



Intellivative Incorporated

XML Interface Specification For Intellivative XML API v2.0

Last updated: August 3, 2009
Document version 2.1

Purpose

This document provides technical guidelines for integrating to the Intellivative Web services using the Intellivative API. The intended audiences for this document are software developers with a basic understanding of XML and how to send data to a web service using HTTPS / POST.

Dictionary of Terms

Credit Card Authorization: A request is sent to the card issuer to validate funds are available on the account. An authorization code is returned by the processor that is used later during the settlement process.

Voice Authorization: When a response to an authorization attempt identifies that there was a referral, meaning the authorization could not proceed without additional information, the merchant is informed they need to call the Voice Center. This is a service provided by the card issuer that requires additional information, such as validating the person using the card is actually who they say they are, or some other piece of information to alleviate any concerns they have in accepting the transaction. If they are convinced the transaction should proceed, they will provide a Voice Authorization Number that you would use later in a <authForce> or <saleForce> transaction.

Settlement: A process that collects all transactions identified by the merchant to settle, and submits them in a batch mode to the processor. This triggers a process to obtain funds from the card issuing bank (charges the card) and places funds into the merchant's bank account (deposit less fees as defined by the acquirer). The funds are available to the merchant depending on when items were marked for settlement and when the processor credits the merchant account.

Overview

Intellivative is a server-side application (Web service) that provides access to the Intellivative payment processing services. All communications take place over secure HTTPS links on the Internet. Requests are sent to the Intellivative servers by "POSTing" XML data that has been formatted per the schema defined in this specification. The response is returned as XML data in the body of the HTTP reply.

XML Data

Each XML request sent to the Intellivative servers and responses received from the servers have a standard format defined by a schema. The basic structure is shown below:

Request:

```
<transaction-request>
  <verification/>
  <order/>
</transaction-request>
```

Response:

```
<transaction-response>
  ...body...
</transaction-response>
```

Error Response: Returned when a validation, authentication or system error occurs:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<api-error>
  <errorCode></errorCode>
  <errorMsg></errorMsg>
</api-error>
```

Overview

The Intellivative XML API is a server-side application (Web service) that provides access to payment processing services. All communications take place over secure HTTPS links on the Internet. Requests are sent to the payment processing servers by "POSTing" XML data that has been formatted per the schema defined in this specification. The response is returned as XML data in the body of the HTTP reply.

The XML can be posted using any of several different web languages, including:

- .Net
- C#
- Java
- Perl
- PHP

See the sample code provided in client-samples.zip for implementation.

Posting URL

The posting URL to use for testing API transaction requests is:

```
https://apiint.intellivative.net:7443/merchantAPI/postXML
```

There is also a XML test web page available that you can use to test your XML strings:

`https://apiint.intellivative.net:7443/merchantAPI/xmlpost.html`

To use the XML test page:

1. Enter the appropriate posting URL
2. Select or enter the XML into the text area provided on the web page.
3. Click the Submit button
4. You should get a response to the XML request.

This page is handy for troubleshooting XML errors vs. connection errors.

Performing a Credit Card Sale or Authorize Only Transaction

The <auth> method requests an authorization number from the card issuer. An approved authorization means that the transaction dollar amount has been allocated but not deducted from the cardholder's account. This means the <responseCode> value provided in the response from the card issuer is "0" (see <responseCode> values later in this document). In this scenario, funds are not transferred to the merchant (a.k.a. Settlement) until the <capture> method is called using the authorization number for that transaction.

The <sale> method authorizes the amount and also marks the funds for settlement in one operation.

The <sale> and <auth> methods take the same input and produce the same output.

Note that there are two options for sending credit card information—only **one of the two** is required:

1. If the transaction is keyed in manually, credit card number and expiration date are required:
 - a. <creditCard><number/><expMonth/><expYear/></creditCard>
2. If the transaction is a card-swipe transaction, only the track data is required:
 - a. Either <track1Data/> or <track2Data/>

(Note: All "Required Fields" are identified by bold red font, all other fields are optional)

INPUT XML Structure:

```
<transaction-request>
  <verification>
    <merchantId/>
    <merchantKey/>
  </verification>
  <order>
    <sale>
      <referenceNum/>
      <ipAddress/>
      <billing>
        <name/>
        <address/>
        <address2/>
        <city/>
        <state/>
        <postalcode/>
        <country/>
```

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

```
        <phone/>
        <email/>
    </billing>
    <shipping>
        <name/>
        <address/>
        <address2/>
        <city/>
        <state/>
        <postalcode/>
        <country/>
        <phone/>
        <email/>
    </shipping>
    <transactionDetail>
        <payType>
            <creditCard>
                <number/>
                <expMonth/>
                <expYear/>
                <cvvInd/>
                <cvvNumber/>
                <track1Data/>
                <track2Data/>
                <eCommInd/>
            </creditCard>
        </payType>
    </transactionDetail>
    <payment>
        <chargeTotal/>
        <salesTaxTotal/>
        <shippingTotal/>
    </payment>
</sale>
    <clientData>
        <comments/>
    </clientData>
</order>
</transaction-request>
```

Response XML structure:

```
<transaction-response>
    <authCode/>
    <orderID/>
    <referenceNum/>
    <transactionID/>
    <transactionTimestamp/>
    <responseCode/>
    <responseMessage/>
    <avsResponseCode/>
    <cvvResponseCode/>
    <processorCode/>
    <processorMessage/>
    <errorMessage/>
</transaction-response>
```

Sample Input Code for a Credit Card Authorization

The code provided below is for illustration purposes. All data provided is purely fictional. If you copy this code for use in your own implementation, make sure you change the merchantId and the merchantKey to your own unique merchantId and merchantKey provided to you at setup.

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

