Paymentsite Application Programming Interface (API)

Universal API Specification

Last updated: August 9, 2018

Document version 3.5.2.9

Paymentsite API is created, owned, hosted, and managed by Intrix Technology, Inc

Paymentsite Universal API Interface Specification

Contents

1. Purpose

The Paymentsite Universal API allows developers to connect to any of the payment processors that are currently integrated with the Paymentsite Gateway through one common interface.

This document provides technical guidelines for integrating to the Paymentsite Web services using the

Paymentsite Universal 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.

2. 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 for capturing the funds.

Voice Authorization: When a response to an authorization or sale transaction attempt identifies that there was a referral, meaning the authorization could not proceed without additional information, the merchant is informed that 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.

3. XML Data

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

Request:

```
<transaction-request>
    <version/>
    <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>
```

4. Overview

The Paymentsite Universal API is a server-side application (Web service) that provides access to transaction 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

At this time, a Java example is available. Samples in other languages are in development; please ask your sales support representative if you would like a sample in one of the other languages listed.

5. Posting URL for Test Transaction

Before you can begin testing, you will need a test account with Paymentsite. Please contact us at support@paymentsite.com to set up your test account. Test accounts are not connected to the payment processor and even though you will receive a response from each request sent, transactions sent to a test account are not actually processed and no money is moved.

Your test account information should include a merchant ID and a key that you will include in all requests you send to the Paymentsite system. This merchant ID and key is merchant specific.

The posting URL to use for **testing and development with** API **transaction requests** is as follows. This URL is strictly for posting transactions such as auth, capture, sale, return, etc.

Posting URL for test transaction requests:

https://apiint.paymentsite.com/UniversalAPI/postXML

WARNING: DO NOT USE REAL CARD OR ACCOUNT DATA IN TEST ENVIRONMENTS

(test environments must contain only test data including test card numbers)

The posting URL to use for **testing and development with other API requests** (requests such as add-consumer, which are not transactional in nature) is:

Posting URL for test requests which are NON-transactional:

https://apiint.paymentsite.com/UniversalAPI/postAPI

Please note that any transactions sent to either of the testing URLs above are not live and will not result in any financial transactions. Data created in the test system is not transported to the production system, so all setup functions must be re-run in production once you are done testing.

6. Posting URL for Live Transactions

Once you have completed your testing and are ready to go live with your application and process real transactions, you will need a production account and production posting URLs. Please contact support for your production credentials

The posting URL to use for **LIVE** API **transaction requests** is as follows. This URL is strictly for posting transactions such as auth, capture, sale, return, etc

Posting URL for LIVE transaction requests:

https://api.paymentsite.com/UniversalAPI/postXML

DO NOT USE THIS URL FOR TESTING OR DEVELOPMENT!

The posting URL to use for **sending other LIVE API requests** (requests such as add-consumer, which are not transactional in nature) is:

Posting URL for LIVE requests which are NON-transactional:

https://api.paymentsite.com/UniversalAPI/postAPI

DO NOT USE THIS URL FOR TESTING OR DEVELOPMENT!

Please contact support for your production credentials.

Important security note: if you are using the PHP cURL library for establishing a connection, please be aware of security risks if CURLOPT_SSL_VERIFYPEER is disabled. Curl will not verify the server's certificate against the trusted Certificate Authorities if this option is disabled and it will expose you to many "man-in-the-middle" attacks. Details on the security risk can be found on the Public Safety Canada web site. To avoid this security risk, we strongly recommend you ensure that the CURLOPT_SSL_VERIFYPEER option is **always enabled**.

7. Transaction APIs

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—we recommend that** for a swiped transaction that you provide ONLY the track data. There is no need to pass the credit card number if you send the track data. Some processors only require track 2, but others require both tracks, so we recommend you send both tracks:
   a. `<track1Data/>` and/or `<track2Data/>`
   b. Or send `<trackData/>` containing both tracks in one field

(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/>
        <invoiceNumber/>
            <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/>       <!-- Required for keyed credit card transactions -->
                <expMonth/>     <!-- Required for keyed credit card transactions -->
                <expYear/>      <!-- Required for keyed credit card transactions -->
                <cvvInd/>
                <cvvNumber/> <!-- Required if Heartland is the processor -->
                <deviceType/> <!-- Type of device used for card-swipe. If using an encrypted device for swiping the credit card, you must send and populate this tag. If not, you may leave this tag out. -->
                <deviceKSN/><!-- device key serial number. If using an encrypted device for swiping the credit card, you must send and populate this tag. -->
                <track1Data encrypted="Y"/> <!-- track data is required for swiped credit card transactions. We recommend sending both track1 and track2. If the merchant is using an encrypted device, set encrypted = “Y” for all track data sent -->
                <track2Data encrypted="Y"/>
                <eCommInd/>
                <signatureImage/>
            </creditCard>
        </payType>
    </transactionDetail>
    <payment>
        <chargeTotal/>
        <salesTaxTotal/> <!-- if included, must contain a numeric value -->
        <shippingTotal/> <!-- if included, must contain a numeric value -->
        <convenienceFee/> <!-- if included, must contain a numeric value -->
        <currencyCode/> <!-- if included, must contain a value. If NOT included, defaults to USD -->
    </payment>
    <consumerPrimaryId/>
```

```
            <consumerExternalId/>
        </sale>
    </order>
    <clientData>
      <customField1/> <!-- custom fields which can be used to pass your own fields to be associated with a
transaction -->
        <customField2/>
        <customField3/>
        <customField4/>
         <customField5/>
        <comments/> <!-- comments field to be used to allow you to pass comments to be associated with a
transaction -->
    </clientData>
</transaction-request>
```

Response XML structure:

```
<transaction-response>
        <authCode/>
        <orderID/>
        <referenceNum/>
        <transactionID/>
        <transactionTimestamp/>
        <responseCode/>
        <responseMessage/>
        <partiallyApprovedAmount/>
        <accountBalance/>
        <avsResponseCode/>
        <cvvResponseCode/>
        <processorCode/>
        <processorMessage/>
        <processorReferenceNumber/>
        <processorTransactionID/>
        <creditCardScheme/>
        <errorMessage/>
</transaction-response>
```

Or

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

Important! Please note: If a <partiallyApprovedAmount/> tag is included in the response, this means the sale or authorization was only approved for **part of the amount requested**. The transaction will return an APPROVED response, and the tag <partiallyApprovedAmount/> will include the amount that the transaction was approved for. This applies for gift cards and pre-paid cards that have limits.

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.

```
<transaction-request>
  <version>3.1.1.1</version>
  <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 -->
     <invoiceNumber>123456</invoiceNumber>
     <billing>
      <name>Tom Customer</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>customer@example.com</email>
     </billing>
     <shipping>
      <name>Tom Customer</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>customer@example.com</email>
     </shipping>
     <transactionDetail>
      <payType>
       <creditCard>
                    <number>4111111111111111</number>
              <!-- 16-digit credit or debit card number, no dashes or spaces -->
                    <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 -->
          <!-- Required if Heartland is the processor -->
        <track1Data></track1Data>
          <!-- the track data from the first track of the credit or debit card, used for retail transactions -->
        <track2Data></track2Data>
            <!-- 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 -->
        <signatureImage></signatureImage>
```

```
            <!-- signature image: a field to send an image of the customer's signature, if captured electronically
-->
        </creditCard>
      </payType>
    </transactionDetail>
      <payment>
            <chargeTotal>32.00</chargeTotal>
          <!-- order total, typically =subtotal + salesTax + shipping -->
      <salesTaxTotal>1.23</salesTaxTotal>
          <!-- sales tax on the order. If included, must contain a numeric value -->
      <shippingTotal>3.00</shippingTotal>
          <!-- shipping on the order. If included, must contain a numeric value -->
      <currencyCode>USD</currencyCode>
          <!-- if included, must contain a value. If NOT included, defaults to USD -->
    </payment>
  </auth>
 </order>
</transaction-request>
```

Sample Response Code from a Credit Card Authorization transaction

If the response code is **0**, the transaction was successful.

```
<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 -->
    <processorTransactionID>262</processorTransactionID>
    <!-- if the payment processor sends back a transaction id, it will be provided in this tag. Merchants can use
this identifier to help locate a transaction from a processor report -->
    <processorReferenceNumber>111221266</processorReferenceNumber>
    <!-- if the payment processor sends back a transaction reference number, it will be provided in this tag -->
    <partiallyApprovedAmount>12.99</partiallyApprovedAmount>
```

<!-- If this tag is included, the transaction has not been approved for the full amount requested. The amount approved will be included in this tag. -->
<accountBalance>35.99</accountBalance> <!-- the remaining balance that will remain on the card after the transaction. This tag is only included for prepaid and gift cards. -->
<creditCardScheme>Visa</ creditCardScheme > <!-- card brand -->
<errorMessage/> <!-- had an error occurred, this field would contain an explanation message -->
</transaction-response>

Performing a Forced Sale or Authorization Transaction

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.

Similarly, the only reason a <saleForce> 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 that Force Sale transactions are exceedingly rare.

(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/>
      <invoiceNumber/>
      <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/>      <!-- Required for keyed credit card transactions -->
              <expMonth/>    <!-- Required for keyed credit card transactions -->
              <expYear/>     <!-- Required for keyed credit card transactions -->
        <cvvInd/>
        <cvvNumber/> <!-- Required if Heartland is the processor -->
        <deviceType/>
          <!-- Type of device used for card-swipe. If using an encrypted device for swiping the credit card, you
must send and populate this tag. If not, you may leave this tag out. -->
        <deviceKSN/>
          <!-- device key serial number. If using an encrypted device for swiping the credit card, you must send
and populate this tag. -->
        <track1Data encrypted="Y"/>
          <!-- track data is required for swiped credit card transactions. We recommend sending both track1
and track2. If the merchant is using an encrypted device, set encrypted = "Y" for all track data sent -->
        <track2Data encrypted="Y"/>
        <eCommInd/>
       </creditCard>
       </payType>
     </transactionDetail>
     <payment>
          <chargeTotal/>
      <salesTaxTotal/>
        <!-- sales tax on the order. If included, must contain a numeric value -->
      <shippingTotal/>
        <!-- shipping on the order. If included, must contain a numeric value -->
      <currencyCode/>
        <!-- if included, must contain a value. If NOT included, defaults to USD -->
     </payment>
   </authForce>
   <clientData>
    <comments/>
   </clientData>
  </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
    <orderID/>
    <referenceNum/>
    <transactionID/>
    <transactionTimestamp/>
    <responseCode/>
    <responseMessage/>
    <avsResponseCode/>
    <cvvResponseCode/>
    <processorCode/>
    <processorMessage/>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <partiallyApprovedAmount/>
    <creditCardScheme/>
    <accountBalance/>
    <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 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>Tom Customer</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>customer@example.com</email>
```

```
            </billing>
            <shipping>
                <name>Tom Customer</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>customer@example.com</email>
            </shipping>
            <transactionDetail>
                <payType>
                    <creditCard>
                        <number>4111111111111111</number>
                        <expMonth>12</expMonth>
                        <expYear>2008</expYear>
                        <cvvInd></cvvInd>
                        <cvvNumber>123</cvvNumber><!-- Required if Heartland is the processor -->
                        <eCommInd>retail</eCommInd>
                    </creditCard>
                </payType>
            </transactionDetail>
        <payment>
            <chargeTotal>3.00</chargeTotal>
            <salesTaxTotal>0.38</salesTaxTotal>
            <!-- sales tax on the order. If included, must contain a numeric value -->
            <currencyCode>USD</currencyCode>
        </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

If the response code is 0, the transaction was successful.

```
<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>
    <creditCardScheme>Visa</ creditCardScheme >
    <errorMessage/>
</transaction-response>
```

Performing a Credit Card 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/>
      <referenceNum/>
      <payment>
        <chargeTotal/>
        <currencyCode/>
      </payment>
    </capture>
  </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
    <orderID/>
    <referenceNum/>
    <transactionID/>
    <transactionTimestamp/>
    <responseCode/>
    <responseMessage/>
    <processorCode/>
    <processorMessage/>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <partiallyApprovedAmount/>
    <creditCardScheme/>
    <accountBalance/>
    <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 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>
    <version>3.5.2.6</version>
    <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

If the response code is 0, the transaction was successful. Items that indicate whether the transaction was

successful are shown in **bold blue font**.

```
<transaction-response>
    <orderID>C0A8C866:0119C7DF2702:E1D7:00E3FDA3</orderID>
    <referenceNum>846392233</referenceNum>
    <transactionID>32523465463</transactionID>
    <transactionTimestamp>1210664603</transactionTimestamp>
    <responseCode>0</responseCode>
    <responseMessage>CAPTURED</responseMessage>
    <avsResponseCode />
    <cvvResponseCode />
```

```
    <processorCode>A</processorCode>
    <processorMessage>APPROVED</processorMessage>
    <creditCardScheme>Visa</ creditCardScheme >
    <errorMessage/>
</transaction-response>
```

Processing Restaurant Transactions

The same <auth> and <capture> methods apply when processing transactions for restaurants, but the captured amount can differ from the authorized amount by adding a tip. The ecommInd field should be set to "restaurant" to allow for the tip to be added.

(Note: Restaurant-specific fields are identified by bold blue font.)

INPUT XML Structure for a Restaurant Authorization:

```
<transaction-request>
    <verification>
        <merchantId/>
        <merchantKey/>
    </verification>
    <order>
        <auth>
            <referenceNum/>
            <ipAddress/>
            <transactionDetail>
                <payType>
                    <creditCard>
                        <number/>          <!-- Required for keyed credit card transactions -->
                        <expMonth/>        <!-- Required for keyed credit card transactions -->
                        <expYear/>         <!-- Required for keyed credit card transactions -->
                        <cvvInd/>
                        <cvvNumber/>
<!-- track data is required for swiped credit card transactions. We recommend sending both track1 and track2. If the merchant is using an encrypted device, set encrypted = "Y" for all track data that is sent. -->
                        <track1Data/>
                        <track2Data/>
                        <eCommInd>restaurant</eCommInd>
                    </creditCard>
                </payType>
            </transactionDetail>
            <payment>
                <chargeTotal/>
                <salesTaxTotal/>
            </payment>
        </auth>
    </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
      <orderID/>
      <referenceNum/>
      <transactionID/>
      <transactionTimestamp/>
      <responseCode/>
      <responseMessage/>
      <processorCode/>
      <processorMessage/>
      <processorReferenceNumber/>
      <processorTransactionID/>
      <partiallyApprovedAmount/>
      <creditCardScheme/>
      <accountBalance/>
      <errorMessage/>
</transaction-response>
```

Or

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

INPUT XML Structure for a Restaurant Capture:

Now when the capture transaction is sent, a tip can be added. The amount captured will be the authorized amount (or chargetotal) plus the tip amount.

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

Sample Response from a Credit Card Capture Transaction

If the response code is 0, the transaction was successful. Items that indicate whether the transaction was

successful are shown in **bold blue font**.

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

Processing a Credit Card Return

The <return> method allows a specified amount to be refunded to the credit card account of a primary transaction.

(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/>
                <currencyCode/>
            </payment>
        </return>
    </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
    <orderID/>
    <referenceNum/>
    <transactionID/>
    <transactionTimestamp/>
    <responseCode/>
    <responseMessage/>
    <processorCode/>
    <processorMessage/>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <partiallyApprovedAmount/>
    <creditCardScheme/>
    <accountBalance/>
    <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 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>
    <version>3.5.2.6</version>
    <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>
```

Sample Response from a Credit Card Return Transaction

If the response code is 0, the transaction was successful.

```
<transaction-response>
        <orderID>C0A8C866:0119C7DF2702:E1D7:00E3FDA6</orderID>
        <referenceNum>846392236</referenceNum>
        <transactionID>32523465466</transactionID>
        <transactionTimestamp>1210664606</transactionTimestamp>
        <responseCode>0</responseCode>
        <avsResponseCode />
        <cvvResponseCode />
        <responseMessage>CAPTURED</responseMessage>
        <processorCode>A</processorCode>
        <processorMessage>APPROVED</processorMessage>
       <creditCardScheme>Visa</ creditCardScheme >
        <errorMessage/>
</transaction-response>
```

Partial returns are allowed. That is, the amount of the return can be any amount up to the full amount of the order.

Please note: Returns must **always** be associated with a previous order. If you have a need to perform credit transactions that are NOT associated with a previous order, use the <credit> method. We recommend using return transactions rather than credits in most situations and limiting user access to credit transactions.

Processing a Credit Card Credit

The <credit> method allows a specified amount to be refunded to the credit card account. We recommend using return transactions rather than credits in most situations and limiting user access to credit transactions.

