaction Description	xs:string	Yes (1)
OrderItems	Transaction line items	pc:OrderItems	Yes (1)
PreAuth	Initiates a PreAuth transaction	xs:string	No (2)
RecurringData	Recurring transaction information	pc:RecurringData	No
Total	Transaction Total	xs:string	Yes (1)
TransactionControl	Transaction control parameters	pc:TransactionControl	No
VendorData	Merchant supplied meta data	pc:VendorData	No

(1) Either OrderItems or Total and Description must be provided

(2) Only one of PreAuth and AuthCode can be provided since these fields initiate different authorization types.

Table 3.6. XMLTrans2.cgi AuthTransaction Example

```
<?xml version="1.0"?>
<GatewayInterface>
   <APICredentials>
     <Username>username</Username>
     <PayloadSignature>signature</PayloadSignature>
     <TargetGateway>12345</TargetGateway>
   </APICredentials>
    <AuthTransaction>
        <!-- Optional.  Suplying AuthCode results in Force transaction -->
        <AuthCode>12345</AuthCode>
        <!-- Optional.  Supplying Preauth results in Pre-Auth transaction -->
        <Preauth/>
        <CustomerData>
            <Email>kevin@itransact.com</Email>
            <BillingAddress>
                <Address1>test</Address1>
                <Address2>test2</Address2>
                <FirstName>John</FirstName>
                <LastName>Smith</LastName>
                <City>Bountiful</City>
                <State>UT</State>
                <Zip>84032</Zip>
                <Country>USA</Country>
                <Phone>801-555-1212</Phone>
            </BillingAddress>
            <!-- Optional ShippingAddress -->
            <ShippingAddress>
                <Address1>test</Address1>
                <Address2>test2</Address2>
                <FirstName>John</FirstName>
                <LastName>Smith</LastName>
                <City>Bountiful</City>
                <State>UT</State>
                <Zip>84032</Zip>
                <Country>USA</Country>
                <Phone>801-555-1212</Phone>
            </ShippingAddress>
            <!-- Optional Customer ID -->
            <CustId>12345</CustId>
        </CustomerData>
        <!-- Can either supply OrderItems or Total and Description -->
        <OrderItems>
            <Item>
                <Description>test</Description>
                <Cost>10.00</Cost>
                <Qty>1</Qty>
```

```
            </Item>
        </OrderItems>
        <Total>10.00</Total>
        <Description>desc</Description>
        <AccountInfo>
            <!-- Can supply either CardAccount or CheckAccount -->
            <CardAccount>
                <!-- Supply AccountNumber, ExpirationMonth and ExpirationYear or TrackData -->
                <AccountNumber>5454545454545454</AccountNumber>
                <ExpirationMonth>01</ExpirationMonth>
                <ExpirationYear>2000</ExpirationYear>
                <!-- Optional -->
                <CVVNumber>123</CVVNumber>
                <!-- Track Data if running swipe transaction -->
                <TrackData>TRACK DATA</TrackData>
                <!-- Supply Ksn, Pin along with TrackData for Debit transactions -->
                <Ksn>12345</Ksn>
                <Pin>1234</Pin>
           <CheckAccount>
                <AccountNumber>123456</AccountNumber>
                <ABA>324377516</ABA>
                <!-- SecCode if required by processor -->
                <SecCode>PPD</SecCode>
                <!-- AccountSource if required by processor. Can be "checking" or "savings"-->
                <AccountSource>checking</AccountSource>
                <!-- AccountType if required by processor. Can be "personal" or "business"-->
                <AccountType>personal</AccountType>
           </CheckAccount>
        </AccountInfo>
        <!-- Optional  -->
        <RecurringData>
            <Recipe>text</Recipe>
            <RemReps>1</RemReps>
            <!-- Optional -- Can either supply OrderItems or Total and Description to have recurrin
            <OrderItems>
                <Item>
                    <Description>test</Description>
                    <Cost>10.00</Cost>
                    <Qty>1</Qty>
                </Item>
            </OrderItems>
            <Total>10.00</Total>
            <Description>desc</Description>
        </RecurringData>
        <!-- Optional  -->
        <TransactionControl>
            <SendCustomerEmail>TRUE</SendCustomerEmail> <!-- TRUE/FALSE -->
            <SendMerchantEmail>TRUE</SendMerchantEmail> <!-- TRUE/FALSE -->
            <TestMode>TRUE</TestMode> <!-- TRUE/FALSE -->
            <EmailText>
                <EmailTextItem>test1</EmailTextItem>
                <EmailTextItem>test1</EmailTextItem>
                <EmailTextItem>test1</EmailTextItem>
                <EmailTextItem>test1</EmailTextItem>
                <EmailTextItem>test1</EmailTextItem>
                <EmailTextItem>test1</EmailTextItem>
                <EmailTextItem>test1</EmailTextItem>
                <EmailTextItem>test1</EmailTextItem>
                <EmailTextItem>test1</EmailTextItem>
                <EmailTextItem>test1</EmailTextItem>
            </EmailText>
        </TransactionControl>
        <!-- Optional. This information will be saved on our
        servers and is available in the XML transaction report. This is
        useful if you want to save your own transaction meta-data with a
        transaction. -->
        <VendorData>
          <Element>
            <Name>repId</Name>
            <Value>1234567</Value>
          </Element>
        </VendorData>
    </AuthTransaction>
</GatewayInterface>
```

CreditTransaction

This will generate a credit/refund transaction for a transaction that was not originally processed through the gateway. This works for both credit cards and EFTs.

Figure 3.3. CreditTransaction Diagram

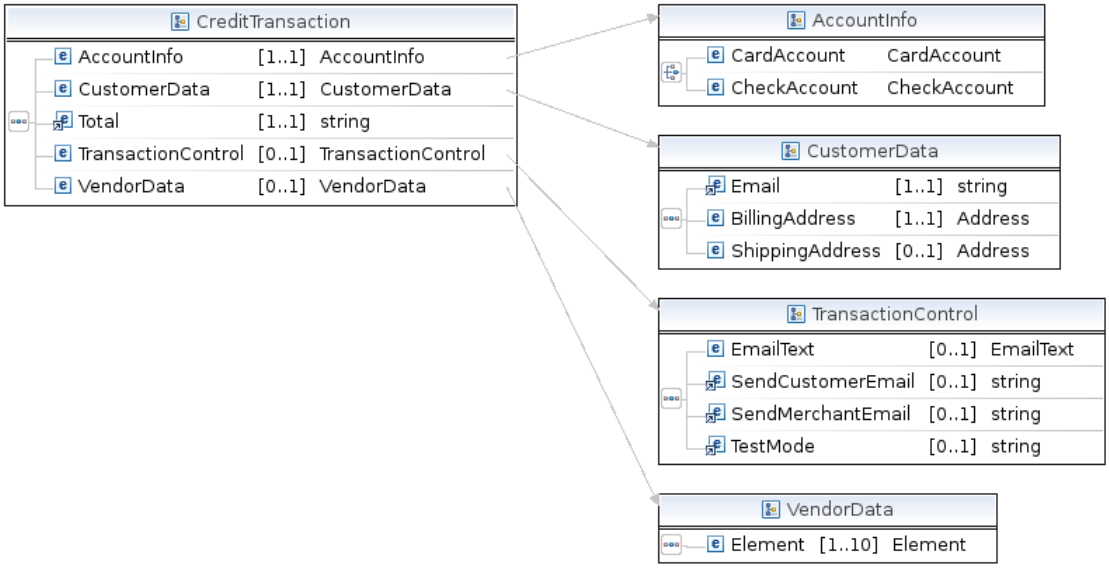


Table 3.7. CreditTransaction Fields

Field	Description	Data Type	Required
AccountInfo	Contains either card or check information	pc:AccountInfo	Yes
CustomerData	Customer Information	pc:CustomerData	Yes
Total	Transaction Total	xs:string	Yes
TransactionControl	Transaction control parameters	pc:TransactionControl	No
VendorData	Merchant supplied meta data	pc:VendorData	No