```
<transaction-request>

  <verification>
    <merchantId>11111</merchantId>
      <!-- merchant ID number provided at setup -->
    <merchantKey>key</merchantKey>
      <!-- merchant security key provided at setup -->
  </verification>

  <order>
    <auth>
      <referenceNum>846392232</referenceNum>
        <!-- reference number for this transaction -->

      <ipAddress>123.123.123.123</ipAddress>
        <!-- IP address of the consumer -->
      <billing>
        <name>Susan Johnson</name>
          <!-- bill-to name -->
        <address>123 Main Street</address>
          <!-- billing address of the consumer -->
        <address2>Apt 36</address2>
          <!-- billing address, line 2 of the consumer -->
        <city>Moorpark</city>
        <state>CA</state>
        <postalCode>91307</postalCode>
        <country>US</country>
        <phone>818-123-1234</phone>
        <email>susjohnson@gmail.com</email>
      </billing>
      <shipping>
        <name>Susan Johnson</name>
          <!-- ship-to name for this order -->
        <address>123 Main Street</address>
          <!-- shipping address for this order -->
        <address2>Apt 36</address2>
        <city>Moorpark</city>
        <state>CA</state>
        <postalCode>91307</postalCode>
        <country>US</country>
        <phone>818-123-1234</phone>
        <email>susjohnson@gmail.com</email>
      </shipping>
      <transactionDetail>
        <payType>
          <creditCard>
            <number>4111-1111-1111-1111</number>
              <!-- 16-digit credit or debit card number, dashes and
spaces are okay -->
            <expMonth>12</expMonth>
              <!-- the 2-digit month the credit card expires -->
            <expYear>2008</expYear>
              <!-- the 4-digit year the credit card expires -->
            <cvvInd></cvvInd>
            <cvvNumber>123</cvvNumber>
              <!-- the 3- or 4-digit code, typically found on the
signature panel on the back of the card -->
            <track1Data></track1Data>
              <!-- the track data from the first track of the credit or
debit card, used for retail transactions -->
            <track2Data></track2Data>
```

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

```
        <!-- the track data from the 2nd track of the credit or
debit card, used for retail transactions -->
        <eCommInd>retail</eCommInd>
        <!-- e-commerce indicator: set to retail for in-person
txns, moto for mail or telephone order, or eci for e-commerce txns -->
        </creditCard>
    </payType>
</transactionDetail>
<payment>
    <chargeTotal>32.00</chargeTotal>    <!-- order total, typically =
subtotal + salesTax + shipping -->
    <salesTaxTotal>1.23</salesTaxTotal> <!-- sales tax on the order -->
    <shippingTotal>3.00</shippingTotal> <!-- shipping on the order -->
</payment>
</auth>
</order>

</transaction-request>
```

Sample Response Code from a Credit Card Authorization transaction

```
<transaction-response>
    <authCode>432582</authCode> <!-- authorization response code -->
    <orderID>C0A8C866:0119C7DF2702:E1D7:00E3FDA2</orderID> <!-- order identification
code -->

    <referenceNum>846392232</referenceNum> <!-- reference number -->
    <transactionID>32523465462</transactionID> <!-- identification number for the txn
-->
    <transactionTimestamp>1210664602</transactionTimestamp> <!-- time and date when
the txn was run, in GMT time zone -->
    <responseCode>0</responseCode> <!-- code indicating whether the txn was approved,
0 = approved, 1 = declined, 2 = fraud -->
    <responseMessage>AUTHORIZED</responseMessage> <!-- message to explain response
code -->
    <avsResponseCode>YYY</avsResponseCode> <!-- response code for address
verification, YYY = address and postal code match the address on file -->
    <cvvResponseCode>M</cvvResponseCode> <!-- response code for CVV test, M means the
CVV code sent matched -->
    <processorCode>A</processorCode> <!-- code from the processor indicating the
status of the transaction -->
    <processorMessage>APPROVED</processorMessage> <!-- message from the processor
indicating the status of the transaction -->
    <errorMessage/> <!-- had an error occurred, this field would contain an
explanation message -->
</transaction-response>
```

Performing a Forced Sale or Authorization

Forced sales and authorizations are used when an authorization code was received over the phone. Once a voice authorization is received, you would first determine which of these two transactions you choose to process.

The `<authForce>` method is similar to `<auth>` except the authorization number must be passed in. The only reason a `<authForce>` is used is when a `<auth>` is attempted and the response from the processor is to “Call Voice Center” to acquire the authorization code. Once an authorization code has been received over the phone by the merchant, then a `<authForce>` can be processed. When using the `<authForce>` method, you will not contact the payment processor to obtain an authorization since the code was already acquired verbally. In this scenario, funds are not transferred to the merchant’s account (a.k.a. Settlement) until the `<capture>` method is called using the authorization number passed in for that transaction.

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

Similarly, the only reason a <authSale> is used is when a <sale> is attempted and the response from the processor is to “Call Voice Center” to acquire the authorization code. Once an authorization code has been received over the phone by the merchant, then a <saleForce> can be processed to create the transaction in a captured state without going to the payment processor to obtain an authorization. It uses the same authorization code acquired verbally from the call center.

(Note: All "Required Fields" are identified by bold red font, all other fields are optional)

INPUT XML Structure:

```
<transaction-request>
  <verification>
    <merchantId/>
    <merchantKey/>
  </verification>
  <order>
    <authForce>
      <authCode/>
      <referenceNum/>
      <ipAddress/>
      <billing>
        <name/>
        <address/>
        <address2/>
        <city/>
        <state/>
        <postalcode/>
        <country/>
        <phone/>
        <email/>
      </billing>
      <shipping>
        <name/>
        <address/>
        <address2/>
        <city/>
        <state/>
        <postalcode/>
        <country/>
        <phone/>
        <email/>
      </shipping>
      <transactionDetail>
        <payType>
          <creditCard>
            <number/>
            <expMonth/>
            <expYear/>
            <cvvInd/>
            <cvvNumber/>
            <track1Data/>
            <track2Data/>
            <eCommInd/>
          </creditCard>
        </payType>
      </transactionDetail>
      <payment>
        <chargeTotal/>
        <salesTaxTotal/>
        <shippingTotal/>
      </payment>
    </authForce>
  </order>
</transaction-request>
```

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