(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>
  <credit>
    <orderID/>
    <invoiceNumber/>
     <referenceNum/>
    <ipAddress/>
    <billing/>
    <transactionDetail>
     <payType>
       <creditCard>
          <number/>          <!-- Required for keyed credit card transactions -->
          <expMonth/>        <!-- Required for keyed credit card transactions -->
          <expYear/>         <!-- Required for keyed credit card transactions -->
        <cvvInd/>
        <cvvNumber/> <!-- Required if Heartland is the processor -->
<!-- track data is required for swiped credit card transactions. We recommend sending both track1 and track2. If
the merchant is using an encrypted device, set encrypted = "Y" for all track data that is sent. -->
          <track1Data></track1Data>
          <track2Data></track2Data>
          <eCommInd>retail</eCommInd>
       </creditCard>
     </payType>
    </transactionDetail>
    <payment>
        <chargeTotal/>
     <salesTaxTotal/>
       <!-- sales tax on the order. If included, must contain a numeric value -->
     <currencyCode/>
       <!-- if included, must contain a value. If NOT included, defaults to USD --
    </payment>
    <processorID/>
  </credit>
 </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
    <orderID/>
    <referenceNum/>
    <transactionID/>
    <transactionTimestamp/>
    <responseCode/>
    <responseMessage/>
    <processorCode/>
    <processorMessage/>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <partiallyApprovedAmount/>
    <creditCardScheme/>
    <accountBalance/>
    <errorMessage/>
</transaction-response>
```

Or

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

Performing Purchasing Card Transactions

Purchasing cards are only currently supported on the Chase Paymentech (Levels 2 & 3) and First Data (Level 2 only) platforms. If you are using any other processor, you can send transactions as purchasing card transactions, but they will be processed as regular credit card transactions.

Purchase cards are only issued as Credit Cards. There is no Level 2 or Level 3 processing for Debit Cards or other types of cards.

The same methods you use for credit card transactions are also used for purchasing card transactions; however, the fields required for purchasing card transactions are different than standard credit card transactions—also it depends on whether you wish to do Level 2 or Level 3 purchasing card processing.

Level 2

To process a purchasing card at Level 2, the sales tax, tax indicator, purchase order number, and shipping postal code are required. Note that you also need to indicate that the purchasing card level you wish to process at is Level 2 by setting the <pcLevel> field to "2". Fields specifically used for Level 2 purchase cards are noted below in **bold blue text**.

INPUT XML Structure for a Level 2 purchasing card pre-authorization:

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

```
                <country> </country>
                <phone></phone>
                <email> </email>
            </billing>
            <shipping>
                <name> </name>
                <address> </address>
                <address2 />
                <city> </city>
                <state> </state>
                <postalcode></postalcode>
                <country> </country>
                <phone></phone>
                <email> </email>
            </shipping>
            <transactionDetail>
                <payType>
                    <creditCard>
                        <number></number>          <!-- Required for keyed credit card transactions -->
                        <expMonth></expMonth>      <!-- Required for keyed credit card transactions -->
                        <expYear></expYear>        <!-- Required for keyed credit card transactions -->
                        <cvvInd />
                        <cvvNumber></cvvNumber>
                        <deviceType/>
                        <deviceKSN>
                        <track1Data encrypted="Y"/>
        <!-- track data is required for swiped credit card transactions. We recommend sending both track1 and
track2. If the merchant is using an encrypted device, set encrypted = “Y” for all track data that is sent. -->
                        <track2Data encrypted="Y"/>
                        <eCommInd> </eCommInd>
                    </creditCard>
                </payType>
            </transactionDetail>
            <payment>
                <chargeTotal></chargeTotal>
                <salesTaxTotal></salesTaxTotal>
            </payment>
            <purchaseCardDetail>
                <AmexTransactionAdviceAddendum1/>
                <AmexTransactionAdviceAddendum2/>
                <AmexTransactionAdviceAddendum3/>
                <AmexTransactionAdviceAddendum4/>
                <PCLevel>2</PCLevel>
                <PCOrderNumber></PCOrderNumber>
                <TaxIndicator></TaxIndicator>
                <SDMerchantName/>
                <SDProductDescription/>
                <SDMerchantCity/>
                <SDMerchantPhone/>
                <SDMerchantURL/>
                <SDMerchantEmail/>
            </purchaseCardDetail>
        </auth>
    </order>
```

</transaction-request>

Level 3

To qualify for Level 3 purchasing card rates, line items, product codes, item description, unit price, unit quantities, and ship-to postal data are required in addition to the Level 2 data.

If you are doing an authorization, then capture, you can capture the item information in the capture transaction. If you are doing a sale transaction (both auth and capture in one step), you should include all the information required for Level 3 purchasing cards. One <item> tag for each item in the order should be included—you must also indicate the number of items included via the itemcount; e.g.: **<itemList**

itemCount="2">.

Note that you also need to indicate that the purchasing card level you wish to process at is Level 3 by setting the <pcLevel> field to "3". Fields specifically used for Level 3 purchase cards are noted below in blue text and Level 3 specifc fields that are required are in **bold blue text**.

INPUT XML Structure for a Level 3 purchasing card pre-authorization:

```
<?xml version="1.0"?>
<transaction-request>
  <verification>
    <merchantId></merchantId>
    <merchantKey></merchantKey>
  </verification>
  <order>
    <auth>
      <referenceNum></referenceNum>
      <ipAddress></ipAddress>
      <invoiceNumber/>
      <billing>
        <name></name>
        <address> </address>
        <address2/>
        <city></city>
        <state></state>
        <postalcode></postalcode>
        <country> </country>
        <phone></phone>
        <email></email>
      </billing>
      <shipping>
        <name></name>
        <address></address>
        <address2 />
        <city></city>
        <state></state>
        <postalcode></postalcode>
```

```
        <country></country>
        <phone></phone>
        <email></email>
      </shipping>
      <transactionDetail>
        <payType>
          <creditCard>
              <number></number>              <!-- Required for keyed credit card transactions -->
              <expMonth></expMonth>          <!-- Required for keyed credit card transactions -->
              <expYear></expYear>            <!-- Required for keyed credit card transactions -->
              <cvvInd />
              <cvvNumber></cvvNumber>
              <deviceType/>
              <deviceKSN
              <track1Data encrypted="Y"/> <!—track data is required for swiped credit card transactions. We
recommend sending both
track1 and track2. If the merchant is using an encrypted device, set encrypted = “Y” for all track data that is
sent. -->
              <track2Data encrypted="Y"/>
              <eCommInd></eCommInd>
          </creditCard>
        </payType>
      </transactionDetail>
      <payment>
        <chargeTotal></chargeTotal>
        <salesTaxTotal></salesTaxTotal>
      </payment>
      <purchaseCardDetail>
        <AmexTransactionAdviceAddendum1/>
        <AmexTransactionAdviceAddendum2/>
        <AmexTransactionAdviceAddendum3/>
        <AmexTransactionAdviceAddendum4/>
        <PCLevel>3</PCLevel>
        <PCOrderNumber></PCOrderNumber>
        <TaxIndicator></TaxIndicator>
        <SDMerchantName/>
        <SDProductDescription/>
        <SDMerchantCity/>
        <SDMerchantPhone/>
        <SDMerchantURL/>
        <SDMerchantEmail/>
        <PC3Detail>
          <PC3AlternateTaxAmount></PC3AlternateTaxAmount>
          <PC3AlternateTaxID></PC3AlternateTaxID>
          <PC3VatTaxRate></PC3VatTaxRate>
          <PC3VatTaxAmount></PC3VatTaxAmount>
          <PC3DiscountAmount></PC3DiscountAmount>
          <PC3ShipFromZip></PC3ShipFromZip>
          <PC3DestCountryCode></PC3DestCountryCode>
          <PC3DutyAmount></PC3DutyAmount>
          <PC3FreightAmount></PC3FreightAmount>
        </PC3Detail>
      </purchaseCardDetail>
    </auth>
```

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

INPUT XML Structure for a Level 3 purchasing card post-authorization:

Note that you must include item information in the post-authorization.

```
<?xml version="1.0" encoding="UTF-8" ?>
<transaction-request>
    <verification>
        <merchantId></merchantId>
        <merchantKey></merchantKey>
    </verification>
    <order>
        <capture>
            <orderID></orderID>
            <referenceNum></referenceNum>
            <ipAddress></ipAddress>
            <PC3Detail>
                <itemList itemCount="2">
                    <item>
                        <itemIndex>1</itemIndex>
                        <itemDescription></itemDescription>
                        <itemProductCode></itemProductCode>
                        <itemQuantity></itemQuantity>
                        <itemUnitOfMeasurement/>
                        <itemTaxAmount></itemTaxAmount>
                        <itemTaxRate></itemTaxRate>
                        <itemTotalAmount></itemTotalAmount>
                        <itemDiscountAmount/>
                        <itemCommodityCode></itemCommodityCode>
                        <itemUnitCost></itemUnitCost>
                        <itemTaxIncluded></itemTaxIncluded>
                        <itemTaxType></itemTaxType>
                        <itemDiscountIndicator/>
                        <itemDebitIndicator/>
                    </item>
                    <item>
                        <itemIndex>2</itemIndex>
                        <itemDescription></itemDescription>
                        <itemProductCode></itemProductCode>
                        <itemQuantity></itemQuantity>
                        <itemUnitOfMeasurement/>
                        <itemTaxAmount></itemTaxAmount>
                        <itemTaxRate></itemTaxRate>
                        <itemTotalAmount></itemTotalAmount>
                        <itemDiscountAmount/>
                        <itemCommodityCode></itemCommodityCode>
                        <itemUnitCost></itemUnitCost>
                        <itemTaxIncluded></itemTaxIncluded>
                        <itemTaxType></itemTaxType>
                        <itemDiscountIndicator/>
                        <itemDebitIndicator/>
                    </item>
```

```
                </itemList>
            </PC3Detail>
        <payment>
            <chargeTotal></chargeTotal>
        </payment>
    </capture>
    </order>
</transaction-request>
```

INPUT XML Structure for a Level 3 purchasing card sale:

Processes both the pre-authorization and the post-authorization in one step.

```
<transaction-request>
    <verification>
        <merchantId></merchantId>
        <merchantKey></merchantKey>
    </verification>
    <order>
        <sale>
            <referenceNum></referenceNum>
            <ipAddress></ipAddress>
            <invoiceNumber/>
            <billing>
                <name></name>
                <address></address>
                <address2 />
                <city></city>
                <state></state>
                <postalcode></postalcode>
                <country></country>
                <phone></phone>
                <email></email>
            </billing>
            <shipping>
                <name></name>
                <address></address>
                <address2 />
                <city></city>
                <state></state>
                <postalcode></postalcode>
                <country></country>
                <phone></phone>
                <email></email>
            </shipping>
            <transactionDetail>
                <payType>
                    <creditCard>
                        <number></number>          <!-- Required for keyed credit card transactions -->
                        <expMonth></expMonth>      <!-- Required for keyed credit card transactions -->
                        <expYear></expYear>        <!-- Required for keyed credit card transactions -->
                        <cvvInd />
                        <cvvNumber></cvvNumber>
<!—track data is required for swiped credit card transactions. We recommend sending both
```

track1 and track2. If the merchant is using an encrypted device, set encrypted = “Y” for all track data that is sent. -->

```
                        <track1Data />
                        <track2Data />
                        <eCommInd>eci</eCommInd>
                    </creditCard>
                </payType>
            </transactionDetail>
            <payment>
                <chargeTotal></chargeTotal>
                <salesTaxTotal>1.00</salesTaxTotal>
            </payment>
            <purchaseCardDetail>
                <AmexTransactionAdviceAddendum1/>
                <AmexTransactionAdviceAddendum2/>
                <AmexTransactionAdviceAddendum3/>
                <AmexTransactionAdviceAddendum4/>
                <PCLevel>3</PCLevel>
                <PCOrderNumber></PCOrderNumber>
                <TaxIndicator></TaxIndicator>
                <PC3Detail>
                    <PC3AlternateTaxAmount></PC3AlternateTaxAmount>
                    <PC3AlternateTaxID></PC3AlternateTaxID>
                    <PC3VatTaxRate></PC3VatTaxRate>
                    <PC3VatTaxAmount></PC3VatTaxAmount>
                    <PC3DiscountAmount></PC3DiscountAmount>
                    <PC3ShipFromZip></PC3ShipFromZip>
                    <PC3DestCountryCode> </PC3DestCountryCode>
                    <PC3DutyAmount></PC3DutyAmount>
                    <PC3FreightAmount></PC3FreightAmount>
                    <itemList itemCount="2">
                        <item>
                            <itemIndex></itemIndex>
                            <itemDescription></itemDescription>
                            <itemProductCode></itemProductCode>
                            <itemQuantity></itemQuantity>
                            <itemUnitOfMeasurement></itemUnitOfMeasurement>
                            <itemTaxAmount></itemTaxAmount>
                            <itemTaxRate></itemTaxRate>
                            <itemTotalAmount></itemTotalAmount>
                            <itemDiscountAmount></itemDiscountAmount>
                            <itemCommodityCode></itemCommodityCode>
                            <itemUnitCost></itemUnitCost>
                            <itemTaxIncluded></itemTaxIncluded>
                            <itemTaxType></itemTaxType>
                            <itemDiscountIndicator>
                            </itemDiscountIndicator>
                            <itemDebitIndicator>
                            </itemDebitIndicator>
                        </item>
                        <item>
                            <itemIndex></itemIndex>
                            <itemDescription></itemDescription>
                            <itemProductCode></itemProductCode>
```

```
                        <itemQuantity></itemQuantity>
                        <itemUnitOfMeasurement></itemUnitOfMeasurement>
                        <itemTaxAmount></itemTaxAmount>
                        <itemTaxRate></itemTaxRate>
                        <itemTotalAmount></itemTotalAmount>
                        <itemDiscountAmount></itemDiscountAmount>
                        <itemCommodityCode></itemCommodityCode>
                        <itemUnitCost></itemUnitCost>
                        <itemTaxIncluded></itemTaxIncluded>
                        <itemTaxType></itemTaxType>
                        <itemDiscountIndicator>
                        </itemDiscountIndicator>
                        <itemDebitIndicator>
                        </itemDebitIndicator>
                    </item>
                </itemList>
            </PC3Detail>
        </purchaseCardDetail>
    </sale>
</order>
</transaction-request>
```

OUTPUT XML Structure:

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

Performing an Electronic Check Transaction

There are two ways to electronically process a check transaction:

1. Use the `<sale>` method with the `<ach>` tag to submit an ACH transaction to the ACH network for processing. You can include a captured check image in the `<ach>` tag as well as quite a bit of other information from the check; however the image you send with an <ach> tag will NOT get submitted to the banking system for processing. The transaction will be processed as an ACH transaction.

2. Use the `<sale>` method with the `<rdc>` tag to electronically process a captured check image.
   RDC stands for Remote Data Capture—it is the term used when submitting a check image in lieu of a paper check (called Check 21 in the banking world). By using RDC, typically check processing fees are reduced.

See NACHA's ACH eCheck Primer for further information on the difference between ACH and RDC transactions.

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

INPUT XML Structure using the <ach> tag:

```
<transaction-request>
  <verification>
        <merchantId/>
        <merchantKey/>
  </verification>
  <order>
    <sale>
      <referenceNum/>
      <ipAddress/> <!-- required for JHA -->
      <invoiceNumber/>
      <billing>
       <address/>
       <address2/>
       <city/>
       <state/>
       <postalcode/> <!-- must be 5 digits for echecks with JHA -->
       <country/>
       <phone/>
       <email/>
      </billing>
      <transactionDetail>
       <payType>
         <flagAsRecurring/>
         <firstRecurring/>
         <ach>
            <bankRoutingNumber/>
            <achAccountNumber/>
          <checkNumber/>
          <achAccountType/>
          <achBusinessName/>
          <achPaymentType/>
          <achEffectiveDate/>
          <MICR/>
          <bankName/>
          <bankState/>
          <frontImage/>
          <backImage/>
         </ach>
```

```
        </payType>
      </transactionDetail>
      <payment>
          <chargeTotal/>
       <shippingTotal/>
          <!-- shipping on the order. If included, must contain a numeric value -->
       <salesTaxTotal/>
          <!-- sales tax on the order. If included, must contain a numeric value -->
      </payment>
   </sale>
  </order>
</transaction-request>
```

INPUT XML Structure using the <rdc> tag:

```
<transaction-request>
  <verification>
        <merchantId/>
        <merchantKey/>
  </verification>
  <order>
    <sale>
       <referenceNum/>
       <ipAddress/>
     <invoiceNumber/>
       <billing>
       <address/>
       <address2/>
       <city/>
          <state/>
         <postalcode/> <!-- must be 5 digits for echecks with JHA -->
       <country/>
       <phone/>
       <email/>
      </billing>
      <transactionDetail>
       <payType>
        <flagAsRecurring/>
          <firstRecurring/>
          <rdc>
            <MICR/> <!-- must be included for RDC transactions -->
            <frontImage/> <!-- must be included for RDC transactions -->
            <backImage/> <!-- must be included for RDC transactions -->
          </rdc>
       </payType>
      </transactionDetail>
      <payment>
          <chargeTotal/>
       <shippingTotal/>
          <!-- shipping on the order. If included, must contain a numeric value -->
       <salesTaxTotal/>
          <!-- sales tax on the order. If included, must contain        a numeric value -->
      </payment>
```

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

OUTPUT XML Structure:

```
<transaction-response>
      <referenceNum/>
      <transactionID/>
      <transactionTimestamp/>
      <responseCode/>
      <responseMessage/>
      <processorReferenceNumber/>
      <processorTransactionID/>
      <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>          <!-- required for JHA -->
                  <billing>
                        <name>Tom Customer</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>customer@example.com</email>
                  </billing>
                  <transactionDetail>
                        <payType>
```

```
                    <ach>
                        <bankRoutingNumber>111111111</bankRoutingNumber>
                        <achAccountNumber>12345678</achAccountNumber>
                        <checkNumber>1200</checkNumber>
                        <achAccountType>PC</achAccountType>
                        <achBusinessName> ... </achBusinessName>
                        <achPaymentType> ... </achPaymentType>
                        <achEffectiveDate/>
                        <MICR> ... </MICR>
                        <bankName> ... </bankName>
                        <bankState> ... </bankState>
                        <frontImage> ... </frontImage>
                        <backImage> ... </backImage>
                    </ach>
                </payType>
            </transactionDetail>
            <payment>
                <chargeTotal>195.00</chargeTotal>
                <salesTaxTotal>0.00</salesTaxTotal> <!-- sales tax on the order. If included, must contain a
numeric value -->
                <shippingTotal>0.00</shippingTotal>
                <!-- shipping on the order. If included, must contain a numeric value -->
            </payment>
        </sale>
        <clientData>
            <comments/>
        </clientData>
    </order>
</transaction-request>
```

Sample Accepted Response from an Electronic Check Transaction

If the response code is 0, the transaction was successful.

```
<transaction-response>
    <referenceNum>846392231</referenceNum>
    <transactionID>32523465461</transactionID>
    <transactionTimestamp>1210664601</transactionTimestamp>
    <responseCode>0</responseCode>
    <responseMessage>ACCEPTED</responseMessage>
    <avsResponseCode />
    <cvvResponseCode />
    <processorCode />
    <processorMessage />
    <processorReferenceNumber/>
    <processorTransactionID/>
    <errorMessage/>
</transaction-response>
```

Saving a Checking Account on File with a Check Sale Transaction

Saving a Checking Account on File along with a Sale transaction

When you perform a sale transaction, you can also save the account information on file in the same operation. This returns a payment token which can be sent for future transactions with this customer and checking account in lieu of the account information. Saving an account on file can also be performed in a separate operation—see the ***Saving a Card on File*** section later in this document for further details, posting URL and XML structure.

Before you can save a card or account on file for a customer, you must first add the customer. See the Adding and Deleting Customers section for instructions.

- Note: Starting in July, 2017, a Customer record is no longer a pre-requisite for adding a payment on file.

INPUT XML Structure for a Check Sale with Save Account on File

```
<?xml version="1.0" encoding="UTF-8" ?>
<transaction-request>
    <verification>
        <merchantId/>
        <merchantKey/>
    </verification>
    <order>
        <sale>
            <referenceNum/>
            <ipAddress/>
            <consumerPrimaryID/>
            <invoiceNumber/>
            <billing>
                <name></name>
                <address/>
                <address2/>
                <city></city>
                <state></state>
                <postalcode/>
                <country/>
                <phone/>
                <email/>
            </billing>
            <shipping>
                <name/>
                <address/>
                <address2/>
                <city/>
                <state/>
                <postalcode/>
                <country/
                <phone/>
                <email/>
            </shipping>
            <transactionDetail>
```

```
                <payType>
                    <ach>
                        <bankRoutingNumber></bankRoutingNumber>
                        <achAccountNumber></achAccountNumber>
                    </ach>
                </payType>
            </transactionDetail>
            <payment>
                <chargeTotal></chargeTotal>
            </payment>
            <saveOnFile>
                <customerToken></customerToken> <-- populate this field with the
                customerId returned from the add-consumer function -->
                <onFileEndDate></onFileEndDate>
                <onFilePermissions></onFilePermissions>
                <onFileComment></onFileComment>
                <onFileMaxChargeAmount></onFileMaxChargeAmount>
            </saveOnFile>
        </sale>
    </order>
</transaction-request>
```

Sample Response Code from Check Sale with Save Account on File

If the response code is **0**, the transaction was successful. (Indicators that the transaction succeeded are shown in **bold blue**.) The <token> sent back in the <save-onfile> tag is the payment token to be used in lieu of the checking account data on subsequent transactions with this customer and account.