Table 3.8. XMLTrans2.cgi CreditTransaction Example

```
<?xml version="1.0"?>
 <GatewayInterface>
   <APICredentials>
     <Username>username</Username>
     <PayloadSignature>signature</PayloadSignature>
     <TargetGateway>12345</TargetGateway>
   </APICredentials>
   <CreditTransaction>
     <Total>5.00</Total>
```

```
      <CustomerData>
       <Email>demo@demo.com</Email>
       <!-- Optional --><CustId>12345</CustId>
        <BillingAddress>
         <Address1>test</Address1>
         <FirstName>John</FirstName>
         <LastName>Smith</LastName>
         <City>Bountiful</City>
         <State>UT</State>
         <Zip>84032</Zip>
         <Country>USA</Country>
         <Phone>801-555-1212</Phone>
        </BillingAddress>
        <!-- Optional -->
        <ShippingAddress>
         <Address1>test</Address1>
         <FirstName>John</FirstName>
         <LastName>Smith</LastName>
         <City>Bountiful</City>
         <State>UT</State>
         <Zip>84032</Zip>
         <Country>USA</Country>
        </ShippingAddress>
      </CustomerData>
       <AccountInfo>
        <!-- For Credit card transaction. -->
         <CardAccount>
          <AccountNumber>5454545454545454</AccountNumber>
          <ExpirationMonth>01</ExpirationMonth>
          <ExpirationYear>2000</ExpirationYear>
         </CardAccount>
        <!-- For EFT transactions. -->
         <CheckAccount>
          <AccountNumber>123456</AccountNumber>
          <ABA>124000054</ABA>
         </CheckAccount>
        <!-- or For NACHA transactions. -->
         <CheckAccount>
               <AccountNumber>123456</AccountNumber>
               <ABA>324377516</ABA>
               <!-- AccountSource if required by processor. Can be "checking" or "savings"-->
               <AccountSource>checking</AccountSource>
         </CheckAccount>
       </AccountInfo>
       <!-- All TransactionControl elements are optional including TransactionControl -->
       <TransactionControl>
        <SendCustomerEmail>TRUE</SendCustomerEmail> <!-- TRUE/FALSE -->
        <SendMerchantEmail>TRUE</SendMerchantEmail> <!-- TRUE/FALSE -->
        <TestMode>TRUE</TestMode> <!-- TRUE/FALSE -->
        <EmailText> <!-- Up to 10 EmailTextItem elements allowed -->
         <EmailTextItem>test1</EmailTextItem>
         <EmailTextItem>test1</EmailTextItem>
         <EmailTextItem>test1</EmailTextItem>
         <EmailTextItem>test1</EmailTextItem>
         <EmailTextItem>test1</EmailTextItem>
         <EmailTextItem>test1</EmailTextItem>
         <EmailTextItem>test1</EmailTextItem>
         <EmailTextItem>test1</EmailTextItem>
         <EmailTextItem>test1</EmailTextItem>
         <EmailTextItem>test1</EmailTextItem>
        </EmailText>
       </TransactionControl>
<!-- Optional.  This information will be saved on our servers and is
available in the XML transaction report.  This is useful if you want
to save your own transaction meta-data with a transaction.    -->
       <VendorData>
        <Element>
         <Name>repId</Name>
         <Value>1234567</Value>
        </Element>
       </VendorData>
   </CreditTransaction>
 </GatewayInterface>
```

PostAuthTransaction

This will generate a postauth (capture) for a previously run Preauth transaction. The OperationXID field should contain the XID for the original Preauth.

Figure 3.4. PostAuthTransaction Diagram

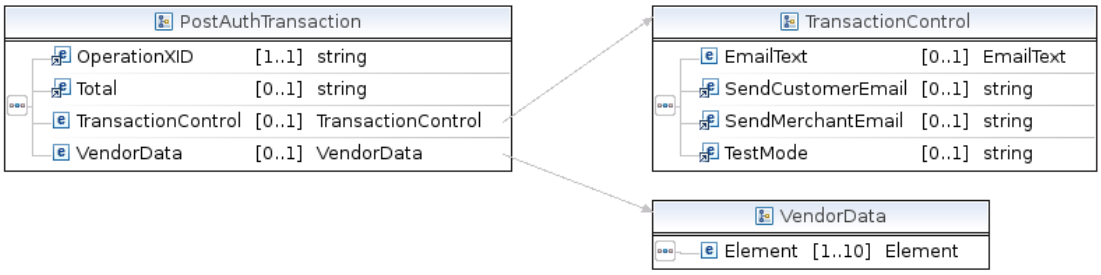


Table 3.9. PostAuthTransaction Fields

Field	Description	Data Type	Required
OperationXID	XID of PreAuth transaction	xs:string	Yes
Total	Total if different from PreAuth amount	xs:string	No
TransactionControl	Transaction control parameters	pc:TransactionControl	No
VendorData	Merchant supplied meta data	pc:VendorData	No

Table 3.10. XMLTrans2.cgi PostAuthTransaction Example

```
<?xml version="1.0"?>
 <GatewayInterface>
   <APICredentials>
     <Username>username</Username>
     <PayloadSignature>signature</PayloadSignature>
     <TargetGateway>12345</TargetGateway>
   </APICredentials>
  <PostAuthTransaction>
    <OperationXID>12345</OperationXID>
     <Total>5.00</Total>
     <!-- Optional - Will use original transaction amount if not specified here -->
     <!-- All TransactionControl elements are optional including TransactionControl -->
      <TransactionControl>
       <SendCustomerEmail>TRUE</SendCustomerEmail> <!-- TRUE/FALSE -->
       <SendMerchantEmail>TRUE</SendMerchantEmail> <!-- TRUE/FALSE -->
       <TestMode>TRUE</TestMode> <!-- TRUE/FALSE -->
       <EmailText> <!-- Up to 10 EmailTextItem elements allowed -->
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
```

```
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
       </EmailText>
      </TransactionControl>
<!-- Optional.  This information will be saved on our servers and is
available in the XML transaction report.  This is useful if you want
to save your own transaction meta-data with a transaction.    -->
       <VendorData>
        <Element>
         <Name>repId</Name>
         <Value>1234567</Value>
        </Element>
       </VendorData>
   </PostAuthTransaction>
 </GatewayInterface>
```

TranCredTransaction

This will generate a refund for a previously run transaction. The OperationXID field should contain the XID for the original transaction.

Figure 3.5. TranCredTransaction Diagram

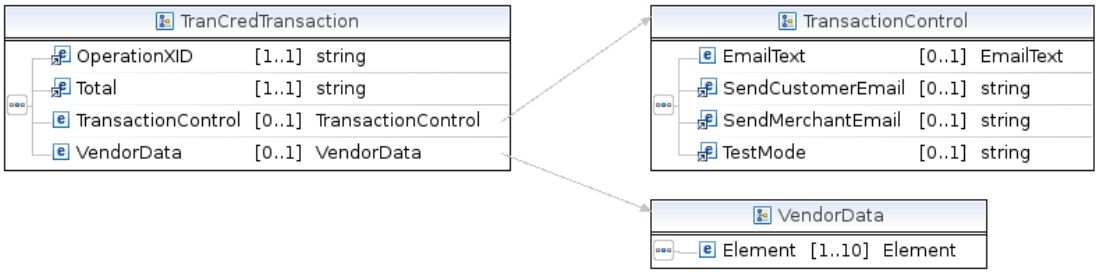


Table 3.11. TranCredTransaction Fields

Field	Description	Data Type	Required
OperationXID	XID of PreAuth transaction	xs:string	Yes
Total	Total dollar amount of transaction	xs:string	Yes
TransactionControl	Transaction control parameters	pc:TransactionControl	No
VendorData	Merchant supplied meta data	pc:VendorData	No

Table 3.12. XMLTrans2.cgi TranCredTransaction Example

```
<?xml version="1.0"?>
 <GatewayInterface>
   <APICredentials>
```

```
    <Username>username</Username>
    <PayloadSignature>signature</PayloadSignature>
    <TargetGateway>12345</TargetGateway>
   </APICredentials>
   <TranCredTransaction>
    <OperationXID>12345</OperationXID>
    <Total>5.00</Total>
    <!-- All TransactionControl elements are optional including TransactionControl -->
    <TransactionControl>
     <SendCustomerEmail>TRUE</SendCustomerEmail> <!-- TRUE/FALSE -->
     <SendMerchantEmail>TRUE</SendMerchantEmail> <!-- TRUE/FALSE -->
     <TestMode>TRUE</TestMode> <!-- TRUE/FALSE -->
     <EmailText> <!-- Up to 10 EmailTextItem elements allowed -->
      <EmailTextItem>test1</EmailTextItem>
      <EmailTextItem>test1</EmailTextItem>
      <EmailTextItem>test1</EmailTextItem>
      <EmailTextItem>test1</EmailTextItem>
      <EmailTextItem>test1</EmailTextItem>
      <EmailTextItem>test1</EmailTextItem>
      <EmailTextItem>test1</EmailTextItem>
      <EmailTextItem>test1</EmailTextItem>
      <EmailTextItem>test1</EmailTextItem>
      <EmailTextItem>test1</EmailTextItem>
     </EmailText>
    </TransactionControl>
<!-- Optional.  This information will be saved on our servers and is
available in the XML transaction report.  This is useful if you want
to save your own transaction meta-data with a transaction.    -->
     <VendorData>
      <Element>
       <Name>repId</Name>
       <Value>1234567</Value>
      </Element>
     </VendorData>
   </TranCredTransaction>
 </GatewayInterface>
```

TranForceTransaction

This will generate a Force (capture) for a previously failed transaction. Obtain a voice approval from the credit card processor's voice approval center and use that as value for the AuthCode. The OperationXID field should contain the XID for the original failed transaction.