```
    </authForce>
    <clientData>
      <comments/>
    </clientData>
  </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
  <orderID/>
  <referenceNum/>
  <transactionID/>
  <transactionTimestamp/>
  <responseCode/>
  <responseMessage/>
  <avsResponseCode/>
  <cvvResponseCode/>
  <processorCode/>
  <processorMessage/>
  <errorMessage/>
</transaction-response>
```

Sample Input Code for a Credit Card Forced Sale

The code provided below is for illustration purposes. All data provided is purely fictional. If you copy this code for use in your own implementation, make sure you change the merchantId and the merchantKey to your own unique merchantId and merchantKey provided to you at setup.

```
<transaction-request>
  <verification>
    <merchantId>11111</merchantId>
    <merchantKey>key</merchantKey>
  </verification>
  <order>
    <saleForce>
      <authCode>6432511111111</authCode>
      <referenceNum>22222222</referenceNum>
      <ipAddress>123.123.123.123</ipAddress>
      <billing>
        <name>Susan Johnson</name>
        <address>123 Garden Street</address>
        <address2>Apt 234</address2>
        <city>Moorpark</city>
        <state>CA</state>
        <postalcode>91307</postalcode>
        <country>US</country>
        <phone>818-123-1234</phone>
        <email>sjohnson334@yahoo.com</email>
      </billing>
      <shipping>
        <name>Susan Johnson</name>
        <address>123 Garden Street</address>
        <address2>Apt 234</address2>
        <city>Moorpark</city>
        <state>CA</state>
        <postalcode>91307</postalcode>
        <country>US</country>
        <phone>818-123-1234</phone>
        <email>sjohnson334@yahoo.com</email>
      </shipping>
      <transactionDetail>
        <payType>
```

```

    <creditCard>
      <number>4111111111111111</number>
      <expMonth>12</expMonth>
      <expYear>2008</expYear>
      <cvvInd></cvvInd>
      <cvvNumber>123</cvvNumber>
      <track1Data></track1Data>
      <track2Data></track2Data>
      <eCommInd>retail</eCommInd>
    </creditCard>
  </payType>
</transactionDetail>
<payment>
  <chargeTotal>3.00</chargeTotal>
  <salesTaxTotal>0.38</salesTaxTotal>
</payment>
</saleForce>
<clientData> <!-- optional fields for merchant-specific data -->
  <comments></comments>
</clientData>
</order>
</transaction-request>

```

Sample Response Code from a Forced Credit Card Sale transaction

```

<transaction-response>
  <orderID>C0A8C866:0119C7DF2702:E1D7:00E3FDA5</orderID>
  <referenceNum>846392235</referenceNum>
  <transactionID>32523465465</transactionID>
  <transactionTimestamp>1210664605</transactionTimestamp> <!-- time
of transaction, in Seconds since Jan 1 1970 -->
  <responseCode>0</responseCode>
  <responseMessage>CAPTURED</responseMessage>
  <avsResponseCode>YYY</avsResponseCode>
  <cvvResponseCode>M</cvvResponseCode>
  <processorCode>A</processorCode>
  <processorMessage>APPROVED</processorMessage>
  <errorMessage/>
</transaction-response>

```

Performing a Credit Capture

The <capture> method settles a transaction previously authorized with <auth> or <authForce>. Typically, the funds that are reserved during the authorization will be released after 7 days and a new authorization will need to be obtained and submitted.

(Note: All "Required Fields" are identified by bold red font, all other fields are optional)

INPUT XML Structure:

```

<transaction-request>
  <verification>
    <merchantId/>
    <merchantKey/>
  </verification>
  <order>
    <capture>
      <orderID/>
    </capture>
  </order>
</transaction-request>

```

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

```
        <referenceNum/>
        <payment>
            <chargeTotal/>
        </payment>
    </capture>
</order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
    <orderID/>
    <referenceNum/>
    <transactionID/>
    <transactionTimestamp/>
    <responseCode/>
    <responseMessage/>
    <processorCode/>
    <processorMessage/>
    <errorMessage/>
</transaction-response>
```

Sample Input Code for a Credit Card Capture

The code provided below is for illustration purposes. All data provided is purely fictional. If you copy this code for use in your own implementation, make sure you change the merchantID and the merchantKey to your own unique merchantID and merchantKey provided to you at setup.

```
<transaction-request>
    <verification>
        <merchantId>11111</merchantId>
        <merchantKey>key</merchantKey>
    </verification>
    <order>
        <capture>
            <orderID>C0A8C866:0119C7CF0530:3B39:009770A3</orderID>
            <referenceNum>846392233</referenceNum>
            <ipAddress>123.123.123.123</ipAddress>
            <payment>
                <chargeTotal>3.00</chargeTotal>
            </payment>
        </capture>
    </order>
</transaction-request>
```

Sample Response from a Credit Card Capture Transaction

```
<transaction-response>
    <orderID>C0A8C866:0119C7DF2702:E1D7:00E3FDA3</orderID>
    <referenceNum>846392233</referenceNum>
    <transactionID>32523465463</transactionID>
    <transactionTimestamp>1210664603</transactionTimestamp>
    <responseCode>0</responseCode>
    <responseMessage>CAPTURED</responseMessage>
    <processorCode>A</processorCode>
    <processorMessage>APPROVED</processorMessage>
    <errorMessage/>
</transaction-response>
```

Performing a Credit Card Return

The <return> method allows a specified amount to be refunded to the credit card account of a primary transaction. Intellivative will accept returns for up to 30 days after the captured transaction was processed.

(Note: All "Required Fields" are identified by bold red font, all other fields are optional)

INPUT XML Structure:

```
<transaction-request>
  <verification>
    <merchantId/>
    <merchantKey/>
  </verification>
  <order>
    <return>
      <orderID/>
      <referenceNum/>
      <payment>
        <chargeTotal/>
      </payment>
    </return>
  </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
  <orderID/>
  <referenceNum/>
  <transactionID/>
  <transactionTimestamp/>
  <responseCode/>
  <responseMessage/>
  <processorCode/>
  <processorMessage/>
  <errorMessage/>
</transaction-response>
```

Sample Input Code for a Credit Card Return

The code provided below is for illustration purposes. All data provided is purely fictional. If you copy this code for use in your own implementation, make sure you change the merchantID and the merchantKey to your own unique merchantID and merchantKey provided to you at setup.

```
<transaction-request>
  <verification>
    <merchantId>11111</merchantId>
    <merchantKey>key</merchantKey>
  </verification>
  <order>
    <return>
      <orderID>C0A8C866:0119C7DF2702:E1D7:00E3FDA4</orderID>
      <referenceNum>846392236</referenceNum>
      <ipAddress>123.123.123.123</ipAddress>
      <billing>
        <state>CA</state>
      </billing>
      <payment>
        <chargeTotal>3.00</chargeTotal>
      </payment>
    </return>
  </order>
</transaction-request>
```

```

    </return>
  </order>
</transaction-request>

```

Sample Response from a Credit Card Return Transaction

```

<transaction-response>
  <orderID>COA8C866:0119C7DF2702:E1D7:00E3FDA6</orderID>
  <referenceNum>846392236</referenceNum>
  <transactionID>32523465466</transactionID>
  <transactionTimestamp>1210664606</transactionTimestamp>
  <responseCode>0</responseCode>
  <responseMessage>CAPTURED</responseMessage>
  <processorCode>A</processorCode>
  <processorMessage>APPROVED</processorMessage>
  <errorMessage/>
</transaction-response>

```

Performing an Electronic Check Transaction

Use the <sale> method to submit an electronic check transaction to the ACH network for processing.

(Note: All "Required Fields" are identified by bold red font, all other fields are optional)

INPUT XML Structure:

```

<transaction-request>
  <verification>
    <merchantId/>
    <merchantKey/>
  </verification>
  <order>
    <sale>
      <referenceNum/>
      <ipAddress/>
      <billing>
        <address/>
        <address2/>
        <city/>
        <state/>
        <postalcode/>
        <country/>
        <phone/>
        <email/>
      </billing>
      <transactionDetail>
        <payType>
          <flagAsRecurring/>
          <ach>
            <bankRoutingNumber/>
            <achAccountNumber/>
          </ach>
        </payType>
      </transactionDetail>
      <payment>
        <chargeTotal/>
      </payment>
    </sale>
  </order>
</transaction-request>

```

OUTPUT XML Structure:

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

```
<transaction-response>
  <referenceNum/>
  <transactionID/>
  <transactionTimestamp/>
  <responseCode/>
  <responseMessage/>
  <errorMessage/>
</transaction-response>
```

Or

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<api-error>
  <errorCode></errorCode>
  <errorMsg></errorMsg>
</api-error>
```

Sample Input Code for an Electronic Check Transaction

The code provided below is for illustration purposes. All data provided is purely fictional. If you copy this code for use in your own implementation, make sure you change the merchantId and the merchantKey to your own unique merchantId and merchantKey provided to you at setup.

```
<transaction-request>
  <verification>
    <merchantId>11111</merchantId>
    <merchantKey>key</merchantKey>
  </verification>
  <order>
    <sale>
      <referenceNum>846392231</referenceNum>
      <ipAddress>123.123.123.123</ipAddress>
      <billing>
        <name>Susan Johnson</name>
        <address>123 Garden Street</address>
        <address2>Apt 234</address2>
        <city>Santa Barbara</city>
        <state>CA</state>
        <postalcode>93101</postalcode>
        <country>US</country>
        <phone>805-234-1888</phone>
        <email>sjohnson334@gmail.com</email>
      </billing>
      <transactionDetail>
        <payType>
          <ach>
            <bankRoutingNumber>111111111</bankRoutingNumber>
            <achAccountNumber>12345678</achAccountNumber>
          </ach>
        </payType>
      </transactionDetail>
      <payment>
        <chargeTotal>195.00</chargeTotal>
        <salesTaxTotal>0.00</salesTaxTotal>
        <shippingTotal></shippingTotal>
      </payment>
    </sale>
    <clientData>
      <comments/>
    </clientData>
  </order>
</transaction-request>
```

Sample Accepted Response from an Electronic Check Transaction

```
<transaction-response>
  <referenceNum>846392231</referenceNum>
  <transactionID>32523465461</transactionID>
  <transactionTimestamp>1210664601</transactionTimestamp>
  <responseCode>0</responseCode>
  <responseMessage>ACCEPTED</responseMessage>
  <errorMessage/>
</transaction-response>
```

Saving a Card (or account) on File

You use the `<command>` method when you wish to keep a repeat customer's credit card, debit card, or bank account information on file for use in future transactions.