```
<transaction-response>
    <authCode>123456</authCode>
    <orderID>0AF90446:012BCCFA0D11:44A0:018F4F91</orderID>
    <referenceNum>1</referenceNum>
    <transactionID>876268</transactionID>
    <transactionTimestamp>1287634160</transactionTimestamp>
    <responseCode>0</responseCode>
    <responseMessage>ACCEPTED</responseMessage>
    <avsResponseCode>YYY</avsResponseCode>
    <cvvResponseCode>M</cvvResponseCode>
    <processorCode>A</processorCode>
    <processorMessage>APPROVED</processorMessage>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <errorMessage />
    <save-onfile>
        <token>TaFHDt1pBKoA=</token>
    </save-onfile>
</transaction-response>
```

If the transaction went through, but there was a problem with saving the card on file, the <save-onfile> tag will contain an <error> tag with an explanatory message.

Following is an example:

```
<transaction-response>
    <authCode>123456</authCode>
    <orderID>0AF90446:012BCCFA0D11:44A0:018F4F91</orderID>
    <referenceNum>1</referenceNum>
<transactionID>876268</transactionID>
    <transactionTimestamp>1287634160</transactionTimestamp>
    <responseCode>0</responseCode>
    <responseMessage>ACCEPTED</responseMessage>
    <avsResponseCode>YYY</avsResponseCode>
    <cvvResponseCode>M</cvvResponseCode>
    <processorCode>A</processorCode>
<processorMessage>APPROVED</processorMessage>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <errorMessage />
    <save-onfile>
        <error>Unable to find consumer.</error>
    </save-onfile>
</transaction-response>
```

Performing a PIN Debit Transaction

PIN debit is used with a device where the customer inputs their Personal Identification Number (PIN) using a PIN pad at the point of sale. To use the Paymentsite XML API for a PIN debit transaction, you use the <debitSale> method. You will need to integrate your software with the PIN pad as well.

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

INPUT XML Structure:

```
<transaction-request>
 <verification>
       <merchantId></merchantId>
       <merchantKey></merchantKey>
 </verification>
 <order>
  <debitSale>
      <referenceNum></referenceNum>
     <ipAddress></ipAddress>
     <invoiceNumber/>
     <billing>
       <name></name>
       <address></address>
       <address2/>
```

```
        <city></city>
        <state></state>
        <postalcode></postalcode>
        <country></country>
        <phone></phone>
        <email></email>
      </billing>
    <shipping>
      <name> </name>
        <address> </address>
        <address2/>
        <city> </city>
        <state> </state>
        <postalcode></postalcode>
        <country> </country>
        <phone></phone>
        <email> </email>
    </shipping>
      <transactionDetail>
        <payType>
          <flagAsRecurring/>
            <firstRecurring/>
            <debitCard>
              <deviceType/>
              <deviceKSN />
                <track1Data encrypted="Y"/>
                  <!-- If the merchant is using an encrypted device for swiping the card, set encrypted = “Y” for all
track data that is sent. -->
                <track2Data encrypted="Y"/>
                <pinBlock> </pinBlock>
                <keySerialNo> </keySerialNo>
                <eCommInd>retail</eCommInd>
            </debitCard>
        </payType>
      </transactionDetail>
      <payment>
          <chargeTotal></chargeTotal>
        <salesTaxTotal></salesTaxTotal>
          <!-- sales tax on the order. If included, must contain a numeric value -->
        <shippingTotal></shippingTotal>
          <!-- shipping on the order. If included, must contain a numeric value -->
        <cashBack></cashBack>
          <!-- If included, must contain a numeric value -->
      </payment>
   </debitSale>
  </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
      <authCode/>
      <orderID/>
```

```
    <referenceNum/>
    <transactionID/>
    <transactionTimestamp/>
    <responseCode/>
    <responseMessage/>
    <avsResponseCode/>
    <cvvResponseCode/>
    <processorCode/>
    <processorMessage/>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <errorMessage />
    <gatewayDebitNetworkID/>
    <creditCardScheme/>
</transaction-response>
Or:
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <api-error>
        <errorCode></errorCode>
        <errorMsg></errorMsg>
    </api-error>
```

Sample Input Code for a PIN Debit 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>
        <debitSale>
            <referenceNum>1</referenceNum>
            <ipAddress>123.123.123.123</ipAddress>
            <billing>
                <name>Tom Customer</name>
                <address>123 1st Street</address>
                <address2/>
                <city>Moorpark</city>
                <state>CA</state>
                <postalcode>91307</postalcode>
                <country>US</country>
                <phone>8181231234</phone>
                <email>customer@example.com</email>
            </billing>
            <shipping>
                <name>Tom Customer</name>
                <address>123 1st Street</address>
                <address2/>
```

```
                <city>Moorpark</city>
                <state>CA</state>
                <postalcode>91307</postalcode>
                <country>US</country>
                <phone>8181231234</phone>
                <email>customer@example.com</email>
            </shipping>
            <transactionDetail>
                <payType>
                    <flagAsRecurring/>
                    <firstRecurring/>
                    <debitCard>
                        <track1Data>1234567124351258312</track1Data>
                        <track2Data>12376128931682910239821</track2Data>
                        <pinBlock>273A61283641823</pinBlock>
                        <keySerialNo>10002320930000</keySerialNo>
                        <eCommInd>retail</eCommInd>
                    </debitCard>
                </payType>
            </transactionDetail>
            <payment>
                <chargeTotal>3.00</chargeTotal>
                <salesTaxTotal>0.38</salesTaxTotal>
                <!-- sales tax on the order. If included, must contain a numeric value -->
                <shippingTotal>3.03</shippingTotal>
                <!-- shipping on the order. If included, must contain a numeric value -->
                <cashBack>2.05</cashBack>
                <!-- amount of cash back the customer wants. If included, must contain a numeric value -->
            </payment>
        </debitSale>
    </order>
</transaction-request>
```

Sample Accepted Response from a PIN Debit Transaction

If the response code is 0, the transaction was successful.

```
<transaction-response>
    <authCode>123456</authCode>
    <orderID>0AF90446:0125E8E41FC7:49A5:001145CC</orderID>
    <referenceNum>1</referenceNum>
    <transactionID>745032</transactionID>
    <transactionTimestamp>1262332682</transactionTimestamp>
    <responseCode>0</responseCode>
    <responseMessage>CAPTURED</responseMessage>
    <avsResponseCode>YYY</avsResponseCode>
    <cvvResponseCode>M</cvvResponseCode>
    <processorCode>A</processorCode>
    <processorMessage>APPROVED</processorMessage>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <creditCardScheme>Visa<creditCardScheme/>
   <gatewayDebitNetworkID>060004</gatewayDebitNetworkID>
    <errorMessage />
```

</transaction-response>

Performing a PIN Debit Return

To use the Paymentsite XML API for a PIN debit return, you use the <debitReturn> method. This will refund monies back to the customer's account.

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

INPUT XML Structure:

```
<?xml version="1.0" encoding="UTF-8" ?>
    <transaction-request>
    <verification>
        <merchantId></merchantId>
        <merchantKey> </merchantKey>
    </verification>
    <order>
        <debitReturn>
            <orderID></orderID>
            <referenceNum></referenceNum>
            <ipAddress></ipAddress>
            <invoiceNumber/>
            <billing>
                <name> </name>
                <address> </address>
                <address2/>
                <city></city>
                <state> </state>
                <postalcode></postalcode>
                <country></country>
                <phone></phone>
                <email> </email>
            </billing>
            <payment>
                <chargeTotal></chargeTotal>
                <salesTaxTotal></salesTaxTotal>
                <shippingTotal></shippingTotal>
                <cashBack></cashBack>
                <!-- amount of cash back the customer wants. If included, must contain a numeric value -->
            </payment>
            <transactionDetail>
                <payType>
                    <debitCard>
                        <deviceType/>
                        <deviceKSN />
                        <track1Data encrypted="Y"/> <!-- If the merchant is using an encrypted device
for swiping the card, set encrypted = "Y" for all track data that is sent. -->
                        <track2Data encrypted="Y"/>
                        <pinBlock></pinBlock>
                        <keySerialNo></keySerialNo>
                        <eCommInd> </eCommInd>
```

```
                    </debitCard>
                </payType>
            </transactionDetail>
        </debitReturn>
    </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
    <referenceNum/>
    <transactionID/>
    <transactionTimestamp/>
    <responseCode/>
    <responseMessage/>
    <avsResponseCode />
    <cvvResponseCode />
    <processorCode/>
    <processorMessage/>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <errorMessage/>
    <gatewayDebitNetworkID/>
</transaction-response>
Or:
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <api-error>
    <errorCode></errorCode>
    <errorMsg></errorMsg>
</api-error>
```

Sample Input Code for a PIN Debit 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>
      <debitReturn>
          <orderID>123456789123456789</orderID>
          <referenceNum>1</referenceNum>
        <ipAddress>123.123.123.123</ipAddress>
        <invoiceNumber>19832</invoiceNumber>
        <billing>
          <name>Tom Customer</name>
```

```
            <address>123 1st Street</address>
            <address2/>
            <city>Moorpark</city>
            <state>CA</state>
            <postalcode>91307</postalcode>
            <country>US</country>
            <phone>8181231234</phone>
            <email>customer@example.com</email>
          </billing>
          <payment>
            <chargeTotal>30.08</chargeTotal>
            <salesTaxTotal>10.06</salesTaxTotal>
              <!-- sales tax on the order. If included, must contain a numeric value -->
            <shippingTotal>20.04</shippingTotal>
              <!-- shipping on the order. If included, must contain a numeric value -->
            <cashBack>2.00</cashBack>
              <!-- amount of cash back the customer wants. If included, must contain a numeric value -->
          </payment>
          <transactionDetail>
          <payType>
            <debitCard>
                <track1Data>%1239712938179</track1Data>
                <track2Data>;1i2736127860239?</track2Data>
                <pinBlock>12A29387429324</pinBlock>
                <keySerialNo>100010290000010000</keySerialNo>
                <eCommInd>retail</eCommInd>
            </debitCard>
          </payType>
        </transactionDetail>
      </debitReturn>
    </order>
</transaction-request>
```

Sample Accepted Response from a PIN Debit Return Transaction

If the response code is 0, the transaction was successful.

```
<transaction-response>
    <referenceNum>846392231</referenceNum>
    <transactionID>32523465461</transactionID>
    <transactionTimestamp>1210664601</transactionTimestamp>
    <responseCode>0</responseCode>
    <responseMessage>ACCEPTED</responseMessage>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <errorMessage/>
    <gatewayDebitNetworkID>060004</gatewayDebitNetworkID>
</transaction-response>
```

Voiding a PIN Debit Transaction

To void a PIN debit transaction, you use the <debitVoid> method. This will void a transaction that has not yet settled.

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

INPUT XML Structure:

```
<transaction-request>
    <verification>
        <merchantId></merchantId>
        <merchantKey></merchantKey>
    </verification>
    <order>
        <debitVoid>
            <transactionID></transactionID>
            <referenceNum></referenceNum>
            <ipAddress></ipAddress>
            <transactionDetail>
                <payType>
                    <debitCard>
                        <deviceType/>
                        <deviceKSN />
                        <track1Data encrypted="Y"/>
                        <!-- If the merchant is using an encrypted device for swiping the card, set encrypted
= "Y" for all track data that is sent. -->
                        <track2Data encrypted="Y"/>
                        <pinBlock></pinBlock>
                        <keySerialNo></keySerialNo>
                        <eCommInd>retail</eCommInd>
                    </debitCard>
                </payType>
            </transactionDetail>
        </debitVoid>
    </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
    <referenceNum/>
    <transactionID/>
    <transactionTimestamp/>
    <responseCode/>
    <responseMessage/>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <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 a PIN Debit Void

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>
        <debitVoid>
            <transactionID>123456789123456789</transactionID>
            <referenceNum>1</referenceNum>
            <ipAddress>123.123.123.123</ipAddress>
            <transactionDetail>
                <payType>
                    <debitCard>
                        <track1Data>%1239712938179</track1Data>
                        <track2Data>;1i2736127860239?</track2Data>
                        <pinBlock>12A29387429324</pinBlock>
                        <keySerialNo>100010290000010000</keySerialNo>
                        <eCommInd>retail</eCommInd>
                    </debitCard>
                </payType>
            </transactionDetail>
        </debitVoid>
    </order>
</transaction-request>
```

Sample Accepted Response from a PIN Debit Return Transaction

If the response code is 0, the transaction was successful.

```
<transaction-response>
    <authCode />
    <orderID />
    <referenceNum />
    <transactionID>94967</transactionID>
    <transactionTimestamp />
    <responseCode>0</responseCode>
    <responseMessage>VOIDED</responseMessage>
    <avsResponseCode />
    <cvvResponseCode />
    <processorCode>A</processorCode>
```

```
    <processorMessage>APPROVED</processorMessage>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <errorMessage />
</transaction-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. Different processors have different rules for the window between when at Sale is submitted and when it can be Voided, and this period can be as short as 20 minutes.

(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>
    <authCode />
    <orderID />
    <referenceNum />
    <transactionID/>
    <transactionTimestamp/>
    <responseCode/>
    <responseMessage/>
    <avsResponseCode />
    <cvvResponseCode />
    <processorCode/>
    <processorMessage/>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <partiallyApprovedAmount/>
<accountBalance/>
```

```
<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 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

If the response code is 0, the transaction was successfully voided.

```
<transaction-response>
    <authCode />
    <orderID />
    <referenceNum />
    <transactionID>976661</transactionID>
    <transactionTimestamp />
    <responseCode>0</responseCode>
    <responseMessage>VOIDED</responseMessage>
    <avsResponseCode />
    <cvvResponseCode />
    <processorCode>A</processorCode>
    <processorMessage>APPROVED</processorMessage>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <errorMessage />
</transaction-response>
```

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/>
      <invoiceNumber/>
      <billing>
        <name/>
        <address/>
        <address2/>
        <city/>
        <state/>
        <postalcode/>
        <country/>
        <phone/>
        <email/>
      </billing>
      <shipping>
        <name/>
        <address/>
        <address2/>
        <city/>
        <state/>
        <postalcode/>
        <country/>
        <phone/>
        <email/>
      </shipping>
      <transactionDetail>
        <payType>
          <onFile>
            <consumerPrimaryId/>
            <consumerExternalId/>
```

```
            <token/>
          </onFile>
        </payType>
      </transactionDetail>
      <payment>
        <chargeTotal/>
        <salesTaxTotal/>
          <!-- sales tax on the order. If included, must contain a numeric value -->
        <shippingTotal/>
          <!-- shipping on the order. If included, must contain a numeric value -->
        <currencyCode/>
          <!-- if included, must contain a value. If NOT included, defaults to USD -->
      </payment>
    </sale>
    <clientData>
      <comments/>
    </clientData>
  </order>
</transaction-request>
```

OUTPUT XML Structure:

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

Or, if there is an error:

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

Europay-Visa-Mastercard (EMV) Transactions

EMV is currently supported only on the First Data platform. If you are using any other processor, please do not attempt to send any EMV requests. Please refer to the documentation at https://www.emvco.com/specifications.aspx?id=223 for details on the EMV specification.

Note that all the normal transaction types may be used for a EMV implementation—what relates it to EMV is the inclusion of emv order types (i.e. emvSale, emvAuth, emvSaleForce , emvCapture, emvReturn, emvCredit, emvVoid , emvReverse, emvDebitSale, emvDebitReturn or emvDebitVoid) and EMV related data fields under `<payType><creditCard>` or `<payType><debitCard>`, shown in **bold blue font**.

EMV Credit Card Sale or Authorize

(Note: All "Required Fields" are identified by bold red font)

(Note: EMV related fields are in bold blue font)

INPUT XML Structure for a EMV Credit Card Sale Transaction:

```
<transaction-request>
  <verification>
    <merchantId/> <!-- Required field -->
    <merchantKey/> <!-- Required field -->
  </verification>
  <order>
    <emvSale>
      <referenceNum/> <!-- Required field -->
      <ipAddress/>
      <invoiceNumber/>
      <billing>
        <name/>
        <address/>
        <address2/>
        <city/>
        <state/>
        <postalcode/>
        <country/>
        <phone/>
        <email/>
      </billing>
      <shipping>
        <name/>
        <address/>
        <address2/>
        <city/>
        <state/>
        <postalcode/>
        <country/>
        <phone/>
        <email/>
      </shipping>
      <transactionDetail> <!--number,expMonth,and expYear OR track2 are Reqd fields-->
        <payType>
          <creditCard>
```

```
          <track1Data/>
          <track2Data/>
          <eCommInd/>
          <cardSequenceNumber>
          <emvDataRequest>
          <isFallBackToSwipe>
          <systemTraceNumber>
          <processorMid>
          <processorTid>
        </creditCard>
      </payType>
    </transactionDetail>
    <payment>
      <chargeTotal/> <!-- Required field -->
     <salesTaxTotal/>
 <shippingTotal/>
 <convenienceFee/>
 <currencyCode/>
    </payment>
   </emvSale>
   <clientData/>
 </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
   <authCode/>
   <orderID/>
   <referenceNum/>
   <transactionID/>
   <transactionTimestamp/>
   <responseCode/>
   <responseMessage/>
   <avsResponseCode/>
   <cvvResponseCode/>
   <processorCode/>
   <processorMessage/>
   <errorMessage/>
   <partiallyApprovedAmount/>
   <creditCardScheme/>
   <emvDataResponse/>  <!--Base64 Encoded -->
   <fileUpdateName/>
   <fileUpdateCreateDate/>
   <fileUpdateSize/>
   <fileUpdateCRC/>
   <fileUpdateFCode/>
</transaction-response>
```

Or