Figure 3.6. TranForceTransaction Diagram

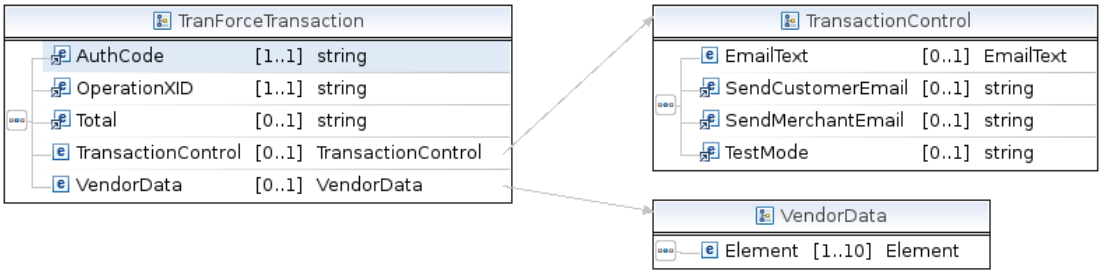


Table 3.13. TranForceTransaction Fields

Field	Description	Data Type	Required
AuthCode	Authorization code from processor	xs:string	Yes

Field	Description	Data Type	Required
OperationXID	XID of PreAuth transaction	xs:string	Yes
Total	Total dollar amount of transaction	xs:string	No (will use total from original transaction if not specified)
TransactionControl	Transaction control parameters	pc:TransactionControl	No
VendorData	Merchant supplied meta data	pc:VendorData	No

Table 3.14. XMLTrans2.cgi TranForceTransaction Example

```
<?xml version="1.0"?>
 <GatewayInterface>
   <APICredentials>
     <Username>username</Username>
     <PayloadSignature>signature</PayloadSignature>
     <TargetGateway>12345</TargetGateway>
   </APICredentials>
   <TranForceTransaction>
    <OperationXID>12345</OperationXID>
    <AuthCode>1234</AuthCode>
    <!-- Optional - Will use original transaction amount if not specified here -->
    <Total>5.00</Total>
     <TransactionControl>
      <SendCustomerEmail>TRUE</SendCustomerEmail> <!-- TRUE/FALSE -->
      <SendMerchantEmail>TRUE</SendMerchantEmail> <!-- TRUE/FALSE -->
      <TestMode>TRUE</TestMode> <!-- TRUE/FALSE -->
      <EmailText> <!-- Up to 10 EmailTextItem elements allowed -->
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
      </EmailText>
     </TransactionControl>
<!-- Optional.  This information will be saved on our servers and is
available in the XML transaction report.  This is useful if you want
to save your own transaction meta-data with a transaction.    -->
       <VendorData>
        <Element>
         <Name>repId</Name>
         <Value>1234567</Value>
        </Element>
       </VendorData>
   </TranForceTransaction>
 </GatewayInterface>
```

TranRetryTransaction

This will generate a sale transaction from a previously failed or successful transaction. The OperationXID field should contain the XID for the original transaction.

Figure 3.7. TranRetryTransaction Diagram

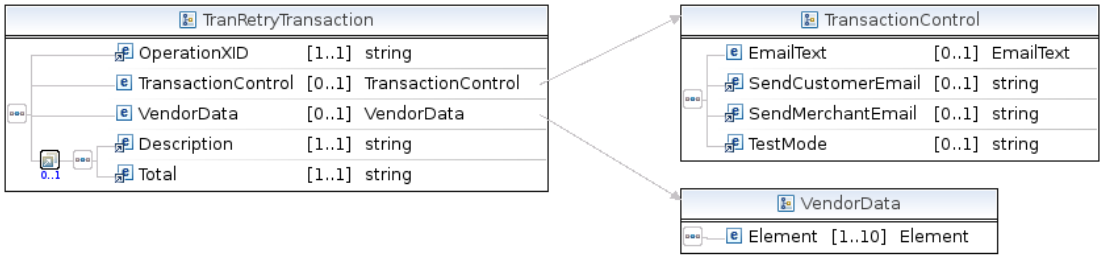


Table 3.15. TranRetryTransaction Fields

Field	Description	Data Type	Required
Description	Transaction description	xs:string	No (1)
OperationXID	XID of PreAuth transaction	xs:string	Yes
Total	Total dollar amount of transaction	xs:string	No (1)
TransactionControl	Transaction control parameters	pc:TransactionControl	No
VendorData	Merchant supplied meta data	pc:VendorData	No

(1) Total and Description are optional but if used both fields must be present

Table 3.16. XMLTrans2.cgi TranRetryTransaction Example

```
<?xml version="1.0"?>
 <GatewayInterface>
   <APICredentials>
     <Username>username</Username>
     <PayloadSignature>signature</PayloadSignature>
     <TargetGateway>12345</TargetGateway>
   </APICredentials>
   <TranRetryTransaction>
    <OperationXID>12345</OperationXID>
     <!-- Optional.  Description and Total are optional but if used
     both fields must be passed through -->
     <Description>1.00</Description>
     <Total>1.00</Total>
      <!-- All TransactionControl elements are optional including TransactionControl -->
      <TransactionControl>
       <SendCustomerEmail>TRUE</SendCustomerEmail> <!-- TRUE/FALSE -->
       <SendMerchantEmail>TRUE</SendMerchantEmail> <!-- TRUE/FALSE -->
       <TestMode>TRUE</TestMode> <!-- TRUE/FALSE -->
       <EmailText> <!-- Up to 10 EmailTextItem elements allowed -->
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
        <EmailTextItem>test1</EmailTextItem>
       </EmailText>
      </TransactionControl>
<!-- Optional.  This information will be saved on our servers and is
```

```
available in the XML transaction report.  This is useful if you want
to save your own transaction meta-data with a transaction.    -->
       <VendorData>
        <Element>
         <Name>repId</Name>
         <Value>1234567</Value>
        </Element>
       </VendorData>
   </TranRetryTransaction>
 </GatewayInterface>
```

VoidTransaction

This will void any sale, credit, or refund transaction if processed prior to the daily batch settlement.

Figure 3.8. VoidTransaction Diagram

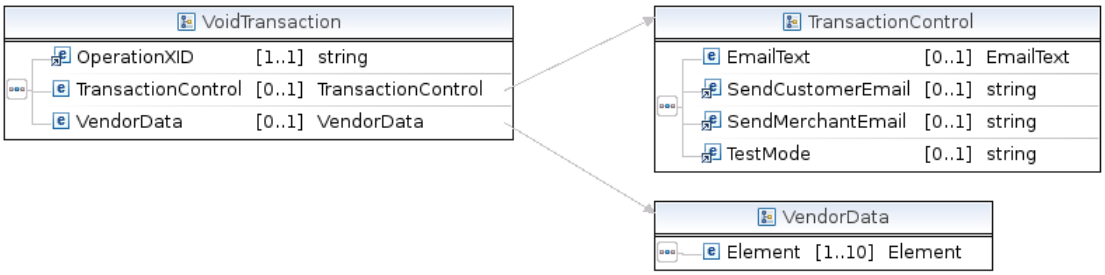


Table 3.17. VoidTransaction Fields

Field	Description	Data Type	Required
OperationXID	XID of PreAuth transaction	xs:string	Yes
TransactionControl	Transaction control parameters	pc:TransactionControl	No
VendorData	Merchant supplied meta data	pc:VendorData	No

Table 3.18. XMLTrans2.cgi VoidTransaction Example

```
<?xml version="1.0"?>
 <GatewayInterface>
   <APICredentials>
     <Username>username</Username>
     <PayloadSignature>signature</PayloadSignature>
     <TargetGateway>12345</TargetGateway>
   </APICredentials>
   <VoidTransaction>
    <OperationXID>12345</OperationXID>
     <!-- All TransactionControl elements are optional including TransactionControl -->
     <TransactionControl>
      <SendCustomerEmail>TRUE</SendCustomerEmail> <!-- TRUE/FALSE -->
      <SendMerchantEmail>TRUE</SendMerchantEmail> <!-- TRUE/FALSE -->
      <TestMode>TRUE</TestMode> <!-- TRUE/FALSE -->
```

```
      <EmailText> <!-- Up to 10 EmailTextItem elements allowed -->
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
       <EmailTextItem>test1</EmailTextItem>
      </EmailText>
     </TransactionControl>
<!-- Optional.  This information will be saved on our servers and is
available in the XML transaction report.  This is useful if you want
to save your own transaction meta-data with a transaction.    -->
      <VendorData>
       <Element>
        <Name>repId</Name>
        <Value>1234567</Value>
       </Element>
      </VendorData>
   </VoidTransaction>
 </GatewayInterface>
```

TransactionStatus

The TransactionStatus request provides a way to determine the status of a request that was interrupted for some reason. For a transaction where the response was lost the transaction xid would be unknown to the client. To be able to use this request you have to have passed through a unique transaction identifier of your own using the VendorData elements which can be passed through with the original request. Ideally you would pass through a unique ID with every request, although this request will do the lookup based on multiple VendorData elements. If more than one record is matched based on the passed VendorData the first matching transaction will be used to generate the response. The transaction response will contain a warning message in the WarningMessage field indicating that multiple transactions were matched in this case. Please note that this is NOT a search tool since it does not return multiple responses. This tool was also not written to automate the mass retrieval of past transaction history. Please use the Transaction Report API interface for that purpose.