Please note: The posting URL for this function differs from the transaction posting URL. The URL to use for testing the adding of a card on file is:

```
https://apiint.intellivative.net:7443/merchantAPI/postAPI
```

INPUT XML Structure for a saving a credit or debit card on file:

```
<api-request>
  <verification>
    <merchantId/>
    <merchantKey/>
  </verification>
  <command>add-card-on-file</command>

  <request>
    <customerId/>
    <creditCardNumber/>
    <expirationMonth/>
    <expirationYear/>
    <billingName/>
    <billingAddress1/>
    <billingAddress2/>
    <billingCity/>
    <billingState/>
    <billingZip/>
    <billingCountry/>
  </request>
</api-request>
```

Sample Input Code for adding a Card on File

The code provided below is for illustration purposes. All data provided is purely fictional. If you copy this code for use in your own implementation, make sure you change the merchantId and the merchantKey to your own unique merchantId and merchantKey provided to you at setup.

```
<api-request>
  <verification>
    <merchantId>11</merchantId>
    <merchantKey>key</merchantKey>
  </verification>
  <command>add-card-on-file</command>

  <request>
```

```

<customerId>2</customerId>
<creditCardNumber>4111111111111111</creditCardNumber>
<expirationMonth>12</expirationMonth>
<expirationYear>2008</expirationYear>
<billingName>Joe Consumer</billingName>
<billingAddress1>123 Main St.</billingAddress1>
<billingAddress2></billingAddress2>
<billingCity>Moorpark</billingCity>
<billingState>CA</billingState>
<billingZip>91311</billingZip>
<billingCountry>US</billingCountry>
</request>
</api-request>

```

Sample Accepted Response from a Save Card on File Request

```

<api-response>
  <errorCode>0</errorCode>
  <errorMessage></errorMessage>
  <command>add-card-on-file</command>
  <time>12012222222222</time>

  <result>
    <token>k11112233d</token>
  </result>
</api-response>

```

The **<token>** is the key piece of information in this response. You will need to send that token (in lieu of the credit card or bank account information) when you submit a transaction to be processed using the card on file.

Sample Input Code for adding a customer's checking account on File

The code provided below is for illustration purposes. All data provided is purely fictional. If you copy this code for use in your own implementation, make sure you change the merchantId and the merchantKey to your own unique merchantId and merchantKey provided to you at setup.

```

<api-request>
  <verification>
    <merchantId>00</merchantId>
    <merchantKey>key</merchantKey>
  </verification>
  <command>add-ach-on-file</command>
  <request>
    <customerId>1122</customerId>
    <bankRoutingNumber>123456789</bankRoutingNumber>
    <achAccountNumber>11223344</achAccountNumber>
  </request>
</api-request>

```

Removing a Card (or account) on File

You use the **<command>** method when you wish to delete a previously saved customer's credit card, debit card, or bank account information on file.

Please note: The posting URL for this function differs from the transaction posting URL. The URL to use for testing the removal of a card on file is:

```
https://apiint.intellivative.net:7443/merchantAPI/postAPI
```

INPUT XML Structure for a removing a card or account on file:

```

<api-request>
  <verification>
    <merchantId/>
    <merchantKey/>
  </verification>
  <command>delete-card-on-file</command>

  <request>
    <customerId/>
    <token/>
  </request>
</api-request>

```

Sample Accepted Response from a Delete Card on File Request

```

<api-response>
  <errorCode>0</errorCode>
  <errorMessage></errorMessage>
  <command>delete-card-on-file</command>
  <time>120123123123123</time>

  <result></result>
</api-response>

```

Performing a Void Transaction

The <void> method allows you to cancel a transaction before the transaction is funded. The transaction you are trying to void must be a credit or debit card transaction in a captured state. Once a transaction has been funded it will move to a settled state and will no longer be able to be voided. (If the transaction has already been funded, you will need to perform a **Return** to reverse the funds.)

(Note: All "Required Fields" are identified by bold red font, all other fields are optional)

INPUT XML Structure:

```

<transaction-request>
  <verification>
    <merchantId/>
    <merchantKey/>
  </verification>
  <order>
    <void>
      <transactionID/>
      <ipAddress/>
    </void>
  </order>
</transaction-request>

```

OUTPUT XML Structure:

```

<transaction-response>
  <transactionID/>
  <responseCode/>
  <responseMessage/>
  <processorCode/>
  <processorMessage/>
  <errorMessage/>
</transaction-response>

```

Or

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<api-error>
  <errorCode></errorCode>
  <errorMsg></errorMsg>
</api-error>
```

Sample Input Code for a Credit Card Void Transaction

The code provided below is for illustration purposes. All data provided is purely fictional. If you copy this code for use in your own implementation, make sure you change the merchantId and the merchantKey to your own unique merchantId and merchantKey provided to you at setup.

```
<transaction-request>
  <verification>
    <merchantId>11111</merchantId>
    <merchantKey>key</merchantKey>
  </verification>
  <order>
    <void>
      <transactionID>32123456456</transactionID>
      <ipAddress>123.123.123.123</ipAddress>
    </void>
  </order>
</transaction-request>
```

Sample Response from a Credit Card Void Transaction

```
<transaction-response>
  <transactionID>32123456456</transactionID>
  <responseCode>0</responseCode>
  <responseMessage>VOIDED</responseMessage>
  <processorCode>A</processorCode>
  <processorMessage>APPROVED</processorMessage>
  <errorMessage/>
</transaction-response>
```

Or

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<api-error>
  <errorCode></errorCode>
  <errorMsg></errorMsg>
</api-error>
```

Setting up a Recurring Payment

The <RecurringPayment> method allows merchant to create a scheduled payment (also known as a recurring transaction). A recurring transaction is a sale transaction, which is repeatedly run at specified intervals with the same values. Payment can be via credit/debit card or ACH.