```
<api-error>
   <errorCode></errorCode>
   <errorMsg></errorMsg>
</api-error>
```

Sample Input Code for a EMV Credit Card Sale 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>123</merchantId> <!-- Required field -->
  <merchantKey>key</merchantKey> <!-- Required field -->
 </verification>
 <order>
  <emvSale>
   <referenceNum>1</referenceNum> <!-- Required field -->
   <ipAddress>192.123.123.123</ipAddress>
   <billing>
    <name>Tom Customer</name>
    <address>123 Street</address>
    <address2/>
    <city>Moorpark</city>
    <state>CA</state>
    <postalcode>91307</postalcode>
    <country>US</country>
    <phone>8181231234</phone>
    <email>customer@example.com</email>
   </billing>
   <shipping>
    <name>Tom Customer</name>
    <address>123 Street</address>
    <address2/>
    <city>Moorpark</city>
    <state>CA</state>
    <postalcode>91307</postalcode>
    <country>US</country>
    <phone>8181231234</phone>
    <email>customer@example.com</email>
   </shipping>
   <transactionDetail>
    <payType>
     <creditCard>
      <track1Data></track1Data>
      <track2Data>;1237612893168291=023343439821?</track2Data>
      <eCommInd>retail</eCommInd>
      <cardSequenceNumber>001</cardSequenceNumber>
      <emvDataRequest>Base64EncodedEMVRequestData</emvDataRequest>
```

```
        <isFallBackToSwipe>0</isFallBackToSwipe>
        <systemTraceNumber>123456</systemTraceNumber>
        <processorMid>1234567</processorMid>
        <processorTid>1234567</processorTid>
       </creditCard>
     </payType>
    </transactionDetail>
    <payment>
     <chargeTotal>23.00</chargeTotal> <!-- Required field -->
    </payment>
   </emvSale>
  </order>
</transaction-request>
```

Sample Response Code from a a EMV Credit Card Sale transaction

If the response code is 0, the transaction was successful.

```
<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/>
<creditCardScheme>Visa<creditCardScheme/>
<emvDataResponse>Base64EncodedEMVResponseData<emvDataResponse/>
<fileUpdateName/>
<fileUpdateCreateDate/>
<fileUpdateSize/>
<fileUpdateCRC/>
<fileUpdateFCode/>
</transaction-response>
```

EMV Credit Card Forced Sale

(Note: All "Required Fields" are identified by bold red font)

(Note: EMV related fields are in bold blue font)

INPUT XML Structure

```
<transaction-request>
 <verification>
  <merchantId/> <!-- Required field -->
  <merchantKey/> <!-- Required field -->
 </verification>
 <order>
```

```
  <emvSaleForce>
    <authCode/>
    <referenceNum/> <!-- Required field -->
    <ipAddress/>
    <invoiceNumber/>
    <billing>
      <name/>
      <address/>
      <address2/>
      <city/>
      <state/>
      <postalcode/>
      <country/>
      <phone/>
      <email/>
    </billing>
    <shipping>
      <name/>
      <address/>
      <address2/>
      <city/>
      <state/>
      <postalcode/>
      <country/>
      <phone/>
      <email/>
    </shipping>
    <transactionDetail> <!--number,expMonth,and expYear OR track2 are Reqd fields-->
      <payType>
        <creditCard>
          <track1Data/>
          <track2Data/>
          <eCommInd/>
          <cardSequenceNumber>
          <emvDataRequest>
          <isFallBackToSwipe>
          <systemTraceNumber>
          <processorMid>
          <processorTid>
        </creditCard>
      </payType>
    </transactionDetail>
    <payment>
      <chargeTotal/> <!-- Required field -->
     <salesTaxTotal/>
 <shippingTotal/>
 <convenienceFee/>
 <currencyCode/>
    </payment>
   </emvSaleForce>
   <clientData/>
  </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
   <authCode/>
   <orderID/>
   <referenceNum/>
   <transactionID/>
   <transactionTimestamp/>
   <responseCode/>
   <responseMessage/>
   <avsResponseCode/>
   <cvvResponseCode/>
   <processorCode/>
   <processorMessage/>
   <errorMessage/>
   <partiallyApprovedAmount/>
   <creditCardScheme/>
   <emvDataResponse/>  <!--Base64 Encoded -->
   <fileUpdateName/>
   <fileUpdateCreateDate/>
   <fileUpdateSize/>
   <fileUpdateCRC/>
   <fileUpdateFCode/>
</transaction-response>
```

Or

```
<api-error>
   <errorCode></errorCode>
   <errorMsg></errorMsg>
</api-error>
```

EMV Credit Card Capture

(Note: All "Required Fields" are identified by bold red font)

(Note: EMV related fields are in bold blue font)

INPUT XML Structure

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

```
      <chargeTotal/>
      <salesTaxTotal/>
 <shippingTotal/>
 <convenienceFee/>
 <currencyCode/>
     </payment>
   </emvCapture>
 </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
   <authCode/>
   <orderID/>
   <referenceNum/>
   <transactionID/>
   <transactionTimestamp/>
   <responseCode/>
   <responseMessage/>
   <avsResponseCode/>
   <cvvResponseCode/>
   <processorCode/>
   <processorMessage/>
   <errorMessage/>
   <partiallyApprovedAmount/>
   <creditCardScheme/>
   <emvDataResponse/>
   <fileUpdateName/>
   <fileUpdateCreateDate/>
   <fileUpdateSize/>
   <fileUpdateCRC/>
   <fileUpdateFCode/>
</transaction-response>
```

Or

```
<api-error>
   <errorCode></errorCode>
   <errorMsg></errorMsg>
</api-error>
```

EMV Credit Card Return

(Note: All "Required Fields" are identified by bold red font)

(Note: EMV related fields are in bold blue font)

INPUT XML Structure:

```
<transaction-request>
  <verification>
```

```
    <merchantId/>
    <merchantKey/>
  </verification>
  <order>
    <emvReturn>
      <orderID/>
      <referenceNum/>
      <payment>
        <chargeTotal/>
       <salesTaxTotal/>
 <shippingTotal/>
 <convenienceFee/>
 <currencyCode/>
      </payment>
    </emvReturn>
  </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
    <authCode/>
    <orderID/>
    <referenceNum/>
    <transactionID/>
    <transactionTimestamp/>
    <responseCode/>
    <responseMessage/>
    <avsResponseCode/>
    <cvvResponseCode/>
    <processorCode/>
    <processorMessage/>
    <errorMessage/>
    <partiallyApprovedAmount/>
    <creditCardScheme/>
    <emvDataResponse/>  <!--Base64 Encoded -->
    <fileUpdateName/>
    <fileUpdateCreateDate/>
    <fileUpdateSize/>
    <fileUpdateCRC/>
    <fileUpdateFCode/>
</transaction-response>
```

Or

```
<api-error>
    <errorCode></errorCode>
    <errorMsg></errorMsg>
</api-error>
```

EMV Credit Card Credit

(Note: All "Required Fields" are identified by bold red font)

(Note: EMV related fields are in bold blue font)

INPUT XML Structure:

```
<transaction-request>
  <verification>
    <merchantId/>
    <merchantKey/>
  </verification>
  <order>
    <emvCredit>
      <orderID/>
      <invoiceNumber/>
      <referenceNum/>
      <consumerPrimaryId/>
      <consumerExternalId/>
      <ipAddress/>
      <billing/>
      <transactionDetail><!--number,expMonth,and expYear OR track2 are Reqd fields-->
        <payType>
          <creditCard>
            <track1Data/>
            <track2Data/>
            <eCommInd/>
            <cardSequenceNumber>
            <emvDataRequest>
            <isFallBackToSwipe>
            <systemTraceNumber>
            <processorMid>
            <processorTid>
          </creditCard>
        </payType>
      </transactionDetail>
      <payment>
        <chargeTotal/>
       <salesTaxTotal/>
 <shippingTotal/>
 <convenienceFee/>
 <currencyCode/>
      </payment>
      <processorID/>
    </emvCredit>
  </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
   <authCode/>
   <orderID/>
```

```
   <referenceNum/>
   <transactionID/>
   <transactionTimestamp/>
   <responseCode/>
   <responseMessage/>
   <avsResponseCode/>
   <cvvResponseCode/>
   <processorCode/>
   <processorMessage/>
   <errorMessage/>
   <partiallyApprovedAmount/>
   <creditCardScheme/>
   <emvDataResponse/>  <!--Base64 Encoded -->
   <fileUpdateName/>
   <fileUpdateCreateDate/>
   <fileUpdateSize/>
   <fileUpdateCRC/>
   <fileUpdateFCode/>
</transaction-response>
```

Or

```
<api-error>
   <errorCode></errorCode>
   <errorMsg></errorMsg>
</api-error>
```

EMV Void Transaction

(Note: All "Required Fields" are identified by bold red font)

(Note: EMV related fields are in bold blue font)

INPUT XML Structure:

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

OUTPUT XML Structure:

```
<transaction-response>
   <authCode/>
```

```
   <orderID/>
   <referenceNum/>
   <transactionID/>
   <transactionTimestamp/>
   <responseCode/>
   <responseMessage/>
   <avsResponseCode/>
   <cvvResponseCode/>
   <processorCode/>
   <processorMessage/>
   <errorMessage/>
   <partiallyApprovedAmount/>
   <creditCardScheme/>
   <emvDataResponse/>  <!--Base64 Encoded -->
   <fileUpdateName/>
   <fileUpdateCreateDate/>
   <fileUpdateSize/>
   <fileUpdateCRC/>
   <fileUpdateFCode/>
</transaction-response>
```

Or

```
<api-error>
   <errorCode></errorCode>
   <errorMsg></errorMsg>
</api-error>
```

EMV Reverse Transaction

(Note: All "Required Fields" are identified by bold red font)

(Note: EMV related fields are in bold blue font)

INPUT XML Structure:

```
<transaction-request>
  <verification>
    <merchantId/>
    <merchantKey/>
    <verification>
      <order>
        <emvReverse>
          <transactionID/>
          <ipAddress/>
          <orderID/>
          <referenceNum/>
          <payment>
            <chargeTotal/>
          </payment>
        </emvReverse>
      </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
   <authCode/>
   <orderID/>
   <referenceNum/>
   <transactionID/>
   <transactionTimestamp/>
   <responseCode/>
   <responseMessage/>
   <avsResponseCode/>
   <cvvResponseCode/>
   <processorCode/>
   <processorMessage/>
   <errorMessage/>
   <partiallyApprovedAmount/>
   <creditCardScheme/>
   <emvDataResponse/>  <!--Base64 Encoded -->
   <fileUpdateName/>
   <fileUpdateCreateDate/>
   <fileUpdateSize/>
   <fileUpdateCRC/>
   <fileUpdateFCode/>
</transaction-response>
```

Or

```
<api-error>
   <errorCode></errorCode>
   <errorMsg></errorMsg>
</api-error>
```

EMV Debit Transaction

(Note: All "Required Fields" are identified by bold red font)

(Note: EMV related fields are in bold blue font)

INPUT XML Structure:

```
<transaction-request>
  <verification>
    <merchantId></merchantId>
    <merchantKey></merchantKey>
  </verification>
  <order>
    <emvDebitSale>
      <referenceNum></referenceNum>
      <ipAddress></ipAddress>
      <invoiceNumber/>
      <billing>
        <name></name>
```

```
        <address></address>
        <address2/>
        <city></city>
        <state></state>
        <postalcode></postalcode>
        <country></country>
        <phone></phone>
        <email></email>
      </billing>
      <shipping>
        <name> </name>
        <address> </address>
        <address2/>
        <city> </city>
        <state> </state>
        <postalcode></postalcode>
        <country> </country>
        <phone></phone>
        <email> </email>
      </shipping>
      <transactionDetail>
        <payType>
          <debitCard>
            <deviceType/>
            <deviceKSN />
            <track1Data encrypted="Y"/> <!-- If the merchant is using an
encrypted device for swiping the card, set encrypted = “Y” for
all track data that is sent. -->
            <track2Data encrypted="Y"/>
            <pinBlock> </pinBlock>
            <keySerialNo> </keySerialNo>
            <eCommInd>retail</eCommInd>
            <cardSequenceNumber>
            <emvDataRequest>
            <isFallBackToSwipe>
            <systemTraceNumber>
            <processorMid>
            <processorTid>
            <MessageAuthCode>
            <MessageAuthCheckDigits>
            <debitAccountType>
          </debitCard>
        </payType>
      </transactionDetail>
      <payment>
        <chargeTotal></chargeTotal>
        <salesTaxTotal></salesTaxTotal>
        <shippingTotal></shippingTotal>
        <cashBack></cashBack>
 <currencyCode/>
      </payment>
    </emvDebitSale>
  </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
   <authCode/>
   <orderID/>
   <referenceNum/>
   <transactionID/>
   <transactionTimestamp/>
   <responseCode/>
   <responseMessage/>
   <avsResponseCode/>
   <cvvResponseCode/>
   <processorCode/>
   <processorMessage/>
   <errorMessage/>
   <partiallyApprovedAmount/>
   <creditCardScheme/>
   <emvDataResponse/>  <!--Base64 Encoded -->
   <fileUpdateName/>
   <fileUpdateCreateDate/>
   <fileUpdateSize/>
   <fileUpdateCRC/>
   <fileUpdateFCode/>
   <gatewayMessageAuthCode/>
   <gatewayMessageAuthCheckDigits/>
   <TMACKey/>
   <TKPEKey/>
   <TKMEKey/>
</transaction-response>
```

Or

```
<api-error>
   <errorCode></errorCode>
   <errorMsg></errorMsg>
</api-error>
```

Sample Input Code for a EMV PIN Debit 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>
 <emvDebitSale>
  <referenceNum>1</referenceNum>
  <ipAddress>123.123.123.123</ipAddress>
  <billing>
   <name>Tom Customer</name>
   <address>123 1st Street</address>
   <address2/>
   <city>Moorpark</city>
   <state>CA</state>
   <postalcode>91307</postalcode>
   <country>US</country>
   <phone>8181231234</phone>
   <email>customer@example.com</email>
  </billing>
  <shipping>
   <name>Tom Customer</name>
   <address>123 1st Street</address>
   <address2/>
   <city>Moorpark</city>
   <state>CA</state>
   <postalcode>91307</postalcode>
   <country>US</country>
   <phone>8181231234</phone>
   <email>customer@example.com</email>
  </shipping>
  <transactionDetail>
   <payType>
    <debitCard>
     <track1Data>1234567124351258312</track1Data>
     <track2Data>12376128931682910239821</track2Data>
     <pinBlock>273A61283641823</pinBlock>
     <keySerialNo>10002320930000</keySerialNo>
     <eCommInd>retail</eCommInd>
     <cardSequenceNumber>001</cardSequenceNumber>
     <emvDataRequest>Base64EncodedEMVRequestData</emvDataRequest>
     <isFallBackToSwipe>0</isFallBackToSwipe>
     <systemTraceNumber>123456</systemTraceNumber>
     <processorMid>1234567</processorMid>
     <processorTid>1234567</processorTid>
  <messageAuthCode>1111aaaa2222bbbb</messageAuthCode>
     <messageAuthCheckDigits>11aa</messageAuthCheckDigits>
     <debitAccountType>saving</debitAccountType>
    </debitCard>
   </payType>
  </transactionDetail>
  <payment>
   <chargeTotal>3.00</chargeTotal>
   <salesTaxTotal>0.38</salesTaxTotal>
   <shippingTotal>3.03</shippingTotal>
   <cashBack>2.05</cashBack>
  </payment>
 </emvDebitSale>
</order>
```

```
</transaction-request>
```

Sample Accepted Response from a EMV PIN Debit Transaction

If the response code is 0, the transaction was successful.