Figure 3.9. TransactionStatus Diagram

Table 3.19. TransactionStatus Fields

Field	Description	Data Type	Required
VendorData	Merchant supplied meta data	pc:VendorData	Yes
TransactionControl	Transaction control parameters	pc:TransactionControl	No - Only the TestMode element applies.

Table 3.20. XMLTrans2.cgi TransactionStatus Example

```
<?xml version="1.0"?>
 <GatewayInterface>
   <APICredentials>
     <Username>username</Username>
     <PayloadSignature>signature</PayloadSignature>
     <TargetGateway>12345</TargetGateway>
   </APICredentials>
   <TransactionStatus>
    <VendorData>
     <Element>
      <Name>field1</Name>
      <Value>1234567</Value>
     </Element>
     <Element>
      <Name>field2</Name>
      <Value>1234567</Value>
     </Element>
    </VendorData>
   </TransactionStatus>
  <TransactionControl>
     <TestMode>FALSE</TestMode>
   </TransactionControl>
 </GatewayInterface>
```

TransactionResponse

All transaction requests including TransactionStatus return the TransactionResponse structure. The customer data fields are populated with data from the request designated by the OperationXID field in the request.

Figure 3.10. TransactionResult Diagram

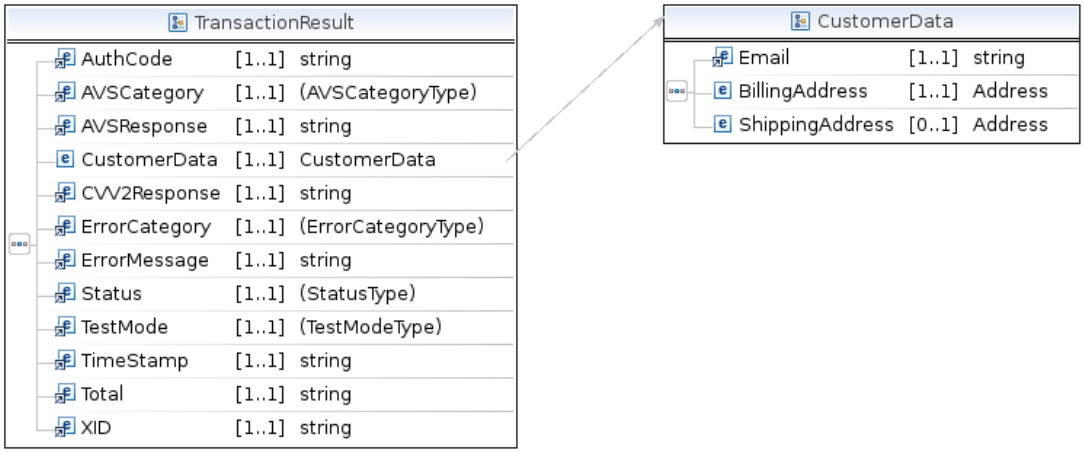


Table 3.21. TransactionResult Fields

Field	Description	Data Type	Required
AuthCode	Transaction authcode	xs:string	Yes
AVSCategory	Address verification response category	pc:AVSCategory	Yes
AVSResponse	Address verification re-	xs:string	Yes

Field	Description	Data Type	Required
	sponse		
CardName	Card name/type	pc:CardName	Yes
CustomerData	Customer data	pc:CustomerData	Yes
CVV2Response	Security code verification response	xs:string	Yes
ErrorCategory	Error category	pc:ErrorCategory	Yes
ErrorMessage	Error message	xs:string	Yes
Status	Transaction status	pc:Status	Yes
TestMode	Indicates if transaction was run in test mode	pc:TrueFalse	Yes
TimeStamp	Transaction timestamp	xs:string	Yes
Total	Transaction total	xs:string	Yes
WarningMessage	Warning Message	xs:string	Yes (1)
XID	Unique gateway transaction ID	xs:string	Yes

(1) This is currently only used with the TransactionStatus action in the case that more than one transaction is matched by the supplied VendorData elements.

Table 3.22. XMLTrans2.cgi TransactionResponse Example

```
<?xml version="1.0" standalone="yes"?>
 <GatewayInterface>
   <TransactionResponse>
    <TransactionResult>
     <Status>text</Status> <!-- Will be one of: error, fail, ok -->
      <ErrorCategory>text</ErrorCategory>
      <!-- ErrorCategory will be one of :
      AVS_FAILURE - Transaction will be automatically voided.
      CVV2_FAILURE - Transaction will be automatically voided.
      INTERNAL_ERROR - Something unexpected happened.
      PROCESSOR_ERROR - Something such as DECLINED, etc .
      PROCESSOR_FAIL -
      REQUEST_FORMAT - Request received has an invalid format.
      REQUEST_VALIDATION - XML content is invalid. -->
      <ErrorMessage>text</ErrorMessage>
      <!-- ErrorMessage could be anything. -->
      <AuthCode></AuthCode>
      <!-- Authorization code received from processing network. -->
      <AVSCategory></AVSCategory>
      <!-- AVSCategory will be one of :
      address - Address Matched
      address_postal - Address and postal patched
      address_zip5 - Address and five digit zip matched
      address_zip9 - Address and nine digit zip matched
      address_ok_postal_format_error - Address matched, postal format error
      global_non_participant - International with no AVS support
      international_address_not_verified - International with no AVS support
      no_match - No address or postal match
      no_response - No response
      not_allowed - Not allowed
      postal - Postal match
      postal_ok_address_format_error - Postal matched, address format error
      service_not_supported - AVS service not supported for card
      unavailable - AVS service unavailable.
      zip5 - Five digit zip matched
      zip9 - Nine digit zip matched -->
      <AVSResponse></AVSResponse>
      <!-- AVSResponse is actual AVS response received from the processing network. -->
```

```
      <CardName></CardName>
      <!-- CardName will be one of:
      American Express
      Australian Bankcard
      Discover
      Diners Club/Carte Blanche
      enRoute
      Japanese Credit Bureau
      MasterCard
      Visa
      -->
      <CVV2Response></CVV2Response>
      <!-- CVV2Response is actual CVV response received from the processing network. -->
      <TimeStamp></TimeStamp>
      <TestMode>FALSE</TestMode>
      <!-- TestMode indicates the test status of your gateway account. TRUE/FALSE -->
      <Total></Total>
      <XID></XID>
       <CustomerData>
        <BillingAddress>
         <Address1 />
         <City></City>
         <FirstName></FirstName>
         <LastName></LastName>
         <State></State>
         <Zip></Zip>
         <Country></Country>
         <Phone></Phone>
        </BillingAddress>
        <ShippingAddress>
         <Address1></Address1>
         <City></City>
         <FirstName></FirstName>
         <LastName></LastName>
         <State></State>
         <Zip></Zip>
         <Country></Country>
         <Phone></Phone>
        </ShippingAddress>
       </CustomerData>
    </TransactionResult>
   </TransactionResponse>
</GatewayInterface>
```

RecurUpdate

This request allows you to modify the transaction information and recurring commands for a recurring transaction.

Figure 3.11. RecurUpdate Diagram

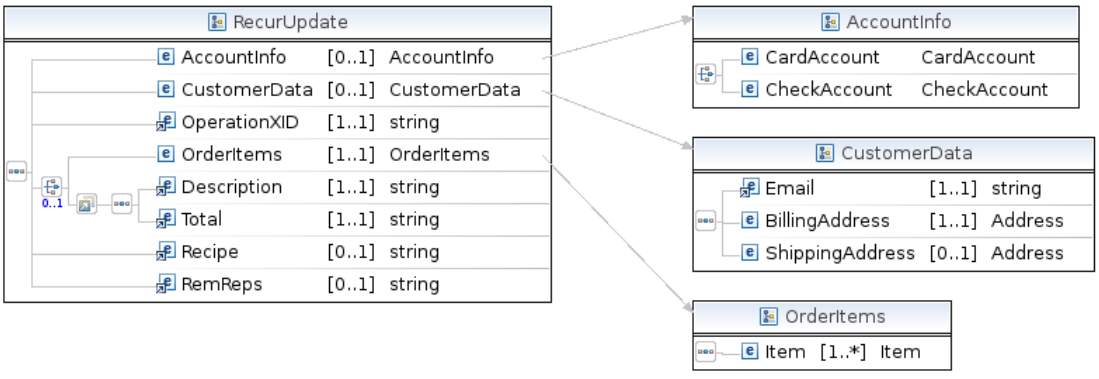


Table 3.23. RecurUpdate Fields

Field	Description	Data Type	Required
AccountInfo	Contains either card or check information	pc:AccountInfo	No (2)
CustomerData	Customer Information	pc:CustomerData	No (2,3)
Description	Transaction Description	xs:string	No (1,2)
OperationXID	XID of PreAuth transaction	xs:string	Yes
OrderItems	Transaction line items	pc:OrderItems	No (1,2)
Recipe	Recipe name	xs:string	No (2)
RemReps	Remaining recurring repetitions	xs:string	No (2)
Total	Transaction Total	xs:string	No (1,2)

(1) Only one of OrderItems or Total and Description must be provided although all are optional.