(Note: All "Required Fields" are identified by bold red font, all other fields are optional)

INPUT XML Structure:

```
<transaction-request>
  <verification>
    <merchantId/>
    <merchantKey/>
  </verification>
  <order>
```

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

```
<recurringPayment>
  <referenceNum/>
  <ipAddress/>
  <billing>
    <name/>
    <address/>
    <address2/>
    <city/>
    <state/>
    <postalcode/>
    <country/>
    <phone/>
    <email/>
  </billing>
  <shipping>
    <name/>
    <address/>
    <address2/>
    <city/>
    <state/>
    <postalcode/>
    <country/>
    <phone/>
    <email/>
  </shipping>
  <transactionDetail>
    <payType>
      <creditCard> <!-- required for credit/debit card
transactions-->
        <number/> <!-- number, expmonth and expyear are
required for manually entered card transactions, whenever track1 and track2
data are not available -->
          <expMonth/>
          <expYear/>
          <cvvInd/>
          <cvvNumber/>
          <track1Data/> <!-- required for swiped card
transactions -->
          <track2Data/>
          <eCommInd/>
        </creditCard>
      </payType>
    </transactionDetail>
  <payment>
    <chargeTotal/>
    <salesTaxTotal/>
    <shippingTotal/>
  </payment>
  <recurring> <!-- fields to specify parameters of the scheduled
payment -->
    <action/>
    <startDate/>
    <period/>
    <frequency/>
    <installments/>
```

```

        <failureThreshold/>
    </recurring>
</recurringPayment>
<clientData>
    <comments/>
</clientData>
</order>
</transaction-request>

```

OUTPUT XML Structure:

```

<transaction-response>
    <authCode/>
    <orderID/>
    <referenceNum/>
    <transactionID/>
    <transactionTimestamp/>
    <responseCode/>
    <responseMessage/>
    <avsResponseCode/>
    <cvvResponseCode/>
    <processorCode/>
    <processorMessage/>
    <errorMessage/>
</transaction-response>

```

Or

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<api-error>
    <errorCode></errorCode>
    <errorMsg></errorMsg>
</api-error>

```

Processing a Card On File Transaction

With the *OnFile* payment type, the Credit card or ACH information for the customer is already stored in the Database. The merchant passes the *customerID* and *payment Token* and the payment information is taken from the Database.

The <Sale> method authorizes the amount and also marks the funds for settlement in one operation.

(Note: All "Required Fields" are identified by bold red font, all other fields are optional)

INPUT XML Structure:

```

<transaction-request>
    <verification>
        <merchantId/>
        <merchantKey/>
    </verification>
    <order>
        <sale>
            <referenceNum/>
            <ipAddress/>
        </sale>
        <billing>

```

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

```

    <name/>
    <address/>
    <address2/>
    <city/>
    <state/>
    <postalcode/>
    <country/>
    <phone/>
    <email/>
  </billing>
  <shipping>
    <name/>
    <address/>
    <address2/>
    <city/>
    <state/>
    <postalcode/>
    <country/>
    <phone/>
    <email/>
  </shipping>
  <transactionDetail>
    <payType>
      <onfile>
        <customerId/>
        <token/>
      </onfile>
    </payType>
  </transactionDetail>
  <payment>
    <chargeTotal/>
    <salesTaxTotal/>
    <shippingTotal/>
  </payment>
</sale>
<clientData>
  <comments/>
</clientData>
</order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
  <authCode/>
  <orderID/>
  <referenceNum/>
  <transactionID/>
  <transactionTimestamp/>
  <responseCode/>
  <responseMessage/>
  <avsResponseCode/>
  <cvvResponseCode/>
  <processorCode/>
  <processorMessage/>
  <errorMessage/>
</transaction-response>
```

Or

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<api-error>
  <errorCode></errorCode>
  <errorMsg></errorMsg>
</api-error>
```

Checking on Transactions that Timed Out

If a transaction times out before you get a response from the system, you can send a transaction inquiry request to find out whether the transaction was received and processed. This is a two-step process.

First, you would send a *submit-transaction-inquiry* request. This request tells the server that you would like to get a status on a specific transaction. You identify the transaction you want status on by including the transaction reference number (`<referenceNum>`), which is a unique identifier that you assign, manage, and use to track your transactions. When the server receives this first request, it puts it in a queue for processing when it has some free time and it sends back a request ID number (`<requestId>`).

Step 1: Sample submit-transaction-inquiry request

```
<api-request>
  <verification>
    <merchantId>123</merchantId>
    <merchantKey>mykey</merchantKey>
  </verification>
  <command>submit-transaction-inquiry</command>

  <request>
    <referenceNum>12345678</referenceNum>
  </request>
</api-request>
```

Sample Response from submit-transaction-inquiry request:

```
<api-response>
  <errorCode>0</errorCode>
  <errorMessage></errorMessage>
  <command>submit-transaction-inquiry</command>
  <time>120123123123123</time>

  <result>
    <requestId>87654321</requestId>
  </result>
</api-response>
```

Now, the second step is to go back sometime later (perhaps 30 minutes later) and send a second request to ask the server if it's completed your request yet, and if so, what is the status of your transaction? You do this by sending a *get-transaction-inquiry-status* request that includes the *requestId* you received from your first request, as shown in the sample code below.

Step 2: Sample get-transaction-inquiry-status request

```
<api-request>
  <verification>
```

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

```
<merchantId>123</merchantId>
  <merchantKey>mykey</merchantKey>
</verification>
<command>get-transaction-inquiry-status</command>

<request>
  <requestId>87654321</requestId>
</request>
</api-request>
```

Sample Response from get-transaction-inquiry-status request:

```
<api-response>
  <errorCode>0</errorCode>
  <errorMessage></errorMessage>
  <command>get-transaction-inquiry-status</command>
  <time>120123123123123</time>

  <result>
    <requestStatus>COMPLETED</requestStatus>
    <transactionStatus>NOT FOUND</transactionStatus>
  </result>
</api-response>
```

In this response, the server is telling you it has finished checking on your inquiry, and it did not find the transaction you sent. In this case, you would resend the transaction.