```
<transaction-response>
    <authCode>123456</authCode>
    <orderID>0AF90446:0125E8E41FC7:49A5:001145CC</orderID>
    <referenceNum>1</referenceNum>
    <transactionID>745032</transactionID>
    <transactionTimestamp>1262332682</transactionTimestamp>
    <responseCode>0</responseCode>
    <responseMessage>ACCEPTED</responseMessage>
    <avsResponseCode>YYY</avsResponseCode>
    <cvvResponseCode>M</cvvResponseCode>
    <processorCode>A</processorCode>
    <processorMessage>APPROVED</processorMessage>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <errorMessage />
    <emvDataResponse>Base64EncodedEMVResponseData<emvDataResponse/>
    <fileUpdateName/>
    <fileUpdateCreateDate/>
    <fileUpdateSize/>
    <fileUpdateCRC/>
    <fileUpdateFCode/>
    <gatewayMessageAuthCode/>
    <gatewayMessageAuthCheckDigits/>
    <TMACKey/>
    <TKPEKey/>
    <TKMEKey/>
</transaction-response>
```

EMV Debit Return

(Note: All "Required Fields" are identified by bold red font)

(Note: EMV related fields are in bold blue font)

INPUT XML Structure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<transaction-request>
  <verification>
    <merchantId></merchantId>
    <merchantKey> </merchantKey>
  </verification>
  <order>
    <emvDebitReturn>
      <orderID></orderID>
      <referenceNum></referenceNum>
      <ipAddress></ipAddress>
      <invoiceNumber/>
```

```
    <billing>
      <name> </name>
      <address> </address>
      <address2/>
      <city></city>
      <state> </state>
      <postalcode></postalcode>
      <country></country>
      <phone></phone>
      <email> </email>
    </billing>
    <payment>
      <chargeTotal></chargeTotal>
      <salesTaxTotal></salesTaxTotal>
      <shippingTotal></shippingTotal>
      <cashBack></cashBack>
    </payment>
    <transactionDetail>
      <payType>
        <debitCard>
          <deviceType/>
          <deviceKSN />
          <track1Data encrypted="Y"/> <!-- If the merchant is using an
encrypted device for swiping the card, set encrypted = "Y" for
all track data that is sent. -->
          <track2Data encrypted="Y"/>
          <pinBlock></pinBlock>
          <keySerialNo></keySerialNo>
          <eCommInd>retail</eCommInd>
          <cardSequenceNumber>
          <emvDataRequest>
          <isFallBackToSwipe>
          <systemTraceNumber>
          <processorMid>
          <processorTid>
            <MessageAuthCode>
          <MessageAuthCheckDigits>
         <debitAccountType>
        </debitCard>
      </payType>
    </transactionDetail>
  </emvDebitReturn>
 </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
   <authCode/>
   <orderID/>
   <referenceNum/>
```

```
    <transactionID/>
    <transactionTimestamp/>
    <responseCode/>
    <responseMessage/>
    <avsResponseCode/>
    <cvvResponseCode/>
    <processorCode/>
    <processorMessage/>
    <errorMessage/>
    <partiallyApprovedAmount/>
    <emvDataResponse/>  <!--Base64 Encoded -->
    <fileUpdateName/>
    <fileUpdateCreateDate/>
    <fileUpdateSize/>
    <fileUpdateCRC/>
    <fileUpdateFCode/>
    <gatewayMessageAuthCode/>
    <gatewayMessageAuthCheckDigits/>
    <TMACKey/>
    <TKPEKey/>
    <TKMEKey/>
</transaction-response>
```

Or

```
<api-error>
    <errorCode></errorCode>
    <errorMsg></errorMsg>
</api-error>
```

EMV Debit Void

(Note: All "Required Fields" are identified by bold red font)

(Note: EMV related fields are in bold blue font)

INPUT XML Structure:

```
<transaction-request>
  <verification>
    <merchantId></merchantId>
    <merchantKey></merchantKey>
  </verification>
  <order>
    <emvDebitVoid>
      <transactionID></transactionID>
      <ipAddress></ipAddress>
      <transactionDetail>
        <payType>
          <debitCard>
            <deviceType/>
            <deviceKSN />
            <track1Data encrypted="Y"/> <!-- If the merchant is using an
encrypted device for swiping the card, set encrypted = "Y" for all
track data that is sent. -->
            <track2Data encrypted="Y"/>
```

```
        <pinBlock></pinBlock>
        <keySerialNo></keySerialNo>
        <eCommInd>retail</eCommInd>
        <cardSequenceNumber>
        <emvDataRequest>
        <isFallBackToSwipe>
        <systemTraceNumber>
        <processorMid>
        <processorTid>
    <MessageAuthCode>
        <MessageAuthCheckDigits>
       <debitAccountType>
      </debitCard>
    </payType>
  </transactionDetail>
 </emvDebitVoid>
 </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
   <authCode/>
   <orderID/>
   <referenceNum/>
   <transactionID/>
   <transactionTimestamp/>
   <responseCode/>
   <responseMessage/>
   <avsResponseCode/>
   <cvvResponseCode/>
   <processorCode/>
   <processorMessage/>
   <errorMessage/>
   <partiallyApprovedAmount/>
   <emvDataResponse/>  <!--Base64 Encoded -->
   <fileUpdateName/>
   <fileUpdateCreateDate/>
   <fileUpdateSize/>
   <fileUpdateCRC/>
   <fileUpdateFCode/>
   <gatewayMessageAuthCode/>
   <gatewayMessageAuthCheckDigits/>
   <TMACKey/>
   <TKPEKey/>
   <TKMEKey/>
</transaction-response>
```

Or

```
<api-error>
   <errorCode></errorCode>
   <errorMsg></errorMsg>
```

```
</api-error>
```

EMV File Download

This is newly added functionality for File download request from terminal and currently supported only on the First Data platform.

(Note: All "Required Fields" are identified by bold red font)

Sample Input Code for a EMV File Download:

```
<transaction-request>
  <verification>
    <merchantId></merchantId>
    <merchantKey></merchantKey>
  </verification>
  <order>
        <emvFileDownload>
          <referenceNum>1</referenceNum>
          <fileDownloadName>EMV2KEY</fileDownloadName>
          <fileBlockSequence>0001</fileBlockSequence>
          <fileBlockMaxSize>500</fileBlockMaxSize>
          <fileOffset>0</fileOffset>
      </emvFileDownload>
  </order>
</transaction-request>
```

Sample Response for a EMV File Download:

```
<transaction-response>
   <authCode/>
   <orderID/>
   <referenceNum>1</referenceNum>
   <transactionID>367734</transactionID>
   <transactionTimestamp>1403200631</transactionTimestamp>
   <responseCode>0</responseCode>
   <responseMessage>SUCCESS</responseMessage>
   <avsResponseCode/>
   <cvvResponseCode/>
   <processorCode>A</processorCode>
   <processorMessage>APPROVED</processorMessage>
   <errorMessage/>
   <fileUpdateName/>
   <fileUpdateCreateDate/>
   <fileUpdateSize/>
   <fileUpdateCRC/>
   <fileUpdateFCode/>
   <fileDownloadName>EMV2KEY</fileDownloadName>
   <fileBlockSequence>0001</fileBlockSequence>
   <fileBlockSize>500</fileBlockSize>
   <fileNextOffset>0000500</fileNextOffset>
   <fileDownloadStatusCode>C</fileDownloadStatusCode>
   <fileBlockData>CA_KEYS,0005
```

12312016,01,01,A000000025,0E,AA94A8C6DAD24F9BA56A27C09B01020819568B81A026BE9FD0A3416
CA9A71166ED5084ED91CED47DD457DB7E6CBCD53E560BC5DF48ABC380993B6D549F5196CFA77DF
B20A0296188E969A2772E8C4141665F8BB2516BA2C7B5FC91F8DA04E8D512EB0F6411516FB86FC021
CE7E969DA94D33937909A53A57F907C40C22009DA7532CB3BE509AE173B39AD6A01BA5BB85,03,A7
266ABAE64B42A3668851191D49856E17F8FBCD
12312017,01,01,A000000025,0F,C8D5AC27A5E1FB89978C7C6479AF993AB3800EB243996FBB2AE26B6
7B23AC482C4B746005A51AFA7D2D83E894F591A
</fileBlockData>
</transaction-response>

Schools Implementation

Paymentsite offers a payment solution specifically designed to support private schools and universities that have the need to tie student and/or parent information to their transactions. An API implementation that implements the school data structure can be used in conjunction with Paymentsite for Schools and/or the Link2Student portal.

Note that all the normal transaction types may be used for a schools implementation—what relates it to a school is the inclusion of one or more of the school-related data fields, shown below in **blue bold font**.

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

INPUT XML Structure:

```
<transaction-request>
    <verification>
        <merchantId/> <!-- Required field -->
        <merchantKey/> <!-- Required field -->
    </verification>
    <order>
        <sale>
            <referenceNum/> <!-- Required field -->
            <studentAccountNumber/>
            <ipAddress/>
            <invoiceNumber/>
            <billing>
                <name/>
                <address/>
                <address2/>
                <city/>
                <state/>
                <postalcode/>
                <country/>
                <phone/>
                <email/>
            </billing>
            <shipping>
                <name/>
```

```
                <address/>
                <address2/>
                <city/>
                <state/>
                <postalcode/>
                <country/>
                <phone/>
                <email/>
            </shipping>
            <transactionDetail>
            <!-- number, expMonth, and expYear OR track2 are Required fields -->
                <payType>
                    <creditCard>
                        <number/>
                        <expMonth/>
                        <expYear/>
                        <cvvInd/>
                        <cvvNumber/>
                        <track1Data/>
                        <track2Data/>
                        <eCommInd/>
                    </creditCard>
                </payType>
            </transactionDetail>
            <payment>
                <chargeTotal/>
                <!-- Required field -->
            </payment>
        </sale>
        <clientData>
            <comments/>
            <program/> <!—Must be a valid program in Paymentsite for Schools →
            <category/>
            <subcategory/>
            <parentName/>
            <schoolId/>
            <studentName/>
            <parentId/>
            <amount/>
            <convienceFee/>
        </clientData>
    </order>
</transaction-request>
```

Sample Input Code for a Credit Card Sale Transaction for a School

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>123</merchantId> <!-- Required field -->
        <merchantKey>key</merchantKey> <!-- Required field -->
    </verification>
    <order>
        <sale>
            <referenceNum>1</referenceNum> <!-- Required field -->
            <studentAccountNumber>BAT01</studentAccountNumber>
            <ipAddress>192.123.123.123</ipAddress>
            <billing>
                <name>Tom Customer</name>
                <address>123 Street</address>
                <address2/>
                <city>Moorpark</city>
                <state>CA</state>
                <postalcode>91307</postalcode>
                <country>US</country>
                <phone>8181231234</phone>
                <email>customer@example.com</email>
            </billing>
            <shipping>
                <name>Tom Customer</name>
                <address>123 Street</address>
                <address2/>
                <city>Moorpark</city>
                <state>CA</state>
                <postalcode>91307</postalcode>
                <country>US</country>
                <phone>8181231234</phone>
                <email>customer@example.com</email>
            </shipping>
            <transactionDetail>
                <payType>
                    <creditCard>
                        <number>5473500000000014</number>
                        <expMonth>12</expMonth>
                        <expYear>2017</expYear>
                        <cvvInd/>
                        <cvvNumber>123</cvvNumber>
                        <track1Data/>
                        <track2Data/>
                        <eCommInd>retail</eCommInd>
                    </creditCard>
                </payType>
            </transactionDetail>
            <payment>
                <chargeTotal>23.00</chargeTotal> <!-- Required field -->
            </payment>
        </sale>
        <clientData>
            <program>Boosters</program>
            <category>Sports Spirit Packs</category>
            <subCategory>Soccer</subCategory>
            <parentName>Susan Johnson</parentName>
            <studentName>Tina Johnson</studentName>
```

```
            <studentId>473</studentId>
            <parentId>sjohnson2291</parentId>
        </clientData>
    </order>
</transaction-request>
```

OUTPUT XML Structure:

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

Property Management Implementation

Paymentsite offers a payment solution specifically designed to support property managers that have the need to tie tenant information to their transactions. An API implementation that implements the property management data structure can be used in conjunction with Property Management Paymentsite and/or Link2Payments.

Note that all the normal transaction types may be used for a property management implementation—what relates it to property management is the inclusion of one or more of the property-management-related data fields, shown below in **blue bold font**.

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

INPUT XML Structure:

```
<?xml version="1.0" encoding="UTF-8" ?>
    <transaction-request>
    <verification>
        <merchantId/>
        <merchantKey/>
    </verification>
    <order>
        <sale>
            <referenceNum/>
            <ipAddress/>
```

```
        <tenantAccountNumber/>
        <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>/>
                    <eCommInd/>
                </creditCard>
            </payType>
        </transactionDetail>
        <payment>
            <chargeTotal/>
            <salesTaxTotal/>
            <convenienceFee/>
        </payment>
    </sale>
    <clientData>
        <comments/>
        <property/>
        <department/>
        <customerPaying/>
        <tenantApartmentNumber/>
        <tenantName/>
        <tenantStatus/>
        <leaseRent/>
        <balanceForward/>
        <parkingGarage/>
        <storage/>
        <associationFee/>
```

```
            <delinquincyLateFee/>
            <utilityServices/>
            <deposit/>
            <applicationProcessing/>
            <petFee/>
            <nsfFee/>
            <specialFeeOther/>
            <taxes/>
            <customField1/>
            <customField2/>
            <customField3/>
            <customField4/>
            <customField5/>
        </clientData>
    </order>
</transaction-request>
```

Sample Input Code for a Credit Card Sale Transaction for a Property Manager

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.

```
<?xml version="1.0" encoding="UTF-8" ?>
<transaction-request>
    <version>1.0.1111.1</version>
    <verification>
        <merchantId>1</merchantId>
        <merchantKey>key</merchantKey>
    </verification>
    <order>
        <sale>
            <referenceNum>1</referenceNum>
            <ipAddress>123.123.123.123</ipAddress>
            <tenantAccountNumber>12345-12432</tenantAccountNumber>
            <billing>
                <name>Tom Customer</name>
                <address>123 My Street</address>
                <address2/>
                <city>MyCity</city>
                <state>CA</state>
                <postalcode>91307</postalcode>
                <country>US</country>
                <phone>8181231234</phone>
                <email>customer@example.com</email>
            </billing>
            <transactionDetail>
                <payType>
                    <creditCard>
                        <number>4111111111111111</number>
                        <expMonth>12</expMonth>
```

```
                        <expYear>2017</expYear>
                        <cvvInd/>
                        <cvvNumber>123</cvvNumber>
                        <eCommInd>retail</eCommInd>
                    </creditCard>
                </payType>
            </transactionDetail>
            <payment>
                <chargeTotal>3.00</chargeTotal>
                <salesTaxTotal>0.38</salesTaxTotal>
                <convenienceFee>1.00</convenienceFee>
            </payment>
            <processorID>3</processorID>
        </sale>
        <clientData>
            <comments>Test comment</comments>
            <property>Westridge Hills</property>
            <department>Billing</department>
            <customerPaying>In Person</customerPaying>
            <tenantApartmentNumber>12</tenantApartmentNumber>
            <tenantName>Troy Customer</tenantName>
            <tenantStatus>current</tenantStatus>
            <leaseRent>$850</leaseRent>
            <balanceForward>$0</balanceForward>
            <parkingGarage>$15</parkingGarage>
            <storage>$20</storage>
            <associationFee>$0</associationFee>
            <delinquincyLateFee>$0</delinquincyLateFee>
            <utilityServices>$105.34</utilityServices>
            <deposit>$0</deposit>
            <applicationProcessing>$0</applicationProcessing>
            <petFee>$0</petFee>
            <nsfFee>$0</nsfFee>
            <specialFeeOther>$0</specialFeeOther>
            <taxes>$0</taxes>
            <customField1> </customField1>
            <customField2> </customField2>
            <customField3> </customField3>
            <customField4> </customField4>
            <customField5> </customField5>
        </clientData>
    </order>
</transaction-request>
```

OUTPUT XML Structure:

```
<transaction-response>
    <authCode/>
    <orderID/>
    <referenceNum/>
    <transactionID/>
    <transactionTimestamp/>
    <responseCode/>
```

```
    <responseMessage/>
    <avsResponseCode/>
    <cvvResponseCode/>
    <processorCode/>
    <processorMessage/>
    <errorMessage/>
</transaction-response>
```

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>key</merchantKey>
    </verification>
    <command>submit-transaction-inquiry</command>
    <request>
        <referenceNum>12345678</referenceNum>
    </request>
</api-request>
```

Sample Response from submit-transaction-inquiry request:

If the response code is 0, the transaction inquiry was successful.

```
<api-response>
    <errorCode>0</errorCode>
    <errorMessage></errorMessage>
    <command>submit-transaction-inquiry</command>
    <time>120123123123123</time>
    <result>
        <requestId>87654321</requestId>
    </result>
</api-response>
```

If the reference number sent was not unique for each transaction, the response will be as follows:

```
<api-response>
    <errorCode>0</errorCode>
    <errorMessage />
    <command>get-transaction-inquiry-status</command>
    <time>1324410315169</time>
    <result>
        <requestStatus>COMPLETED</requestStatus>
        <transactionStatus>MULTIPLE TRANSACTIONS FOUND</transactionStatus>
    </result>
</api-response>
```

It is very important that the merchant assign a unique reference number to every transaction.

The second step is to go back sometime later (perhaps 5 - 10 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>
        <merchantId>123</merchantId>
        <merchantKey>key</merchantKey>
    </verification>
    <command>get-transaction-inquiry-status</command>
    <request>
        <requestId>87654321</requestId>
    </request>
</api-request>
```

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

If the response code is **0**, the transaction inquiry was successful.

```
<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

requestStatus	transactionStatus	Meaning	Recommended action
NEW	--	Your status request hasn't been processed yet.	Resend the *get transaction-inquirystatus* 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	NOT FOUND	Your request is complete; we found your transaction and it was approved.	No action required.
COMPLETED	NOT FOUND	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.

JSON Samples for Transaction Processing

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

Request:

```
{
   "transaction-request": {
      "version": "",
      "verification": {. . .},
      "order": {. . .}
```

```
}
}
```

Response:

```
{
   "transaction-response": {
      ...
   }
}
```

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

```
{
   "api-error": {
      “errorCode”: ,
      “errorMsg”:””
   }
}
```

The posting URL to use for **testing** API **transaction requests** is as follows. This URL is strictly for posting transactions such as auth, capture, sale, return, etc.

Posting URL for test transaction requests:

https://apiint.paymentsite.com/UniversalAPI/postXML

The posting URL to use for **LIVE** API **transaction requests** is as follows. This URL is strictly for posting transactions such as auth, capture, sale, return, etc

Posting URL for LIVE transaction requests:

https://api.paymentsite.com/UniversalAPI/postXML

Important! Please note: If a “partiallyApprovedAmount” tag is included in the response, this means the sale or authorization was only approved for part of the amount requested. The transaction will return an APPROVED response, and the tag “partiallyApprovedAmount” will include the amount that the transaction was approved for.