(2) Other than OperationXID, all of the child elements of RecurUpdate are individually optional but you must pass one of Recipe, RemReps, CustomerData, OrderItems or Total.

(3) The CustomerData element in RecurUpdate only requires one of the following sub elements to be provided: AccountInfo, Email, BillingAddress, CustId or ShippingAddress

Table 3.24. XMLTrans2.cgi RecurUpdate Example

```
<?xml version="1.0"?>
 <GatewayInterface>
  <VendorIdentification>
   <VendorId>1</VendorId>
   <VendorPassword>test</VendorPassword>
   <HomePage>text</HomePage>
  </VendorIdentification>
   <!-- Other than OperationXID, all of the child elements of RecurUpdate
   are individually optional but you must pass -->
   <!-- one of Recipe, RemReps, CustomerData, OrderItems or Total -->
   <RecurUpdate>
    <OperationXID>12345</OperationXID>
     <!-- Optional.-->
     <RemReps>123</RemReps>
     <!-- Optional.-->
     <Recipe>Recipe Name</Recipe>
     <!-- Optional.  Will update customer info tied to recurring transaction if passed-->
      <CustomerData>
       <Email>demo@demo.com</Email>
       <CustId>12345</CustId> <!-- Optional -->
        <BillingAddress>
         <Address1>test</Address1>
         <FirstName>John</FirstName>
         <LastName>Smith</LastName>
         <City>Bountiful</City>
         <State>UT</State>
         <Zip>84032</Zip>
         <Country>USA</Country>
         <Phone>801-555-1212</Phone>
        </BillingAddress>
        <!-- Optional -->
        <ShippingAddress>
         <Address1>test</Address1>
         <FirstName>John</FirstName>
         <LastName>Smith</LastName>
         <City>Bountiful</City>
```

```
        <State>UT</State>
        <Zip>84032</Zip>
        <Country>USA</Country>
       </ShippingAddress>
     </CustomerData>
      <!-- Optional.  Will update customer info tied to recurring transaction if passed-->
       <AccountInfo>
        <!-- For Credit card transaction. -->
        <CardAccount>
         <AccountNumber>5454545454545454</AccountNumber>
         <ExpirationMonth>01</ExpirationMonth>
         <ExpirationYear>2000</ExpirationYear>
         <CVVNumber>123</CVVNumber><!-- Optional -->
        </CardAccount>
        <!-- For EFT transactions. -->
        <CheckAccount>
         <AccountNumber>123456</AccountNumber>
         <ABA>324377516</ABA>
        </CheckAccount>
       </AccountInfo>
        <!-- Only one of OrderItems or Total elements may be passed in but neither are required --
         <OrderItems>
          <Item>
           <Description>item1</Description>
           <Cost>5</Cost>
           <Qty>1</Qty>
          </Item>
         </OrderItems>
        <!-- To use the Total element the original transaction
        can only have one item associated with it  -->
        <Total>5.00</Total>
    </RecurUpdate>
</GatewayInterface>
```

RecurUpdateResponse

This request will return the following response:

Table 3.25. XMLTrans2.cgi RecurUpdateResponse Example

```
<?xml version="1.0" standalone="yes"?>
 <GatewayInterface>
  <RecurUpdateResponse>
   <Status>ok</Status>
   <ErrorCategory></ErrorCategory>
   <ErrorMessage></ErrorMessage>
   <TimeStamp>20060621154341</TimeStamp>
   <TestMode>FALSE</TestMode> <!-- TRUE/FALSE -->
    <RecurDetails>
     <RemReps>10</RemReps>
     <RecipeName>daily</RecipeName>
     <RecurTotal>1.00</RecurTotal>
    </RecurDetails>
  </RecurUpdateResponse>
 </GatewayInterface>
```

RecurDetails

This request allows you to query for details on an existing recurring transaction. This includes information about the recurring details as well as information that helps determine if the credit card tied to the recurring transaction is expired or set to expire soon.

Table 3.26. XMLTrans2.cgi RecurDetails Example

```
<?xml version="1.0"?>
 <GatewayInterface>
   <APICredentials>
     <Username>username</Username>
     <PayloadSignature>signature</PayloadSignature>
     <TargetGateway>12345</TargetGateway>
   </APICredentials>
   <RecurDetails>
     <OperationXID>12345</OperationXID>
   </RecurDetails>
 </GatewayInterface>
```

RecurDetailsResponse

This request will return the following response:

Table 3.27. XMLTrans2.cgi RecurDetailsResponse Example

```
<?xml version="1.0" standalone="yes"?>
 <GatewayInterface>
  <RecurDetailsResponse>
    <Status>ok</Status>
    <ErrorCategory></ErrorCategory>
    <ErrorMessage></ErrorMessage>
    <TimeStamp>20060621154341</TimeStamp>
    <TestMode>FALSE</TestMode> <!-- TRUE/FALSE -->
     <RecurDetails>
      <CardExpired>FALSE</CardExpired> <!-- TRUE/FALSE -->
      <CardExpiresWithinThirty>FALSE</CardExpiresWithinThirty> <!-- TRUE/FALSE -->
      <CardLastFour>1234</CardLastFour>
      <CardName>Visa</CardName>
      <RemReps>10</RemReps>
      <RecipeName>daily</RecipeName>
      <RecurTotal>1.00</RecurTotal>
     </RecurDetails>
  </RecurDetailsResponse>
 </GatewayInterface>
```

Transaction Report API

API Access to Reports

The same tool that generates the merchant reports available from the merchant control panel may also be accessed externally. This allows you to automate the process of downloading either CSV or XML transaction reports.

Request Parameters

The integration with the reporting application is done through a form post. There are seven required parameters that must be included with every request and there are also optional parameters that let you narrow down your result set. The URL of the report application is https://secure.itransact.com/jfe/translist/list.do.

Required Parameters

The following request parameters must be submitted with every request

- **apiUsername**

  This is the API username that has been assigned to you

- **outputFormat** - Report Format. Valid input values are:

  list-csv

  list-xml

- **payloadSignature**

  This is the payloadSignature that you have generated using any other request parameters. See Section 2.3 to see the specifics of how this is generated.

- **transDateFirst**

  This is the start date for the set of transactions you are trying to access. The format is yyyy/MM/dd

- **transDateLast**

  This is the end date for the set of transactions you are trying to access. The format is yyyy/MM/dd

- **transTimeFirs**t

  This is the start time for the set of transactions you are trying to access. The format is HH:mm

- **transTimeLast**

  This is the end time for the set of transactions you are trying to access. The format is HH:mm

Optional Parameters

The following parameters are option search parameters. These search parameters do not result in full text searches, so they will only match records that have identical field values.

- **action** - Transaction Type. The valid action inputs are:

  order - Order which includes Sale, PreAuth and Force transactions. Use the subaction input to narrow down the result to a specific type.

  credit - Credit

  void - Void

  postauth - PostAuth

- **auth** - Authorization Code

- **avsCategory** - AVS Response Category

- **avsResponse** - AVS Response Code

- **batchNumber** - Batch Number
- **card** - Card Type. This input field can contain a single or comma separated list of card types to include in the results. The valid card types are:

  1 - American Express

  3 - Discover

  4 - Diners/Carte Blanche

  7 - Mastercard

  8 - Visa
- **cvv2Response** - CVV Response Code
- **email** - Customer email
- **extUserid** - Merchant supplied user ID (the cust_id field)
- **haddr** - Billing Address
- **hcity** - Billing City
- **hctry** - Billing Country
- **hfname** - Billing First Name
- **hlname** - Billing Last Name
- **hstate** - Billing State
- **hzip** - Billing Postal Code
- **instr** - Payment Instrument. Valid instr inputs are:

  cc - Credit Card

  ck - Check

  eft - EFT
- **ip** - Transaction IP Address Source
- **lastFour** - Last four digits of credit card number
- **parentXid** - Parent transaction ID
- **phone** - Customer phone
- **recurRecipe** - Recurring recipe name
- **recurStatus** - Include only recurring transactions with this status. Status values are the same as the status input parameter.
- **recurXid** - Include only recurring transactions that have recurXid as their parent XID.
- **saddr** - Shipping Address

- **scity** - Shipping City
- **sctry** - Shipping Country
- **sfname** - Shipping First Name
- **slname** - Shipping Last Name
- **sstate** - Shipping State
- **szip** - Shipping Postal Code
- **status** - Transaction Status. Valid status values are:

  ok - Transaction Completed Successfully

  error - The transaction was refused by the processing network

  fail - The transaction failed for an unexpected reason

  avs_fail - The transactions AVS response did not satisfy the merchants AVS requirements at the time the transaction was run

  unknown - An unknown error occurred

  begun - The transaction did not complete which usually indicates an internal error.
- **subAction** - Used to narrow down the result set to one type of action. Valid input values are:

  force - Force Transaction

  preauth - Preauth Transaction

  sale - Sale Transaction
- **total** - Transaction total in the format d.dd.
- **transSource** - Transaction source. Valid input values are:

  av - AutoVoid

  html - Web Form

  phone - Phone

  recur - Recurring

  session - Virtual Terminal

  xml - XML
- **xidFirst** - This is the opening parameter that allows you to pull a range of specified transaction IDs.
- **xidLast** - This is the closing parameter that allows you to pull a range of specified transaction IDs.