Possible responses you might receive from a get-transaction-inquiry request are as follows:

requestStatus	transactionStatus	Meaning	Recommended action
NEW	--	Your status request hasn't been processed yet.	Resend the <i>get-transaction-inquiry-status</i> request at a later time
COMPLETED	NOT FOUND	Your request for status is complete, but we didn't find your transaction.	Resend the transaction—it wasn't received the first time.
COMPLETED	APPROVED	Your request is complete; we found your transaction and it was approved.	No action required.
COMPLETED	DECLINED	Your request is complete; we found your transaction and it was declined.	Use whatever action you normally take for a declined transaction. You may wish to ask the customer for an alternate form of payment.

Frequently Asked Questions

Q: Which URL should I use to post test transactions to the Intellivative system?

A: You need to use the API Integration Test URL (<https://apiint.intellivative.net:7443/merchantAPI/postXML>) to post transactions from your application.

Q: What is this other “XML Test URL” provided with my API test account?

A: The XML Test URL (<https://apiint.intellivative.net:7443/merchantAPI/xmlpost.html>) is a web page that you can open in your browser to test sample XML calls to the API. This is helpful as you begin your integration since you have an easy way of testing your XML strings. It can be helpful for troubleshooting errors as well.

Note that the XML Test URL is for testing your XML strings only. It should not be used for posting transactions since it is not a post interface. Posting transactions to the XML Test URL will yield invalid results.

Q: What type of transaction should I start with?

A: We suggest you first attempt a Credit Card Sale transaction. Before you can attempt a Return or Void transaction, you’ll need a response from a Sale transaction.

Q: When I attempt to post a transaction, my server throws a security exception saying the remote certificate is invalid. What is wrong?

A: We use a self-signed cert in the test environment. You need to import the certificate into your environment since it is not issued by a certificate authority. Once you complete your testing and move your integration to production, this won’t be an issue because we use a fully-signed certificate in the production environment.

Q: I’m attempting to post a transaction, and am getting errorcode 1 in the response. What does this mean and what do I do about it?

Sample XML error message:

```
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?><api-error>
<errorCode>1</errorCode><errorMsg><![CDATA[Fatal Error:
URI=null Line=1: Premature end of file.]]></errorMsg></api-
error>
```

A: If you are getting this error, you are successfully communicating with our server, but something’s wrong in what you’re sending. First, make sure that you have the content type set to “text/xml”.

If you are still getting an error, try copying the XML you sent in the post into the XML Test URL (<https://apiint.intellivative.net:7443/merchantAPI/xmlpost.html>) and comparing the results. If you still can’t track down the problem, contact your technical support person who will be able to help you troubleshoot the problem.

Q: How do I know if my transaction is successful?

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

A: A successful response will include a response code of 0 (indicating that the transaction was approved or accepted) and a response message. Also, no error code or error message will be included, like this:

```
<?xml version="1.0" encoding="UTF-8"?><transaction-
response><authCode>123456</authCode><orderID>C0A86E6F:011CC
02A270B:86E8:01417690</orderID><referenceNum>846392232</ref
erenceNum><transactionID>1700</transactionID><transactionTi
mestamp>1222994700</transactionTimestamp><responseCode>0</r
esponseCode><responseMessage>CAPTURED</responseMessage><avs
ResponseCode>YYY</avsResponseCode><cvvResponseCode>M</cvvRe
sponseCode><processorCode>A</processorCode><processorMessag
e>APPROVED</processorMessage><errorMessage/></transaction-
response>
```

A response code of **0** indicates an approved transaction. See the table below for other possible response codes.

Value	Name	Description
0	APPROVED ACCEPTED VERIFIED	The transaction was approved, accepted or verified
1	DECLINED	The transaction was declined.
2	FRAUD	The transaction was declined due to possible fraud
3	REFERRAL	The transaction was placed in a referral mode which requires follow up by the merchant
259	EXPIRED CARD	The credit card has expired
1022	PROCESSOR ERROR	There was an error at the processor
1024	INVALID REQUEST	The transaction is not valid
1025	INVALID MERCHANT	The merchant is not valid for this transaction
2048	INTERNAL ERROR	There was a system error
4096	COMMUNICATION ERROR	There was a communication error in the payment system

Input Parameter Descriptions

Parameter	Max Size	Description	Input / Output
<ach><bankRoutingNumber>	9	The Routing and Transit (ABA) Number of the bank on which the draft is drawn – used for an e-check transaction	I
<ach><achAccountNumber>	17	The consumer's bank account number – used for an e-check transaction	I
<ach><checkNumber>	8	The check identification number – used for an e-check transaction	I
<billing><address>	128	Billing address for the payment type being	I

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

		used. The numeric portion of the address (the house number) is typically used for address verification on credit/debit card transactions	
<billing><address2>	128	This area is for the suite number or other information that did not fit into the Billing Address parameter	I
<billing><city>	64	Billing city for the payment type being used	I
<billing><country>	64	Billing country for the payment type being used	I
<billing><email>	128	Billing email address for the payment type being used	I
<billing><name>	64	Billing name for the payment type being used	I
<billing><phone>	16	Billing phone number for the payment type being used	I
<billing><postalcode>	16	Billing postal code for the payment type being used such as zip code, for domestic US	I
<billing><state>	32	Billing state for the payment type being used	I
<chargeTotal>	19	Total amount of the transaction	I
<creditCard><expMonth>	2	Two digit Expiration month for the credit card being used	I
<creditCard><expYear>	4	Four digit Expiration year for the credit card being used	I
<creditCard><cvvNumber>	4	Three or Four digit card value printed on the back of the card. For American Express, the Four digit value is printed on the front of the card	I
<creditCard><cvvInd>	14	An optional enumerated field to provide an explanation as to why the CVV code was not passed. Possible values are “not imprinted”, “illegible”, and “bypassed”.	I
<creditCard><number>	19	Credit Card number on the card	I
<creditCard><track1Data>	128	Credit Card track data. Required for card-present swiped transactions	I
<creditCard><track2Data>	128	Credit Card track data. Required for card-present swiped transactions	I
<flagAsRecurring/>	0	If present, indicates this transaction was triggered by a recurring payment schedule maintained by the merchant. Use it only for recurring payments—it should not be present for one-time transactions and is not needed for recurring payments scheduled via the TriHealix system.	I
<clientData><comments/>	19	A field to pass any comments related to this transaction	I
<ipAddress>	16	The IP Address of the consumer’s computer system. The merchant’s IP Address is automatically collected	I
<merchantId>	20	The merchant’s unique identification number	I
<merchantKey>	80	The merchant’s key that was provided during the initial set-up of merchant’s account	I
<onfile><customerId>	20	The Customer’s unique identification number. Used when the customer’s payment information is already on file.	I