*** 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:
   <creditCard><number/><expMonth/><expYear/></creditCard>

2) If the transaction is a card-swipe transaction, only the track data is required—we recommend that for a swiped transaction that you provide ONLY the track data. There is no need to pass the credit card number if you send the track data. Some processors only require track 2, but others require both tracks, so we recommend you send both tracks:

   <track1Data/> and/or <track2Data/>

   Or send <trackData/> containing both tracks in one field

*** Any attruibutes have to be prefixed with '@' for example "itemList@itemCount=\"2\"" where ' itemCount="2" ' is an attribute for 'itemList' tag. It will be converted in xml as <itemList itemCount="2">

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

Auth

Request:

```
{
   "transaction-request": {
      "version": "3.1.1.1",
      "verification": {
         "merchantId": 11111,
         "merchantKey": "key"
      },
      "order": {
         "auth": {
            "shipping": {
               "name": "Susan Johnson",
               "address": "123 Main Street",
               "address2": "Apt 36",
               "city": "Moorpark",
               "state": "CA",
               "country": "US",
               "postalcode": 91307,
               "phone": "818-123-1234",
               "email": "customer@example.com"
            },
            "billing": {
               "name": "Susan Johnson",
               "address": "123 Main Street",
               "address2": "Apt 36",
               "city": "Moorpark",
               "state": "CA",
               "country": "US",
               "postalcode": 91307,
               "phone": "818-123-1234",
               "email": "customer@example.com"
            },
            "payment": {
               "chargeTotal": 32.0,
               "shippingTotal": 3.0,
```

```
            "salesTaxTotal": 1.23,
            "currencyCode": "USD"
         },
         "transactionDetail": {
            "payType": {
               "creditCard": {
                  "number": 4111111111111111,
        "expMonth": 12,
                  "expYear": 2028,
                  "cvvNumber": 123,
                  "cvvInd": "",
                  "track1Data": "",
                  "track2Data": "",
                  "signatureImage": "",
                  "eCommInd": "retail"
               }
            }
         },
         "referenceNum": 846392232,
         "invoiceNumber": 123456,
         "ipAddress": "123.123.123.123"
      }
   }
}
```


Response:


```
{
   "transaction-response": {
      "errorMessage": "",
      "authCode": 123456,
      "responseCode": 0,
      "orderID": "C0A8631F:014EF4400822:FF86:29C0CFF9",
      "cvvResponseCode": "M",
      "number": 4111111111111111,
      "transactionID": 464363,
      "processorMessage": "APPROVED",
      "eCommInd": "retail",
      "avsResponseCode": "YYY",
      "referenceNum": 846392232,
      "responseMessage": "AUTHORIZED",
      "processorCode": "A",
      "merchantId": 11111,
      "transactionTimestamp": 1438616914
   }
}
```


Capture

Request:


```
{
   "transaction-request": {
```

```
      "version": "3.1.1.1",
      "verification": {
         "merchantId": 11111,
         "merchantKey": "key"
      },
      "order": {
         "capture": {
            "orderID": "C0A8631F:014EF46C082D:728F:30F47537",
            "referenceNum": 846392233,
            "ipAddress": "123.123.123.123",
            "payment": {
               "chargeTotal": 32.00,
            }
         }
      }
   }
}
```

Response:

```
{
   "transaction-response": {
      "errorMessage": "",
      "authCode": "",
      "responseCode": 0,
      "orderID": "C0A8631F:014EF46C082D:728F:30F47537",
      "cvvResponseCode": "",
      "transactionID": 464377,
      "processorMessage": "APPROVED",
      "avsResponseCode": "",
      "referenceNum": 846392233,
      "responseMessage": "CAPTURED",
      "processorCode": "A",
      "merchantId": 11111,
      "transactionTimestamp": 1438619839
   }
}
```

Sale

Request:

```
{
   "transaction-request": {
      "version": "3.1.1.1",
      "verification": {
         "merchantId": 11111,
         "merchantKey": "key"
      },
```

```
    "order": {
      "sale": {
        "shipping": {
          "name": "Susan Johnson",
          "address": "123 Main Street",
          "address2": "Apt 36",
          "city": "Moorpark",
          "state": "CA",
          "country": "US",
          "postalcode": 91307,
          "phone": "818-123-1234",
          "email": "customer@example.com"
        },
        "billing": {
          "name": "Susan Johnson",
          "address": "123 Main Street",
          "address2": "Apt 36",
          "city": "Moorpark",
          "state": "CA",
          "country": "US",
          "postalcode": 91307,
          "phone": "818-123-1234",
          "email": "customer@example.com"
        },
        "payment": {
          "chargeTotal": 32.0,
          "shippingTotal": 3.0,
          "salesTaxTotal": 1.23,
          "currencyCode": "USD"
        },
        "transactionDetail": {
          "payType": {
            "creditCard": {
              "number": 4111111111111111,
      "expMonth": 12,
              "expYear": 2028,
              "cvvNumber": 123,
              "cvvInd": "",
              "track1Data": "",
              "track2Data": "",
              "signatureImage": "",
              "eCommInd": "retail"
            }
          }
        },
        "referenceNum": 846392232,
        "invoiceNumber": 123456,
        "ipAddress": "123.123.123.123"
      }
    }
  }
}
```


Response:

```
{
   "transaction-response": {
      "errorMessage": "",
      "authCode": 123456,
      "responseCode": 0,
      "orderID": "C0A8631F:014EF47935EE:4959:2AB4B8ED",
      "cvvResponseCode": "M",
      "number": 4111111111111111,
      "transactionID": 464380,
      "processorMessage": "APPROVED",
      "eCommInd": "retail",
      "avsResponseCode": "YYY",
      "referenceNum": 846392232,
      "responseMessage": "CAPTURED",
      "processorCode": "A",
      "merchantId": 11111,
      "transactionTimestamp": 1438620661
   }
}
```


Force Sale

Request:


```
{
   "transaction-request": {
      "version": "3.1.1.1",
      "verification": {
         "merchantId": 11111,
         "merchantKey": "key"
      },
      "order": {
         "saleForce": {
            "shipping": {
               "name": "Susan Johnson",
               "address": "123 Main Street",
               "address2": "Apt 36",
               "city": "Moorpark",
               "state": "CA",
               "country": "US",
               "postalcode": 91307,
               "phone": "818-123-1234",
               "email": "customer@example.com"
            },
            "billing": {
               "name": "Susan Johnson",
               "address": "123 Main Street",
               "address2": "Apt 36",
               "city": "Moorpark",
               "state": "CA",
               "country": "US",
               "postalcode": 91307,
               "phone": "818-123-1234",
               "email": "customer@example.com"
            },
```

```
            "payment": {
               "chargeTotal": 32.0,
               "shippingTotal": 3.0,
               "salesTaxTotal": 1.23,
               "currencyCode": "USD"
            },
            "transactionDetail": {
               "payType": {
                  "creditCard": {
                     "track1Data": "%B6011000995500000^ Test Card ^15121015432112345678?",
                     "track2Data": ";6011000995500000=15121015432112345678?",
                    "signatureImage": "",
                     "eCommInd": "retail"
                  }
               }
            },
            "invoiceNumber": 123456,
            "ipAddress": "123.123.123.123",
            "referenceNum": 846392232,
            "authCode" : 123456
         }
      }
   }
}
```


Response:


```
{
   "transaction-response": {
      "errorMessage": "",
      "authCode": "",
      "responseCode": 0,
      "orderID": "C0A8631F:014EF4B71E26:8D27:3BB02548",
      "cvvResponseCode": "",
      "transactionID": 464404,
      "processorMessage": "APPROVED",
      "eCommInd": "retail",
      "avsResponseCode": "",
      "referenceNum": 846392232,
      "responseMessage": "CAPTURED",
      "processorCode": "A",
      "merchantId": 11111,
      "transactionTimestamp": 1438624718
   }
}
```


Level2 Sale

Request:


```
{
   "transaction-request": {
      "version": "3.1.1.1",
      "verification": {
```

```
    "merchantId": 11111,
    "merchantKey": "key"
  },
  "order": {
    "sale": {
      "shipping": {
        "name": "Susan Johnson",
        "address": "123 Main Street",
        "address2": "Apt 36",
        "city": "Moorpark",
        "state": "CA",
        "country": "US",
        "postalcode": 91307,
        "phone": "818-123-1234",
        "email": "customer@example.com"
      },
      "billing": {
        "name": "Susan Johnson",
        "address": "123 Main Street",
        "address2": "Apt 36",
        "city": "Moorpark",
        "state": "CA",
        "country": "US",
        "postalcode": 91307,
        "phone": "818-123-1234",
        "email": "customer@example.com"
      },
      "payment": {
        "chargeTotal": 32.0,
        "shippingTotal": 3.0,
        "salesTaxTotal": 1.23,
        "currencyCode": "USD"
      },
      "transactionDetail": {
        "payType": {
          "creditCard": {
            "number": 4111111111111111,
      "expMonth": 12,
            "expYear": 2028,
            "cvvNumber": 123,
            "cvvInd": "",
            "track1Data": "",
            "track2Data": "",
            "signatureImage": "",
            "eCommInd": "retail"
          }
        }
      },
      "purchaseCardDetail": {
        "PCLevel": 2,
        "PCOrderNumber": "order1234",
        "TaxIndicator": 1,
        "SDMerchantName": "mercahnt  xyz",
        "SDMerchantEmail": "",
```

```
                "SDMerchantURL": "",
                "SDMerchantCity": "",
                "SDMerchantPhone": "",
                "SDProductDescription": "",
                "AmexTransactionAdviceAddendum1": "",
                "AmexTransactionAdviceAddendum2": "",
                "AmexTransactionAdviceAddendum3": "",
                "AmexTransactionAdviceAddendum4": "",
            },
            "invoiceNumber": 123456,
            "referenceNum": 846392232,
            "ipAddress": "123.123.123.123"
        }
    }
  }
}
```


Response:


```
{
  "transaction-response": {
    "errorMessage": "",
    "authCode": 123456,
    "responseCode": 0,
    "orderID": "C0A8631F:014EF4BCF1CA:CA42:3E38D6BC",
    "cvvResponseCode": "M",
    "number": 4111111111111111,
    "transactionID": 464409,
    "processorMessage": "APPROVED",
    "eCommInd": "retail",
    "avsResponseCode": "YYY",
    "referenceNum": 846392232,
    "responseMessage": "CAPTURED",
    "processorCode": "A",
    "merchantId": 11111,
    "transactionTimestamp": 1438625100
  }
}
```


Level3 Sale

Request:


```
{
  "transaction-request": {
    "version": "3.1.1.1",
    "verification": {
      "merchantId": 11111,
      "merchantKey": "key"
    },
    "order": {
      "sale": {
        "shipping": {
          "name": "Susan Johnson",
          "address": "123 Main Street",
```

```
    "address2": "Apt 36",
    "city": "Moorpark",
    "state": "CA",
    "country": "US",
    "postalcode": 91307,
    "phone": "818-123-1234",
    "email": "customer@example.com"
},
"billing": {
    "name": "Susan Johnson",
    "address": "123 Main Street",
    "address2": "Apt 36",
    "city": "Moorpark",
    "state": "CA",
    "country": "US",
    "postalcode": 91307,
    "phone": "818-123-1234",
    "email": "customer@example.com"
},
"payment": {
    "chargeTotal": 32.0,
    "shippingTotal": 3.0,
    "salesTaxTotal": 1.23,
    "currencyCode": "USD"
},
"transactionDetail": {
    "payType": {
        "creditCard": {
            "number": 4111111111111111,
    "expMonth": 12,
            "expYear": 2028,
            "cvvNumber": 123,
            "cvvInd": "",
            "track1Data": "",
            "track2Data": "",
            "signatureImage": "",
            "eCommInd": "retail"
        }
    }
},
"purchaseCardDetail": {
    "PCLevel": 3,
    "PCOrderNumber": "order1234",
    "TaxIndicator": 1,
    "SDMerchantName": "mercahnt  xyz",
    "SDMerchantEmail": "",
    "SDMerchantURL": "",
    "SDMerchantCity": "",
    "SDMerchantPhone": "",
    "SDProductDescription": "",
    "AmexTransactionAdviceAddendum1": "",
    "AmexTransactionAdviceAddendum2": "",
    "AmexTransactionAdviceAddendum3": "",
    "AmexTransactionAdviceAddendum4": "",
```

```
    "PC3Detail": {
      "itemList@itemCount=\"2\"": {
        "item": [
          {
            "itemUnitCost": "1.1",
            "itemTotalAmount": "1.1",
            "itemProductCode": "abc",
            "itemIndex": "1",
            "itemTaxIncluded": "Y",
            "itemDescription": "item1",
            "itemTaxRate": "8",
            "itemQuantity": "1",
            "itemTaxAmount": "0.9",
            "itemDiscountAmount": "0.1",
            "itemTaxType": "G",
            "itemCommodityCode": "11",
            "itemUnitOfMeasurement": "lbs",
            "itemDebitIndicator": "C",
            "itemDiscountIndicator": "Y"
          },
          {
            "itemUnitCost": "2.2",
            "itemTotalAmount": "4.4",
            "itemProductCode": "xyz123",
            "itemIndex": "2",
            "itemTaxIncluded": "Y",
            "itemDescription": "electronic",
            "itemTaxRate": "8",
            "itemQuantity": "2",
            "itemTaxAmount": "0.32",
            "itemDiscountAmount": "0.20",
            "itemTaxType": "E",
            "itemCommodityCode": "12",
            "itemUnitOfMeasurement": "qty",
            "itemDebitIndicator": "D",
            "itemDiscountIndicator": "Y"
          }
        ]
      },
      "PC3FreightAmount": "1.2",
      "PC3ShipFromZip": "11223",
      "PC3DiscountAmount": "0.5",
      "PC3DutyAmount": "0.3",
      "PC3DestCountryCode": "USA",
      "PC3VatTaxRate": "5",
      "PC3AlternateTaxID": "1234567",
      "PC3VatTaxAmount": "0.4",
      "PC3AlternateTaxAmount": "0.1"
    }
  },
  "invoiceNumber": 123456,
  "referenceNum": 846392232,
  "ipAddress": "123.123.123.123"
}
```

```
      }
   }
}
```

Response:


```
{
   "transaction-response": {
      "errorMessage": "",
      "authCode": 123456,
      "responseCode": 0,
      "orderID": "C0A8631F:014EF4FDF90A:2F05:59D95947",
      "cvvResponseCode": "M",
      "number": 4111111111111111,
      "transactionID": 464433,
      "processorMessage": "APPROVED",
      "eCommInd": "retail",
      "avsResponseCode": "YYY",
      "referenceNum": 846392232,
      "responseMessage": "CAPTURED",
      "processorCode": "A",
      "merchantId": 11111,
      "transactionTimestamp": 1438629362
   }
}
```


Sale Card on file

Request:

```
{
   "transaction-request": {
      "version": "3.1.1.1",
      "verification": {
         "merchantId": 11111,
         "merchantKey": "key"
      },
      "order": {
         "sale": {
            "shipping": {
               "name": "Susan Johnson",
               "address": "123 Main Street",
               "address2": "Apt 36",
               "city": "Moorpark",
               "state": "CA",
               "country": "US",
               "postalcode": 91307,
               "phone": "818-123-1234",
               "email": "customer@example.com"
            },
            "billing": {
               "name": "Susan Johnson",
               "address": "123 Main Street",
               "address2": "Apt 36",
               "city": "Moorpark",
```

```
            "state": "CA",
            "country": "US",
            "postalcode": 91307,
            "phone": "818-123-1234",
            "email": "customer@example.com"
         },
         "payment": {
            "chargeTotal": 32.0,
            "shippingTotal": 3.0,
            "salesTaxTotal": 1.23,
            "currencyCode": "USD"
         },
         "transactionDetail": {
            "payType": {
               "onFile": {
                  "consumerPrimaryId": 7947,
                  "consumerExternalId": "",
                  "token": "HNZEHK2tCPY="
               }
            }
         },
         "referenceNum": 846392232,
         "ipAddress": "123.123.123.123",
         "invoiceNumber": 123456
      }
   }
}
```


Response:


```
{
   "transaction-response": {
      "errorMessage": "",
      "authCode": 123456,
      "responseCode": 0,
      "orderID": "C0A8631F:014EF519E1A2:02D0:30B13A34",
      "cvvResponseCode": "M",
      "transactionID": 464443,
      "processorMessage": "APPROVED",
      "avsResponseCode": "YYY",
      "referenceNum": 846392232,
      "responseMessage": "ACCEPTED",
      "processorCode": "A",
      "merchantId": 11111,
      "transactionTimestamp": 1438631191
   }
}
```


Return

Request:


```
{
```

```
   "transaction-request": {
     "version": "3.1.1.1",
      "verification": {
         "merchantId": 11111,
         "merchantKey": "key"
      },
      "order": {
         "return": {
            "orderID": "C0A8631F:014EF52096D9:717D:541F5C5B",
            "referenceNum": "123456",
            "payment": {
               "chargeTotal": 30.00,
            },
         }
      }
   }
}
```


Response:


```
{
   "transaction-response": {
      "errorMessage": "",
      "authCode": "",
      "responseCode": 0,
      "orderID": "C0A8631F:014EF52096D9:717D:541F5C5B",
      "cvvResponseCode": "",
      "transactionID": 464447,
      "processorMessage": "APPROVED",
      "avsResponseCode": "",
      "referenceNum": 123456,
      "responseMessage": "CAPTURED",
      "processorCode": "A",
      "merchantId": 11111,
      "transactionTimestamp": 1438631655
   }
}
```


Credit

Request:


```
{
   "transaction-request": {
     "version": "3.1.1.1",
      "verification": {
         "merchantId": 11111,
         "merchantKey": "key"
      },
      "order": {
         "credit": {
            "billing": {
               "name": "Susan Johnson",
               "address": "123 Main Street",
               "address2": "Apt 36",
               "city": "Moorpark",
```

```
            "state": "CA",
            "country": "US",
            "postalcode": 91307,
            "phone": "818-123-1234",
            "email": "customer@example.com"
         },
         "payment": {
            "chargeTotal": 32.0,
            "shippingTotal": 3.0,
            "salesTaxTotal": 1.23,
            "currencyCode": "USD"
         },
         "transactionDetail": {
            "payType": {
               "creditCard": {
                  "number": 4111111111111111,
            "expMonth": 12,
                  "expYear": 2028,
                  "cvvNumber": 123,
                  "cvvInd": "",
                  "track1Data": "",
                  "track2Data": "",
                  "signatureImage": "",
                  "eCommInd": "retail"
               }
            }
         },
         "invoiceNumber": 123456,
         "referenceNum": 846392232,
         "ipAddress": "123.123.123.123"
      }
   }
}
```


Response:


```
{
   "transaction-response": {
      "errorMessage": "",
      "authCode": "",
      "responseCode": 0,
      "orderID": "C0A8631F:014EF5709148:8370:7D3371C5",
      "cvvResponseCode": "",
      "number": 4111111111111111,
      "transactionID": 464465,
      "processorMessage": "APPROVED",
      "eCommInd": "retail",
      "avsResponseCode": "",
      "referenceNum": 846392232,
      "responseMessage": "CAPTURED",
      "processorCode": "A",
      "merchantId": 11111,
```

```
      "transactionTimestamp": 1438636872
   }
}
```


Void

Request:


```
{
   "transaction-request": {
      "version": "3.1.1.1",
      "verification": {
         "merchantId": 11111,
         "merchantKey": "key"
      },
      "order": {
         "void": {
            "transactionID": 463499,
            "ipAddress": "123.123.123.123"
         }
      }
   }
}
```


Response:


```
{
   "transaction-response": {
      "errorMessage": "",
      "authCode": "",
      "responseCode": 0,
      "orderID": "",
      "cvvResponseCode": "",
      "transactionID": 464470,
      "processorMessage": "APPROVED",
      "avsResponseCode": "",
      "referenceNum": "",
      "responseMessage": "VOIDED",
      "processorCode": "A",
      "merchantId": 11111,
      "transactionTimestamp": ""
   }
}
```