API Payload

Overview

To generate the payloadSignature input parameter you are going to generate a string that looks like a HTTP parameter set and sign this string using HMAC-SHA1. You include all the parameters you are going to pass through except for payloadSignature. This parameter set string also needs to be generated in alphabetical order. The HMAC-SHA1 algorithm generates a result that is not web friendly so you have to then Base64 encode it so it can be sent through the request.

Signature Process

For this section we are going to assume that we are going to issue a request to search for all credit card transactions between 10/1/2007 and 10/31/2007. We have been issued API credentials with a username of test with a key of 12345678901234567890.

Assemble Our Request Parameters

To emphasize the point that we are going to sort input parameters alphabetically the 'instr' field is listed out of order

- apiUsername - test
- outputFormat - list-xml
- transDateFirst - 2007/10/01
- transDateLast - 2007/10/31
- transTimeFirst - 00:00
- transTimeLast - 23:59
- instr - cc

Generate The Payload

Note that although we are generating something that looks like part of a HTTP request URL we don't need to URI encode it. The example below shows up in two lines in this document for formatting purposes but would actually be a string without any carriage return or newline characters.

```
apiUsername=test&instr=cc&outputFormat=list-xml&transDateFirst=2007/10/01&
transDateLast=2007/10/31&transTimeFirst=00:00&transTimeLast=23:59
```

Sign The Payload

After Base64 encoding the output of our HMAC-SHA1 library we end up with a signature of 'cyeL36oyyManvmmJWBHZJv3Z1bE='

API Responses

XML Response Structure

If 'list-xml' was specified for the outputFormat parameter a XML response structure is returned. The structure includes both information about the values of the inputs and the individual transaction records. The example provided below contains all the response elements that might be returned in the response.

Table 3.28. Transaction Report API XML Response Structure

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<transactionSearchResult>
  <query>
    <action />
    <auth />
    <avsCategory />
    <avsResponse />
    <batchNumber />
    <card />
    <cvv2Response />
    <email />
    <custId />
    <haddr />
    <hcity />
    <hctry />
    <hfname />
    <hlname />
    <hstate />
    <hzip />
    <instr />
    <ip />
    <lastFour />
    <merName>Merchant Name</merName>
    <parentXid />
    <phone />
    <recurRecipe />
    <recurStatus />
    <recurXid />
    <status />
    <subAction />
    <total />
    <transDateFirst>2007/11/1</transDateFirst>
    <transDateLast>2007/11/2</transDateLast>
    <transTimeFirst />
    <transTimeLast />
    <xidFirst />
    <xidLast />
  </query>
  <transactions>
    <transaction>
      <action>order</action>
      <auth>1234567</auth>
      <avsCategory>zip5</avsCategory>
      <avsResponse>Z</avsResponse>
      <batchNumber>7</batchNumber>
      <card>American Express</card>
      <childXids>
        <childXid>123456789</childXid>
      </childXids>
      <cvv2Response>M</cvv2Response>
      <email>test@test.com</email>
      <custId>12345</custId>
      <haddr>123 Street Name</haddr>
      <hcity>City</hcity>
      <hctry>USA</hctry>
      <hfname>First</hfname>
      <hlname>Last</hlname>
      <hstate>UT</hstate>
      <hzip>84010</hzip>
      <instr>cc</instr>
      <ip>11.11.11.11</ip>
      <items>
        <item index="0">
          <itemDetail code="cost">1.99</itemDetail>
          <itemDetail code="desc">Item Description</itemDetail>
          <itemDetail code="qty">1</itemDetail>
        </item>
      </items>
      <merchantData>
        <value name="orderNumber">123456789</value>
      </merchantData>
      <lastFour>2000</lastFour>
      <merName>Merchant Name</merName>
      <parentXid>123456789</parentXid>
```

```
      <phone>801-555-1212</phone>
      <recurRecipe>recipe name</recurRecipe>
      <recurStatus>ok</recurStatus>
      <recurXid>123456789</recurXid>
      <saddr>123 Street Name</saddr>
      <scity>City</scity>
      <sctry>USA</sctry>
      <sfname>First</sfname>
      <slname>Last</slname>
      <sstate>UT</sstate>
      <szip>84010</szip>
      <status>ok</status>
      <subAction>sale</subAction>
      <total>1.99</total>
      <transDate>11/1/2007 12:00:00</transDate>
      <transSrc>session</transSrc>
      <xid>123456789</xid>
    </transaction>
    <transaction>
      ....
    </transaction>
  </transactions>
</transactionSearchResult>
```

CSV Response Structure

If 'list-csv' was specified for the outputFormat parameter a comma delimited response structure is returned. This report has one line per transaction but the number of columns is variable depending on the transaction data in the report. There are two data elements that cause the number of columns to vary.

- if there are any child transactions each child transaction XID is added as a separate column. If there are no child transactions there are no columns added to the report.
- Since each line contains all the order items, the number of order item columns in a report is based on the transaction in the set with the largest number of order items.

Each line can be viewed as having five different sections.

Header Set 1

This is a fixed set of header values.

Table 3.29. Transaction Report API CS Response - Header Structure 1

```
action,auth,avsCategory,avsResponse,batchNumber,card
```

Header Set 2

This is a variable set of header values based on whether any child aids exist and if so on how many. For each child CID associated with a transaction a column is added. For any given report the number of childed columns is equal to the maximum number of child aids associated with any transaction in the report.

Table 3.30. Transaction Report API CS Response - Header Structure 2

```
childXid_0, childXid_1, childXid_2
```

Header Set 3

This is a fixed set of header values.

Table 3.31. Transaction Report API CS Response - Header Structure 3

```
cvv2Response,custId, email,haddr,hcity,hctry,hfname,hlname,hstate,hzip,instr,ip
```

Header Set 4

This set of headers consists of the order items for the request. Each order item has a minimum of three values associated with it (cost, qty, Des), although there can be more. Each value generates two columns, one for the value code and one for the actual value. The column headers in this section are zero based meaning your order item numbers are not preserved in the report. If you pass through items 1 and 2 you will still have column header names like 'item_0_code_0'.

Let's consider an example transaction with two order items:

Item Number	Cost	Qty	Description
I	$1.00	1	Item 1
2	$2.00	2	Item 2, with comma

Since we have 6 order item attributes this will result in 12 columns being generated. The output that would be generated for this example transaction is shown below.

Table 3.32. Transaction Report API CS Response - Header Structure 4

```
item_0_code_0,item_0_value_0,item_0_code_1,item_0_value_1,item_0_code_2,item_0_value_2,
item_1_code_0,item_1_value_0,item_1_code_1,item_1_value_1,item_1_code_2,item_1_value_2
cost,1.00,desc,Item 1,qty,1,cost,2.00,desc,Item2&amp;#44; with comma,qty,2
```

Header Set 5

This is a fixed set of header values.

Table 3.33. Transaction Report API CS Response - Header Structure 5