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

<onfile><token>	20	Token identifying the payment type for this customer. There can be multiple payment types on file for a customer and this token uniquely identifies the payment type to use for this transaction. Used when the customer's payment information is already on file.	I
<orderId>	128	The value that groups together a set of transactions. The merchant can assign the value or if the merchant doesn't assign an <orderId> then the id of the primary transaction will be used	I/O
<recurring><action>	10	The action to take for a recurring transaction. This field may be set to <i>new</i> , <i>cancel</i> or <i>modify</i> .	I
<recurring><startDate>	12	The date to start the recurring transaction.	I
<recurring><period>	10	The interval of time between transactions. Valid values are <i>day</i> , <i>week</i> , <i>month</i> , and <i>quarter</i> .	I
<recurring><frequency>	2	Integer value indicating the frequency of the recurring transaction. That is, if you wish the recurring transaction to run once every 2 weeks, set the <period> to <i>week</i> and the <frequency> to <i>2</i> .	I
<recurring><installments>	4	Integer value indicating the number of installments to charge the customer.	I
<recurring><failureThreshold>	2	Integer value for the number of times a recurring transaction will be retried in case of failure.	I
<referenceNum>	128	The reference number will typically be an exclusive identification number from the merchant's payment system. This value should be unique	I/O
<shipping><address>	128	Shipping address for the goods purchased	I
<shipping><address2>	128	Shipping address for the suite number or other information that did not fit in the Shipping Address parameter	I
<shipping><city>	64	Shipping city for the goods purchased	I
<shipping><country>	64	Shipping country for the goods purchased	I
<shipping><email>	128	Shipping email address for the goods purchased	I
<shipping><name>	64	Shipping name for the goods purchased	I
<shipping><phone>	16	Shipping phone number for the goods purchased	I
<shipping><postalcode>	16	Shipping postal code for the goods purchased	I
<shipping><state>	32	Shipping state for the goods purchased	I
<transactionId>	64	The unique identification number given to each transaction so that every transaction can be individually identified. This value should be stored in the merchant's system.	I/O

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

Output Parameter Descriptions

Parameter	Max Size	Description	Input / Output
<authCode>	6	Authorization code returned from the processor when an authorization or sale is approved.	O
<avsResponseCode>	3	Response from the processor indicating whether the Address information supplied matches the address on file of the Card Issuer	O
<cvvResponseCode>	1	One digit response code from the processor indicating whether the card code information supplied matches the card code on file at the Card Issuer	O
<errorCode>	1	A code indicating the type of error that occurred	O
<errorMsg>	128	Gives the details of errors that occurred. If you receive error code 1 and an <errorMsg> tag, something is wrong with the XML.	O
<errorMessage>	128	Gives the details of errors that might occur when the XML was properly presented, but the transaction failed for some other reason (e.g., invalid card number)	O
<processorCode>	4	A code from the processor indicating the status of the transaction.	O
<processorMessage>	8	A message from the processor indicating the status of the transaction.	O
<responseCode>	6	A numeric code indicating the results of the attempted transaction. Common responses are 0 for approved/verified/accepted, 1 for declined, 2 for fraud. See the chart below for more possible responseCode values.	O
<responseMessage>	32	Message that corresponds to the <responseCode> value	O
<transactionTimestamp>	13	Transaction creation date and time in GMT, in seconds since Jan 1, 1970.	O

<avsResponseCode> Values

Value	Description
X/Y	Full Match, Address and Postal Code
A	Address match, Postal Code does not
W/Z	Postal Code match, Address does not
N	Nothing matches
R	Retry, unable to process
S/G/U	No AVS data from issuer
E	Invalid AVS data from issuer
B	Address only match
C	Unable to verify
D	Address and Postal Code Match
I	International unable to verify
M	Address and Postal Code match
P	Postal Code match only

Intellivative Incorporated XML v2.0 Interface Specification

Confidential

<cvvResponseCode> Values

Value	Description
M	Card Verification Values match
N	Card Verification Value does not match or is invalid
P	Card Verification Value not processed
U	Issuer not registered

<responseCode> / <responseMessage> Values

Value	Name	Description
0	APPROVED ACCEPTED VERIFIED	The transaction was approved, accepted or verified
1	DECLINED	The transaction was declined.
2	FRAUD	The transaction was declined due to possible fraud
3	REFERRAL	The transaction was placed in a referral mode which requires follow up by the merchant
259	EXPIRED CARD	The credit card has expired
1022	PROCESSOR ERROR	There was an error at the processor
1024	INVALID REQUEST	The transaction is not valid
1025	INVALID MERCHANT	The merchant is not valid for this transaction
2048	INTERNAL ERROR	There was a system error at Intellivative
4096	COMMUNICATION ERROR	There was a communication error in the Intellivative payment system