PIN Debit Sale

Request:


```
{
   "transaction-request": {
      "version": "3.1.1.1",
      "verification": {
         "merchantId": 11111,
         "merchantKey": "key"
      },
      "order": {
```

```
        "debitSale": {
          "shipping": {
            "name": "Susan Johnson",
            "address": "123 Main Street",
            "address2": "Apt 36",
            "city": "Moorpark",
            "state": "CA",
            "country": "US",
            "postalcode": 91307,
            "phone": "818-123-1234",
            "email": "customer@example.com"
          },
          "billing": {
            "name": "Susan Johnson",
            "address": "123 Main Street",
            "address2": "Apt 36",
            "city": "Moorpark",
            "state": "CA",
            "country": "US",
            "postalcode": 91307,
            "phone": "818-123-1234",
            "email": "customer@example.com"
          },
          "payment": {
            "chargeTotal": 32.0,
            "shippingTotal": 3.0,
            "salesTaxTotal": 1.23,
            "currencyCode": "USD"
          },
          "transactionDetail": {
            "payType": {
              "debitCard": {
                "track1Data": "%B6011000995500000^ Test Card^15121015432112345678?",
                "track2Data": ";6011000995500000=15121015432112345678?",
                "pinBlock": "273A61283641823",
                "keySerialNo": "10002320930000",
                "eCommInd": "retail"
              }
            }
          },
          "invoiceNumber": 123456,
          "referenceNum": 846392232,
          "ipAddress": "123.123.123.123"
        }
      }
    }
}
```


Response:


```
{
   "transaction-response": {
      "errorMessage": "",
      "authCode": 123456,
      "responseCode": 0,
```

```
      "orderID": "C0A8631F:014EF585F951:1DEB:6820377B",
      "cvvResponseCode": "M",
      "transactionID": 464472,
      "processorMessage": "APPROVED",
      "eCommInd": "retail",
      "avsResponseCode": "YYY",
      "referenceNum": 846392232,
      "responseMessage": "CAPTURED",
      "processorCode": "A",
      "merchantId": 11111,
      "transactionTimestamp": 1438638275
   }
}
```


PIN Debit Return

Request:


```
{
   "transaction-request": {
      "version": "3.1.1.1",
      "verification": {
         "merchantId": 11111,
         "merchantKey": "key"
      },
      "order": {
         "debitReturn": {
            "billing": {
               "name": "Susan Johnson",
               "address": "123 Main Street",
               "address2": "Apt 36",
               "city": "Moorpark",
               "state": "CA",
               "country": "US",
               "postalcode": 91307,
               "phone": "818-123-1234",
               "email": "customer@example.com"
            },
            "payment": {
               "chargeTotal": 32.0,
               "shippingTotal": 3.0,
               "salesTaxTotal": 1.23,
               "currencyCode": "USD"
            },
            "transactionDetail": {
               "payType": {
                  "debitCard": {
                     "track1Data": "%B6011000995500000^ Test Card^15121015432112345678?",
                     "track2Data": ";6011000995500000=15121015432112345678?",
                     "pinBlock": "273A61283641823",
                     "keySerialNo": "10002320930000",
                     "eCommInd": "retail"
                  }
               }
```

```
            },
            "invoiceNumber": 123456,
            "ipAddress": "123.123.123.123",
            "referenceNum": 846392232,
            "orderID": "C0A8631F:014EF5B87E1C:3B4C:69B6EC01"
        }
    }
  }
}
```


Response:


```
{
  "transaction-response": {
    "errorMessage": "",
    "authCode": "",
    "responseCode": 0,
    "orderID": "C0A8631F:014EF5B87E1C:3B4C:69B6EC01",
    "cvvResponseCode": "",
    "transactionID": 464485,
    "processorMessage": "APPROVED",
    "eCommInd": "retail",
    "avsResponseCode": "",
    "referenceNum": 846392232,
    "responseMessage": "CAPTURED",
    "processorCode": "A",
    "merchantId": 11111,
    "transactionTimestamp": 1438641622
  }
}
```


PIN Debit Void

Request:


```
{
  "transaction-request": {
    "version": "3.1.1.1",
    "verification": {
      "merchantId": 11111,
      "merchantKey": "key"
    },
    "order": {
      "debitVoid": {
        "transactionDetail": {
          "payType": {
            "debitCard": {
              "track1Data": "%B6011000995500000^ Test Card^15121015432112345678?",
              "track2Data": ";6011000995500000=15121015432112345678?",
              "pinBlock": "273A61283641823",
              "keySerialNo": "10002320930000",
              "eCommInd": "retail"
            }
```

```
            }
          },
          "transactionID": 464487,
          "ipAddress": "123.123.123.123"
        }
      }
    }
}
```


Response:


```
{
   "transaction-response": {
      "errorMessage": "",
      "authCode": "",
      "responseCode": 0,
      "orderID": "",
      "cvvResponseCode": "",
      "transactionID": 464487,
      "processorMessage": "APPROVED",
      "eCommInd": "retail",
      "avsResponseCode": "",
      "referenceNum": "",
      "responseMessage": "VOIDED",
      "processorCode": "A",
      "merchantId": 11111,
      "transactionTimestamp": ""
   }
}
```


eCheck

Request:


```
{
   "transaction-request": {
     "version": "3.1.1.1",
     "verification": {
        "merchantId": 11111,
        "merchantKey": "key"
     },
     "order": {
        "clientData": {
           "comments": "comment"
        },
        "sale": {
           "shipping": {
              "name": "Susan Johnson",
              "address": "123 Main Street",
              "address2": "Apt 36",
              "city": "Moorpark",
              "state": "CA",
              "country": "US",
              "postalcode": 91307,
```

```
            "phone": "818-123-1234",
            "email": "customer@example.com"
         },
         "billing": {
            "name": "Susan Johnson",
            "address": "123 Main Street",
            "address2": "Apt 36",
            "city": "Moorpark",
            "state": "CA",
            "country": "US",
            "postalcode": 91307,
            "phone": "818-123-1234",
            "email": "customer@example.com"
         },
         "payment": {
            "chargeTotal": 32.0,
            "shippingTotal": 3.0,
            "salesTaxTotal": 1.23,
            "currencyCode": "USD"
         },
         "transactionDetail": {
            "payType": {
               "ach": {
                  "achAccountType": "PC",
                  "bankRoutingNumber": 111111111,
                  "achAccountNumber": 12345678,
                  "achPaymentType": "PPD",
                   "achEffectiveDate": "2019-02-01 00:00:00",
                  "bankName": "Wells",
                  "bankState": "CA",
                  "checkNumber": 1200,
                  "frontImage": "Base64EncodedImageBinary",
                  "backImage": " Base64EncodedImageBinary ",
                  "MICR": "1234",
                  "achBusinessName": "my business"
               }
            }
         },
         "referenceNum": 846392231,
         "ipAddress": "123.123.123.123"
      }
   }
}
```


Response:


```
{
   "transaction-response": {
      "errorMessage": "",
      "authCode": 123456,
      "responseCode": 0,
      "orderID": "C0A8631F:014EF5C51150:18EC:49EB3221",
      "cvvResponseCode": "M",
      "transactionID": 464490,
```

```
      "processorMessage": "APPROVED",
      "avsResponseCode": "YYY",
      "referenceNum": 846392231,
      "responseMessage": "ACCEPTED",
      "processorCode": "A",
      "merchantId": 11111,
      "transactionTimestamp": 1438642409
   }
}
```


ACH Save on file

Request:


```
{
   "transaction-request": {
      "version": "3.1.1.1",
      "verification": {
         "merchantId": 11111,
         "merchantKey": "key"
      },
      "order": {
         "clientData": {
            "comments": "comment"
         },
         "sale": {
            "shipping": {
               "name": "Susan Johnson",
               "address": "123 Main Street",
               "address2": "Apt 36",
               "city": "Moorpark",
               "state": "CA",
               "country": "US",
               "postalcode": 91307,
               "phone": "818-123-1234",
               "email": "customer@example.com"
            },
            "billing": {
               "name": "Susan Johnson",
               "address": "123 Main Street",
               "address2": "Apt 36",
               "city": "Moorpark",
               "state": "CA",
               "country": "US",
               "postalcode": 91307,
               "phone": "818-123-1234",
               "email": "customer@example.com"
            },
            "payment": {
               "chargeTotal": 32.0,
               "shippingTotal": 3.0,
               "salesTaxTotal": 1.23,
               "currencyCode": "USD"
            },
            "transactionDetail": {
```

```
            "payType": {
               "ach": {
                  "achAccountType": "PC",
                  "bankRoutingNumber": 111111111,
                  "achAccountNumber": 123456789
               }
            }
         },
         "saveOnFile": {
               "customerToken": "7947",
               "onFileEndDate": "11/11/2030",
               "onFilePermissions": "ongoing",
               "onFileComment": "onfile",
               "onFileMaxChargeAmount": "200"
            },
         "referenceNum": 846392231,
         "ipAddress": "123.123.123.123"
      }
   }
}
```

Response:

```
{
   "transaction-response": {
      "errorMessage": "",
      "authCode": 123456,
      "responseCode": 0,
      "orderID": "C0A8631F:014EF5CD3C5E:6672:71378B53",
      "cvvResponseCode": "M",
      "transactionID": 464494,
      "processorMessage": "APPROVED",
      "save-on-file": {
         "token": "vjMH9HBJoMQ="
      },
      "avsResponseCode": "YYY",
      "referenceNum": 846392231,
      "responseMessage": "ACCEPTED",
      "processorCode": "A",
      "merchantId": 11111,
      "transactionTimestamp": 1438642945
   }
}
```

Setting up a Recurring Payment

Request:

```
{
   "transaction-request": {
      "version": "3.1.1.1",
      "verification": {
         "merchantId": 11111,
         "merchantKey": "key"
      },
```

```
"order": {
   "recurringPayment": {
      "shipping": {
         "name": "Susan Johnson",
         "address": "123 Main Street",
         "address2": "Apt 36",
         "city": "Moorpark",
         "state": "CA",
         "country": "US",
         "postalcode": 91307,
         "phone": "818-123-1234",
         "email": "customer@example.com"
      },
      "billing": {
         "name": "Susan Johnson",
         "address": "123 Main Street",
         "address2": "Apt 36",
         "city": "Moorpark",
         "state": "CA",
         "country": "US",
         "postalcode": 91307,
         "phone": "818-123-1234",
         "email": "customer@example.com"
      },
      "payment": {
         "chargeTotal": 32.0,
         "shippingTotal": 3.0,
         "salesTaxTotal": 1.23,
         "currencyCode": "USD"
      },
      "transactionDetail": {
         "payType": {
            "creditCard": {
               "track1Data": "%B6011000995500000^Test Card^15121015432112345678?",
               "track2Data": ";6011000995500000=15121015432112345678?",
               "number": "",
               "expMonth": "",
               "expYear": "",
               "cvvNumber": "",
               "cvvInd": "",
               "eCommInd": "retail"
            }
         }
      },
      "recurring":{
         "action": "new",
         "period": "daily",
         "installments": 2,
         "startDate": "2015-08-03",
         "failureThreshold": 3,
         "frequency": 2
      },
      "referenceNum": 846392232,
      "invoiceNumber": 123456,
```

```
            "ipAddress": "123.123.123.123"
         }
      }
   }
}
```

Response:

```
{
   "transaction-response": {
      "errorMessage": "",
      "authCode": 123456,
      "responseCode": 0,
      "orderID": "C0A8631F:014EF5D891EF:9432:250A58E8",
      "cvvResponseCode": "M",
      "number": "",
      "transactionID": 464499,
      "processorMessage": "APPROVED",
      "eCommInd": "retail",
      "avsResponseCode": "YYY",
      "referenceNum": 846392232,
      "responseMessage": "CAPTURED",
      "processorCode": "A",
      "merchantId": 11111,
      "transactionTimestamp": 1438643688
   }
}
```

8. Non-Transactional APIs

Adding and Deleting Customers

In order to save a card (or account) on file, you must first add the consumer to your customer list. To add a consumer, you use the <add-consumer> method.

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

https://apiint.paymentsite.com/UniversalAPI/postAPI

INPUT XML Structure for Adding a Consumer:

```
<api-request>
    <verification>
        <merchantId/> <!-- Required Field -->
        <merchantKey/> <!-- Required Field -->
    </verification>
    <command>add-consumer</command> <!-- Required Field -->
    <request>
        <customerIdExt/> <!-- Required Field. This field must be unique. -->
        <firstName/> <!-- Required Field -->
```

```
        <middleName/>
        <lastName/> <!-- Required Field -->
        <address1/>
        <address2/>
        <city/>
        <state/>
        <zip/>
        <country/>
<phone/>
        <email/>
        <dob/>
        <alternatePhone/>
    </request>
</api-request>
```

Sample Input Code for Adding a Customer

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.

Note that the customerId field is returned in the response, which is a unique number assigned to this customer by our system. Always use the customerId field when you need to uniquely identify this customer (as in when you set up a card on file for the customer).

```
<api-request>
    <verification>
        <merchantId>12345</merchantId>
        <merchantKey>key</merchantKey>
    </verification>
    <command>add-consumer</command>
    <request>
        <customerIdExt>123456</customerIdExt>
        <firstName></firstName>
        <middleName></middleName>
        <lastName>Test</lastName>
        <address1>123 Main St.</address1>
        <address2></address2>
        <city>Moorpark</city>
        <state>CA</state>
        <zip>91311</zip>
        <phone>123-123-1234</phone>
        <email>customer@example.com</email>
        <dob>12/31/1970</dob>
        <ssn>111-11-1111</ssn>
        <sex>M</sex>
    </request>
</api-request>
```

Sample Accepted Response from an *Add Consumer* Request

If the response code is 0, the customer was added successfully.

```
<api-response>
    <errorCode>0</errorCode>
    <errorMessage />
    <command>add-consumer</command>
    <time>1262352122049</time>
    <result>
    <customerId>74</customerId>
    </result>
</api-response>
```

The ***customer ID*** is a key piece of information—it is the ID number you always use to refer to this customer in transactions. You also need it when you want to update the customer's information.

Deleting a Consumer from the Customer List

If you need to delete a consumer from your customer list, use the `<delete-consumer>` method.

INPUT XML Structure for Deleting a Consumer:

```
<api-request>
    <verification>
        <merchantId/>
        <merchantKey/>
    </verification>
    <command>delete-consumer</command>
    <request>
        <customerId/>
            <deleteCards/>
    </request>
</api-request>
```

Sample Input Code for Deleting a Consumer and Their Cards 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>12345</merchantId>
        <merchantKey>key</merchantKey>
    </verification>
    <command>delete-consumer</command>
    <request>
        <customerId>11240</customerId>
            <deleteCards>Y</deleteCards>
    </request>
</api-request>
```

Sample Accepted Response from a *Delete Consumer* Request

If the response code is 0, the consumer was successfully deleted.

```
<api-response>
    <errorCode>0</errorCode>
    <errorMessage/>
    <command>delete-consumer</command>
    <time>120123123123123</time>
    <result/>
</api-response>
```

Updating Consumer Information

To allow you to modify a consumer's information, there is an update-consumer command. When you use this command, send the data elements you wish to modify along with the customerID. Any tags you do NOT include will remain the same. You may include any of the data elements shown in the tags below, but you MUST include the customerID provided to you when you added the consumer.

INPUT XML Structure for Updating a Consumer:

```
<api-request>
    <verification>
        <merchantId/>
        <merchantKey/>
    </verification>
    <command>update-consumer</command>
    <request>
        <customerId/>
        <customerIdExt/>
        <middleName/>
        <address1/>
        <address2/>
        <city/>
        <state/>
        <zip/>
        <phone/>
        <email/>
    </request>
</api-request>
```

Sample Input Code for Updating a Customer

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.

Note that the customerId field is returned in the response, which is a unique number assigned to this customer by our system. Always use the customerId field when you need to uniquely identify this customer (as in when you set up a card on file for the customer).

When updating a customer's information, only the fields provided will be updated. All other fields will be left as-is.

```
<api-request>
    <verification>
        <merchantId>12345</merchantId>
        <merchantKey>key</merchantKey>
    </verification>
    <command>update-consumer</command>
    <request>
        <customerId>11240</customerId>
        <customerIdExt>11123</customerIdExt>
        <address1>123 Main St.</address1>
    </request>
</api-request>
```

Sample Accepted Response from an *Update Consumer* Request

If the response code is 0, the consumer was successfully updated.

```
<api-response>
    <errorCode>0</errorCode>
    <errorMessage></errorMessage>
    <command>update-consumer</command>
    <time>120123123123123</time>
    <result/>
</api-response>
```

Saving a Card (or account) on File

You use the add-card-onfile 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.paymentsite.com/UniversalAPI/postAPI

NOTE: It is NOT required to have a Customer On File in Paymentsite to add a Card or Account On File.

If you choose to associate a Customer On File with a Token (card or account on file), please see the ***Adding and Deleting Customers*** section for instructions. There is also a way to Save a Customer and the Card on File in one call – see the ***Saving a Customer with Card or Bank Account on File*** section.

For added security of payment information, software based encryption is supported when saving a card or bank account on file. <deviceKSN> will be given to each merchant along with a public key. <deviceType> should be MGSRSA for all software based encryption. <creditCard> and <achAccountNumber> accept encrypted values (RSA 2048) PKCS #8. Intrix will provide a public key for encryption purposes.

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

```
<api-request>
    <verification>
        <merchantId/>
        <merchantKey/>
    </verification>
    <command>add-card-onfile</command>
    <request>
        <customerId/>
        <creditCardNumber encrypted="Y"/>
        <!-- if the card data is encrypted, set encrypted="Y". If it's not, set encrypted="N" or leave it out -->
        <expirationMonth/>
        <expirationYear/>
        <deviceType/>
        <!-- required if the card or account data is encrypted -->
        <deviceKSN/>
        <!-- required if the card or account data is encrypted -->
        <billingName/>
        <billingAddress1/>
        <billingAddress2/>
        <billingCity/>
        <billingState/>
        <billingZip/>
        <billingCountry/>
        <billingPhone/>
        <billingEmail/>
        <onFileEndDate/>
        <onFilePermissions/>
        <onFileComment/>
        <onFileMaxChargeAmount/>
    </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.

NOTE: It is NOT required to have a Customer On File in Paymentsite to add a Card or Account On File.

```
<api-request>
    <verification>
        <merchantId>11</merchantId>
        <merchantKey>key</merchantKey>
    </verification>
    <command>add-card-onfile</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>
        <billingPhone>805-223-9918</billingPhone>
        <billingEmail>customer@example.com</billingEmail>
        <onFileEndDate>12/31/2013</onFileEndDate>
        <onFileMaxChargeAmount>1000.00</onFileMaxChargeAmount>
    </request>
</api-request>
```

Sample Accepted Response from a Save Card on File Request

If the response code is 0, the card information was successfully saved.

```
<api-response>
    <errorCode>0</errorCode>
    <errorMessage></errorMessage>
    <command>add-card-onfile</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.

NOTE: It is NOT required to have a Customer On File in Paymentsite to add a Card or Account On File.