```
lastFour,merName,mid,parentXid,phone,recurRecipe,recurStatus,recurXid,saddr,scity,
sctry,sfname,slname,sstate,szip,status,subAction,total,transDate,transSrc,xid
```

XML Schema Reference

Schema Types

AccountInfo

Figure 3.12. AccountInfo Type

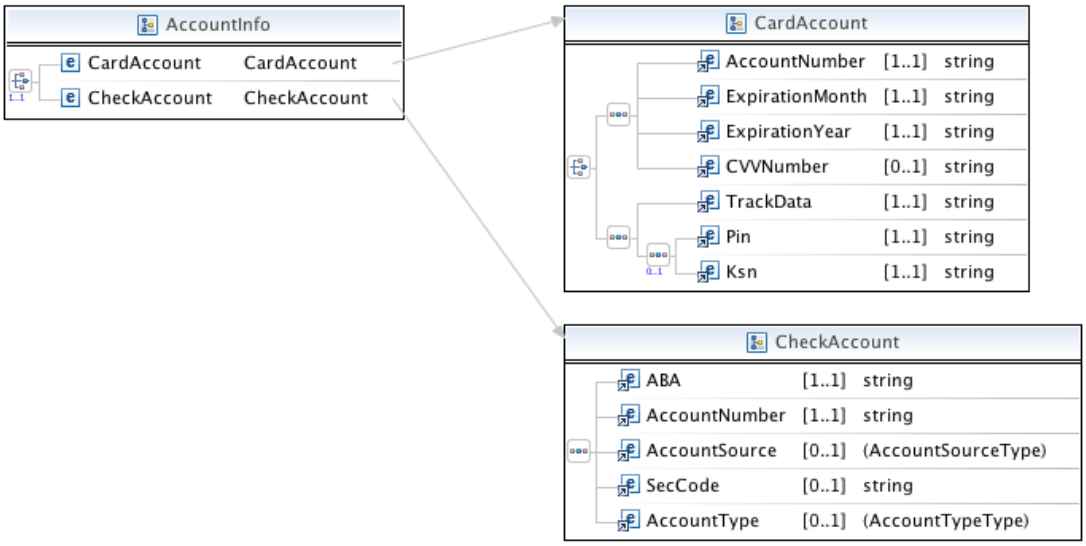


Table 3.34. AccountInfo Fields

Element	Description	Data Type	Required
CardAccount	Card Account Information	pc:CardAccount	CardAccount or CheckAccount Required
CheckAccount	Check Account Information	pc:CheckAccount	CardAccount or CheckAccount Required

Address

Figure 3.13. Address Type

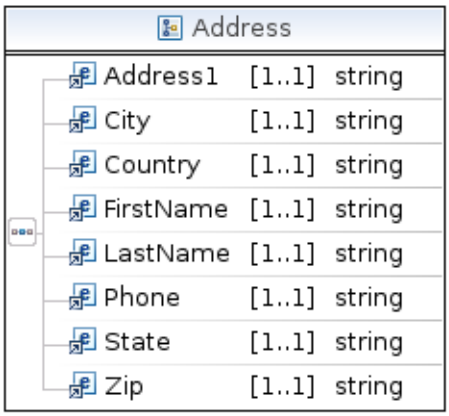


Table 3.35. Address Fields

Element	Description	Data Type	Required
Username	Assigned API Username	xs:string	Yes
TargetGateway	Gateway ID for which request is being run	xs:string	Yes
PayloadSignature	Calculated Signature see Chapter 2	xs:string	Yes

APICredentials

Figure 3.14. APICredentials Type

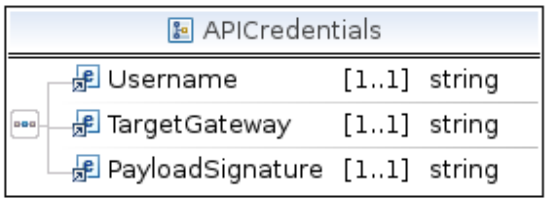


Table 3.36. APICredentials Fields

Element	Description	Data Type	Required
Username	Assigned API Username	xs:string	Yes
TargetGateway	Gateway ID for which request is being run	xs:string	Yes
PayloadSignature	Calculated Signature see Chapter 2	xs:string	Yes

CardAccount

Figure 3.15. CardAccount Type

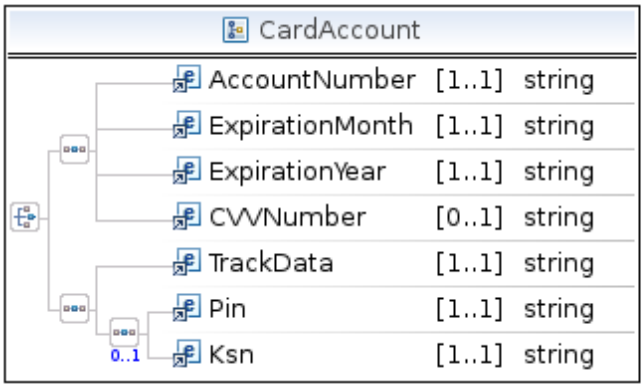


Table 3.37. CardAccount Fields

Element	Description	Data Type	Required
AccountNumber	Assigned API Username	xs:string	Yes if using plain text card
ExpirationMonth	Card Expiration Month	xs:string	Yes if using plain text card
ExpirationYear	Card Expiration Year	xs:string	Yes if using plain text card
CVVNumber	CVV Verification Number	xs:string	No
TrackData	Track Data retrieved using mag stripe reader (2)	xs:string	Yes if using swipe card
Pin	Encrypted PIN Data	xs:string	No (1)
Ksn	Encrypted KSN Data	xs:string	No (1)
(1) Pin and Ksn are both required for Debit transactions (2) This input field can consist of track1, track2, or track1 and track2 data combined. If track1 and track2 are sent, you must include the start and end sentinel characters for both tracks. If only track1 or track2 data is being passed, the sentinel characters are optional. The formats of track1 and track2 are defined by ISO standards. You can lean more about these formats at ht-			

Element	Description	Data Type	Required
tp://en.wikipedia.org/wiki/ISO/IEC_7813			

CheckAccount

Figure 3.16. CheckAccount Type

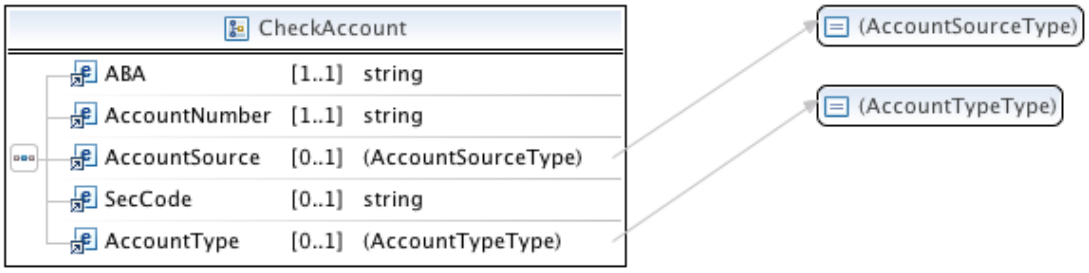


Table 3.38. CheckAccount Fields

Element	Description	Data Type	Required
ABA	9 Digit ABA Number	xs:string	Yes
AccountNumber	Checking Account Number	xs:string	Yes
SecCode	Standard Entry Class	xs:string	Conditionally Required

CustomerData

Figure 3.17. CustomerData Type

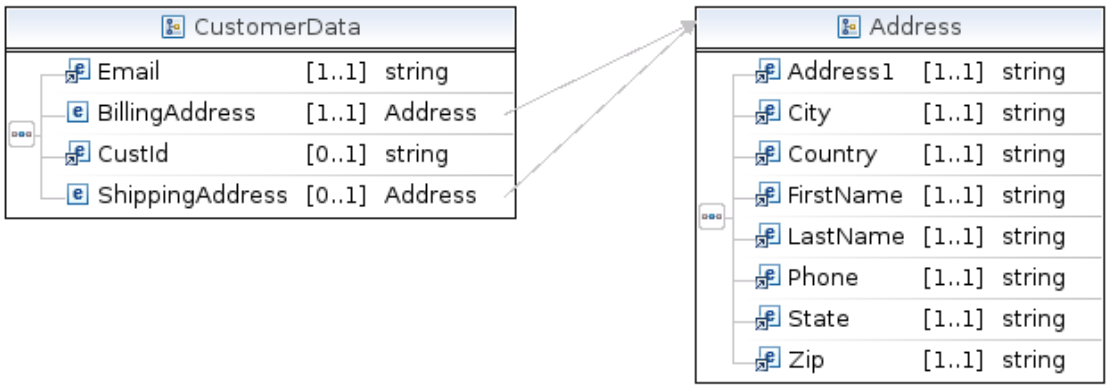


Table 3.39. CardAccount Fields

Element	Description	Data Type	Required
BillingAddress	Billing Information	pc:Address	Yes
CustId	Merchant's customer ID	xs:string	No
Email	Customer Email Address	xs:string	Yes
ShippingAddress	Shipping Information	pc:Address	No

Element

Figure 3.18. Element Type

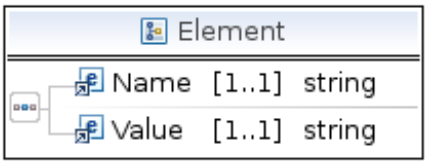


Table 3.40. Element Fields

Element	Description	Data Type	Required
Name	Meta Item Name	xs:string	Yes
Value	Meta Item Value	xs:string	Yes

EmailText

Figure 3.19. EmailText Type

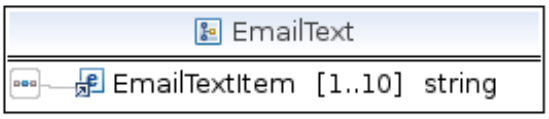


Table 3.41. EmailText Fields

Element	Description	Data Type	Required
EmailTextItem	Line of email text to be included in customer emails	xs:string	Yes (at least one, and up to 10)

Item

Figure 3.20. Item Type

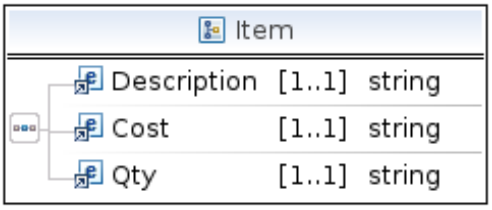


Table 3.42. Item Fields

Element	Description	Data Type	Required
Description	Item Description	xs:string	Yes
Cost	Item Cost	xs:string	Yes
Qty	Item Qty	xs:string	Yes

OrderItems

Figure 3.21. OrderItems Type

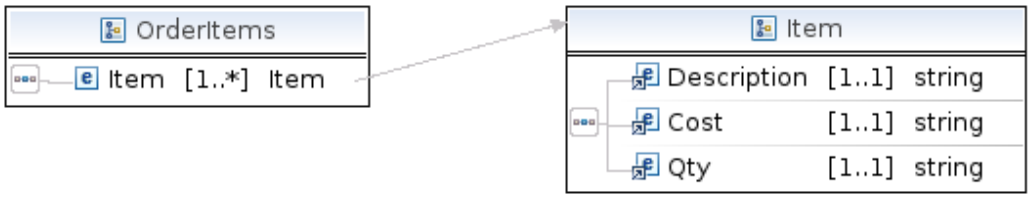


Table 3.43. Item Fields

Element	Description	Data Type	Required
Item	Order item	pc:OrderItem	Yes (at least one)

RecurringData

Figure 3.22. RecurringData Type

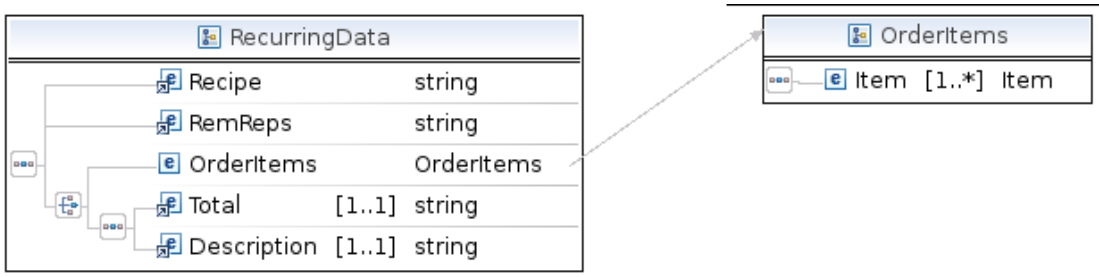


Table 3.44. RecurringData Fields

Element	Description	Data Type	Required
Recipe	Recipe Name	xs:string	Yes
RemReps	Number of recurring repetitions to establish	xs:string	Yes
OrderItems	Card Expiration Year	pc:OrderItems	Yes (1)
Total	Total for recurring instances	xs:string	Yes (1)
Description	Description for recurring instances	xs:string	Yes (1)
(1) Either OrderItems or Total and Description must be provided			

TransactionControl

Figure 3.23. TransactionControl Type

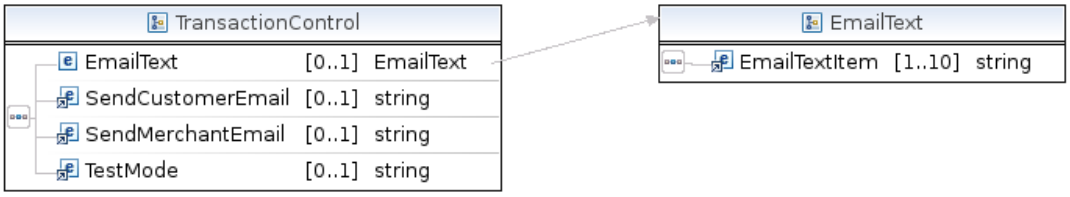


Table 3.45. Item Fields

Element	Description	Data Type	Required
EmailText	Email text lines	pc:EmailText	No
SendCustomerEmail	Controls whether customer email is sent on success	pc:TrueFalse	No
SendMerchantEmail	Controls whether merchant email is sent on success	pc:TrueFalse	No
TestMode	Controls whether transaction is run in test mode	pc:TrueFalse	No

VendorData

Figure 3.24. VendorData Type

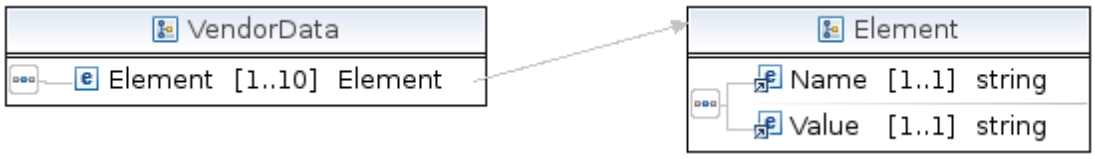


Table 3.46. VendorData Fields

Element	Description	Data Type	Required
Element	One meta data object	pc:Element	Yes (at least one, and up to 10)

Simple Types

AVSCategory

Table 3.47. AVSCategory Values

Value	Description
address	Address Matched
address_postal	Address and postal matched
address_zip5	Address and five digit ZIP matched
address_zip9	Address and nine digit ZIP matched
address_ok_postal_format_error	Address matched, postal format error
global_non_participant	International with no AVS support
international_address_not_verified	International with no AVS support
no_match	No address or postal match
no_response	No response
not_allowed	Not allowed
postal	Postal match
postal_ok_address_format_error	Postal matched, address format error
service_not_supported	AVS service not supported for card
unavailable	AVS service unavailable
zip5	Five digit ZIP matched
zip9	Nine digit ZIP matched

CardName

Table 3.48. CardName

Value
American Express
Australian Bankcard

Value
Discover
Diners Club/Carte Blanche
enRoute
Japanese Credit Bureau
MasterCard
Visa

ErrorCategory

Table 3.49. ErrorCategory Values

Value	Description
AVS_FAILURE	Transaction will be automatically voided
CVV2_FAILURE	Transaction will be automatically voided
INTERNAL_ERROR	Something unexpected happened
PROCESSOR_ERROR	Something such as DECLINED, etc
PROCESSOR_FAIL	Indicates something unexpected happened while trying to transmit the transaction to the processing network.
REQUEST_FORMAT	Request received has an invalid format
REQUEST_VALIDATION	XML content is invalid

Status

Table 3.50. ErrorCategory Values

Value	Description
AVS_FAILURE	Transaction will be automatically voided
CVV2_FAILURE	Transaction will be automatically voided
INTERNAL_ERROR	Something unexpected happened
PROCESSOR_ERROR	Something such as DECLINED, etc
PROCESSOR_FAIL	Indicates something unexpected happened while trying to transmit the transaction to the processing network.
REQUEST_FORMAT	Request received has an invalid format
REQUEST_VALIDATION	XML content is invalid

Status

Table 3.51. Status Values

Value	Description
error	This indicates that the transaction was processed by the procesing network but was not accepted.
fail	This indicates that an error occurred before the transaction was able to be passed to the processing network. The all caps FAILED response was inadvertantly introduced and may be removed at some point. It has not been removed yet in order to maintain backwards compatability with existing user code.
ok	The transaction was processed by the procesing network and was accepted.

TrueFalse

Table 3.52. TrueFalse Values

Value	Description
TRUE	True
FALSE	False

XML Considerations

- The XML Connection method is not activated on a gateway account by default. If a merchant desires to use the XML Connection method, an XML activation request needs to be sent to the sales rep. Also, be sure to enable the XML API in the Account Setting section of the Control Panel.
- The XML Connection method allows for a merchant to designate a TargetGateway value. If used, please pass the five digit gateway ID.
- There are certain characters such as '&' and '<' that may not appear in a XML request unless they are part of the actual XML structure. These characters have to be 'escaped' by using the forms & and <. More information about XML markup can be found at http://www.w3.org/TR/REC-xml/#syntax.
- We suggest that you do not try and create XML structures by simply appending to a string with print statements. Using a XML generation library will ensure that you do not accidentally create badly formatted documents due to user input. The disadvantage of using a library is that it will take a little longer to code. If you do not use a library you need to make sure that any XML node text you are adding that is coming from a user gets escaped as described above. If you do not escape these characters you will have transactions occasionally fail due to a bad XML structure.
- Here are some suggested XML generation libraries for different development environments. This is by no means an authoritative list and there are other good libraries available:
  - Java - http://jakarta.apache.org/ecs/index.html
  - Perl - http://search.cpan.org/~bholzman/XML-Generator-1.01/Generator.pm
  - PHP - http://us2.php.net/xmlwriter
  - Ruby - http://www.germane-software.com/software/rexml/

- .NET/C# - http://msdn2.microsoft.com/en-us/library/system.xml.xmltextwriter(VS.71).aspx