```
<api-request>
    <verification>
        <merchantId>00</merchantId>
        <merchantKey>key</merchantKey>
    </verification>
    <command>add-ach-onfile</command>
    <request>
        <customerId>1122</customerId>
        <bankRoutingNumber>123456789</bankRoutingNumber>
        <achAccountType>PC</achAccountType>
        <achAccountNumber encrypted="N">11223344</achAccountNumber>
         <!-- if the account data is encrypted, set encrypted="Y". If it's not, set encrypted="N" or leave it out --
>
        <deviceType/>
<!-- required if the account data is encrypted. Do not include this tag if not encrypted -->
        <deviceKSN/>
<!-- required if the account data is encrypted. Do not include this tag if not encrypted -->
        <billingName>Joe Consumer</billingName>
        <billingAddress1>123 Main St.< /billingAddress1>
        <billingAddress2></billingAddress2>
        <billingCity>Moorpark</billingCity>
        <billingState>CA</billingState>
        <billingZip>91311</billingZip>
        <billingCountry>US</billingCountry>
        <billingPhone>805-223-9918</billingPhone>
        <billingEmail>customer@example.com</billingEmail>
    </request>
</api-request>
```

Updating the payment info for an ACH account onfile using the token

You use the token-update-ach-account-info method when you wish to update the account number, routing number, or account type for an on file ach account that is referenced from a token.

The posting URL is: https://apiint.paymentsite.com/UniversalAPI/postAPI

Before you can use this method, you must already have a token that was generated by the system from a previously added account on file. See the section titled ***Saving a Card (or account) on File*** for how to add an ach account on file. For added security of payment information, software based encryption is supported when passing the bank account number. <deviceKSN> will be given to each merchant along with a public key.

<deviceType> should be MGSRSA for all software based encryption. <achAccountNumber> accept encrypted values (RSA 2048) PKCS #8. Intrix will provide a public key for encryption purposes.

Notes:

1. Certain fields cannot be pdated to an empty value. These include bankRoutingNumber, achAccountNumber and achAccountType.
2. If you omit passing a tag entirely, that field will be left unchanged in the system.

INPUT XML Structure for a updating an ach account on file:

```
<api-request>
  <verification>
    <merchantId>00</merchantId>
    <merchantKey>key</merchantKey>
  </verification>
  <command>token-update-ach-account-info</command>
  <request>
    <token>xzyabc123</token>
    <customerIdExt>123123</customerIdExt>
    <bankRoutingNumber>123456789</bankRoutingNumber> <!-- empty value is not allowed -->
    <achAccountType>PS</achAccountType>  <!-- empty value is not allowed -->
    <achAccountNumber encrypted="N">11223344</achAccountNumber>   <!-- empty value is not allowed -->
    <billingName>test</billingName>
    <billingAddress1>123 Main</billingAddress1>
    <billingAddress2>Suite A</billingAddress2 >
    <billingCity>Moorpark</billingCity>
    <billingState>CA</billingState>
    <billingZip>91111</billingZip>
    <billingCountry>US</billingCountry>
    <billingPhone>1231231234</billingPhone>
    <billingEmail>customer@example.com</billingEmail>
    <onFileEndDate>12/31/2015</onFileEndDate>
    <onFilePermissions>ongoing</onFilePermissions>
    <onFileComment>Test onfile comment</onFileComment>
    <onFileMaxChargeAmount>100</onFileMaxChargeAmount>
      <!-- if the account data is encrypted, set encrypted="Y". If it's not, set encrypted="N" or leave it out -->
    <deviceType/> <!-- required if the card or account data is encrypted -->
    <deviceKSN/> <!-- required if the card or account data is encrypted -->
  </request>
</api-request>
```

Sample Accepted Response for above request

If the response code is 0, the account information was successfully updated.

```
<api-response>
  <errorCode>0</errorCode>
  <errorMessage></errorMessage>
  <command>token-update-ach-account-info</command>
  <time>12012222222222</time>
  <result/>
</api-response>
```

Sample Error Response for above request

If the response code is 1, the account information was not updated.

```
<api-response>
  <errorCode>1</errorCode>
  errorMessage>token not found</errorMessage>
  <command>token-update-ach-account-info</command>
  <time>12012222222222</time>
  <result/>
</api-response>
```

Updating the payment info for a Credit account onfile using the token

You use the token-update-credit-account-info method when you wish to update the credit card number, expiration month, or expiration year for an on file credit account that is referenced from a token.

The posting URL is: https://apiint.paymentsite.com/UniversalAPI/postAPI

Before you can use this method, you must already have a token that was generated by the system from a previously added account on file. See the section titled ***Saving a Card (or account) on File*** for how to add a credit account on file. For added security of payment information, software based encryption is supported when passing the bank account number. <deviceKSN> will be given to each merchant along with a public key.

<deviceType> should be MGSRSA for all software based encryption. <creditCardNumber> accept encrypted values (RSA 2048) PKCS #8. Intrix will provide a public key for encryption purposes.

Certain fields cannot be pdated to an empty value. These include creditCardNumber, expirationMonth, expirationYear, billingName, billingAddress1 and billingZip.

Notes:

1. Certain fields cannot be pdated to an empty value. These include bankRoutingNumber, achAccountNumber and achAccountType.
2. If you omit passing a tag entirely, that field will be left unchanged in the system.

INPUT XML Structure for updating a credit account on file:

```
<api-request>
  <verification>
    <merchantId>00</merchantId>
    <merchantKey>key</merchantKey>
  </verification>
  <command>token-update-credit-account-info</command>
  <request>
```

```
    <token>xzyabc123</token>
    <customerIdExt>123123</customerIdExt>
    <creditCardNumber    encrypted=”Y”/>  <!-- empty value is not allowed --> <!-- if the card data is
encrypted,set encrypted=”Y”. If it’s not, set encrypted=”N” or leave it out -->
    <expirationMonth>12</expirationMonth>  <!-- empty value is not allowed -->
    <expirationYear>2019</expirationYear/>  <!-- empty value is not allowed -->
    <billingName>test</billingName>  <!-- empty value is not allowed -->
    <billingAddress1>123 Main</billingAddress1>   <!-- empty value is not allowed -->
    <billingAddress2>Suite A</billingAddress2 >
    <billingCity>Moorpark</billingCity>
    <billingState>CA</billingState>
    <billingZip>91111</billingZip>  <!-- empty value is not allowed -->
    <billingCountry>US</billingCountry>
    <billingPhone>1231231234</billingPhone>
    <billingEmail>customer@example.com</billingEmail>
    <onFileEndDate>12/31/2015</onFileEndDate>
    <onFilePermissions>ongoing</onFilePermissions>
    <onFileComment>Test onfile comment</onFileComment>
    <onFileMaxChargeAmount>100</onFileMaxChargeAmount>
   <deviceType/> <!-- required if the card or account data is encrypted -->
   <deviceKSN/> <!-- required if the card or account data is encrypted -->
  </request>
</api-request>
```

Sample Accepted Response for above request

If the response code is 0, the card information was successfully updated.

```
<api-response>
  <errorCode>0</errorCode>
  <errorMessage></errorMessage>
  <command>token-update-credit-account-info</command>
  <time>12012222222222</time>
  <result/>
</api-response>
```

Sample Rejected Response for above request

If the response code is 1, the card information was not successfully updated.

```
<api-response>
  <errorCode>1</errorCode>
  <errorMessage>token not found</errorMessage>
  <command>token-update-credit-account-info</command>
  <time>12012222222222</time>
  <result/>
</api-response>
```

Verifying a Token and associate payment info

You use the token-verify method when you wish to verify that a token value is valid, and verify that the passed payment info is associated with the token. If the token is valid and the passed payment info is associated with the token then all customers that are currently using this card number or ach account info are also returned.

The posting URL is: https://apiint.paymentsite.com/UniversalAPI/postAPI

Before you can use this method, you must already have a token that was generated by the system from a previously added account on file. See the section titled ***Saving a Card (or account) on File*** for how to add a credit or ach account on file to obtain a token. For added security of payment information, software based encryption is supported when passing a card number or bank account number. <deviceKSN> will be given to each merchant along with a public key.

<deviceType> should be MGSRSA for all software based encryption. <creditCardNumber> accept encrypted values (RSA 2048) PKCS #8. Intrix will provide a public key for encryption purposes.

INPUT XML Structure for verifying the credit payment info associated with a token:

```
<api-request>
  <verification>
    <merchantId>00</merchantId>
    <merchantKey>key</merchantKey>
  </verification>
  <command>token-verify</command>
  <request>
    <token>xzyabc123</token>
    <customerIdExt>123123</customerIdExt>
    <creditCardNumber encrypted="Y"/> <!-- if the card data is encrypted, set encrypted="Y". If it's not encrypted omit the reference -->
    <expirationMonth>12</expirationMonth>
    <expirationYear>2019</expirationYear>
    <deviceType/> <!-- required if the card or account data is encrypted -->
    <deviceKSN/> <!-- required if the card or account data is encrypted -->
  </request>
</api-request>
```

INPUT XML Structure for verifying the ach payment info associated with a token:

```
<api-request>
  <verification>
    <merchantId>00</merchantId>
    <merchantKey>key</merchantKey>
  </verification>
  <command>token-verify</command>
  <request>
    <token>xzyabc123</token>
    <customerIdExt>123123</customerIdExt>
    <bankRoutingNumber>123456789</bankRoutingNumber>
```

```
    <achAccountNumber   encrypted=”N”>11223344</achAccountNumber>
     <!-- if the account data is encrypted, set encrypted=”Y”. If it’s not, set encrypted=”N” or leave it out -->
    <deviceType/> <!-- required if the card or account data is encrypted -->
    <deviceKSN/> <!-- required if the card or account data is encrypted -->
  </request>
</api-request>
```

Sample match found response for above request

A response code of 0 indicates that the request was successfully processed. The <match> tag value of Y indicates that a match was found for the pased token and payment info, and all customers associated with that card number or ach account info are also returned.

```
<api-response>
  <errorCode>0</errorCode>
  <errorMessage></errorMessage>
  <command>token-verify</command>
  <time>12012222222222</time>
  <result>
    <match>Y</match>
    <customerIdExt>123123</customerIdExt>
    <customerIdExt>456456</customerIdExt>
    <customerIdExt>789789</customerIdExt>
  </result>
</api-response>
```

Sample no match response for above request

A response code of 0 indicates that the request was successfully processed. The <match> tag value of N indicates that no match was found for passed token and payment info.

```
<api-response>
  <errorCode>0</errorCode>
  <errorMessage></errorMessage>
  <command>token-verify</command>
  <time>12012222222222</time>
  <result>
    <match>N</match>
  </result>
</api-response>
```

Sample error response for above request

A response code of 1 indicates that an error occurred. The error message 'bad token' indicates that the token value itself is not valid.

```
<api-response>
  <errorCode>1</errorCode>
  <errorMessage>bad token</errorMessage>
  <command>token-verify</command>
  <time>12012222222222</time>
```

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

Inactivate a Card (or account) on File

You use the token-inactivation method when you wish to Inactivate/activate a card (or account) on file. Use 'Y' for inactivationFlag to inactivate card (or account) on file for the given token. Use 'N' for inactivationFlag to activate card (or account) on file for the given token. The posting URL is: https://apiint.paymentsite.com/UniversalAPI/postAPI

Before you can use this method, you must already have a token that was generated by the system from a previously added account on file. See the section titled ***Saving a Card (or account) on File*** for how to add an ach account on file.

INPUT XML Structure for a updating an ach account on file:

```
<api-request>
  <verification>
    <merchantId>00</merchantId>
    <merchantKey>key</merchantKey>
  </verification>
  <command> token-inactivation </command>
  <request>
    <token>xzyabc123</token>
    <customerIdExt>123123</customerIdExt>
    <inactivationFlag>Y</inactivationFlag>        <!-- empty value is not allowed -->
  </request>
</api-request>
```

Sample Accepted Response for above request

If the response code is 0, the token inactivation flag was successfully updated.

```
<api-response>
  <errorCode>0</errorCode>
  <errorMessage></errorMessage>
  <command> token-inactivation</command>
  <time>12012222222222</time>
  <result></result>
</api-response>
```

Sample Error Response for above request

If the response code is 1, token inactivation flag was not updated.

```
<api-response>
  <errorCode>1</errorCode>
  <errorMessage><![CDATA[token not found]]></errorMessage>
  <time>1395704808243</time>
```

```
</api-response>
```

Removing a Card (or account) on File

You use the <delete-card-onfile> 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.paymentsite.com/UniversalAPI/postAPI

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

```
<api-request>
    <verification>
        <merchantId/>
        <merchantKey/>
    </verification>
    <command>delete-card-onfile</command>
    <request>
        <customerId/>
        <token/>
    </request>
</api-request>
```

Sample Accepted Response from a Delete Card on File Request

If the response code is 0, the card on file was successfully deleted. The card will no longer be available for card on file transactions.

```
<api-response>
    <errorCode>0</errorCode>
    <errorMessage></errorMessage>
    <command>delete-card-onfile</command>
    <time>120123123123123</time>
    <result></result>
</api-response>
```

Saving a Card on File with a Credit Card Sale Transaction

Saving a Card on File along with a Sale transaction

When you perform a sale transaction, you can also save the card on file in the same operation. This returns a payment token which can be sent for future transactions with this customer and card in lieu of the card information. Saving a card on file can also be performed in a separate operation—see the ***Saving a Card on File*** section later in this document for further details, posting URL and XML structure.

Before you can save a card or account on file for a customer, you must first add the customer. See the ***Adding and Deleting Customers*** section for instructions.

INPUT XML Structure for a Credit Card Sale with Save Card on File

```
<?xml version="1.0" encoding="UTF-8" ?>
    <transaction-request>
    <verification>
        <merchantId></merchantId>
        <merchantKey></merchantKey>
    </verification>
    <order>
        <sale>
            <referenceNum></referenceNum>
            <ipAddress></ipAddress>
            <customerId></customerId>
            <invoiceNumber/>
            <billing>
                <name></name>
                <address></address>
                <address2/>
                <city></city>
                <state></state>
                <postalcode></postalcode>
                <country></country>
                <phone></phone>
                <email></email>
            </billing>
            <shipping>
                <name></name>
                <address></address>
                <address2/>
                <city></city>
                <state></state>
                <postalcode></postalcode>
                <country></country>
                <phone></phone>
                <email></email>
            </shipping>
            <transactionDetail>
                <payType>
                    <creditCard>
                        <number></number>
                        <expMonth></expMonth>
                        <expYear></expYear>
                        <cvvInd/>
                        <cvvNumber></cvvNumber>
                        <track1Data/>
                        <track2Data/>
                        <eCommInd></eCommInd>
                        <signatureImage></signatureImage>
                        <!-- if included, must contain a value -->
                    </creditCard>
                </payType>
```

```
            </transactionDetail>
            <payment>
                <chargeTotal></chargeTotal>
                <currencyCode/>
                <!-- if included, must contain a value. If NOT included, defaults to USD -->
            </payment>
            <saveOnFile>
                <customerToken></customerToken>
                <-- populate this field with the customerId returned from the add-consumer function -->
                <onFileEndDate></onFileEndDate>
                <onFilePermissions></onFilePermissions>
                <onFileComment></onFileComment>
                <onFileMaxChargeAmount></onFileMaxChargeAmount>
            </saveOnFile>
        </sale>
    </order>
</transaction-request>
```

Sample Response Code from Credit Card Sale with Save Card on File

If the response code is **0**, the transaction was successful. (Indicators that the transaction succeeded are shown in **bold blue**.) The <token> sent back in the <save-onfile> tag is the payment token to be used in lieu of the credit card data on subsequent transactions with this customer and card.

```
<transaction-response>
    <authCode>016361</authCode>
    <orderID>0AF90446:012BCCFB0D3D:1263:017494F9</orderID>
    <referenceNum>1</referenceNum>
    <transactionID>876269</transactionID>
    <transactionTimestamp>1287634226</transactionTimestamp>
    <responseCode>0</responseCode>
    <responseMessage>CAPTURED</responseMessage>
    <avsResponseCode>0</avsResponseCode>
    <cvvResponseCode>P</cvvResponseCode>
    <processorCode>029301549965</processorCode>
    <processorMessage>Approved</processorMessage>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <errorMessage />
    <save-onfile>
        <token>I28P65RNklg=</token>
    </save-onfile>
</transaction-response>
```

Saving a Customer with Card or Bank Account on File

Saving a Card or Bank Account on File while adding a customer

When you add a new customer, you can also save the card or bank account on file in the same operation.

This returns a payment token which can be sent for future transactions with this customer and card in lieu of the card information. Saving a card or bank account on file can also be performed in a separate operation—see the ***Saving a Card on File*** section for further details, posting URL and XML structure.

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

https://apiint.paymentsite.com/UniversalAPI/postAPI

For added security of payment information, software based encryption is supported when saving a card or bank account on file. <deviceKSN> will be given to each merchant along with a public key.

<deviceType> should be MGSRSA for all software based encryption. <creditCard> and

<achAccountNumber> accept encrypted values (RSA 2048) PKCS #8. Intrix will provide a public key for encryption purposes.

INPUT XML Structure for Adding a Customer and Adding a Card or Account on File in One Call

```
<api-request>
    <verification>
        <merchantId/>
        <merchantKey/>
    </verification>
    <command>add-consumer-onfile</command>
    <request>
        <customerIdExt/>
        <firstName/>
        <middleName/>
        <lastName/>
        <address1/>
        <address2/>
        <city/>
        <state/>
        <zip/>
        <country/>
        <phone/>
        <email/>
        <dob/>
        <ssn/>
        <sex/>
        <alternatePhone/>
        <approvedTxnEmail/>
        <declinedTxnEmail/>
        <recAdvNoticeEmail/>
        <recAdvNoticeThreshold/>
        <ccOnFileExpiryEmail/>
        <ccOnFileExpiryThreshold/>
        <onFileType>CREDIT</onFileType>
        <!-- must be set to CREDIT or ACH -->
```

```
        <!-- Encryption - Required fields if sending card or account data in encrypted format -->
        <deviceType/>
        <deviceKSN/>
        <!-- "On File Fields – must include EITHER Credit OR ACH fields, not both" -->
        <creditCardNumber encrypted="Y"/>
        <!-- if data is NOT encrypted, set encrypted="N" or leave out the encrypted attribute -->
        <expirationMonth/>
        <expirationYear/>
        <!-- ACH fields -->
        <bankRoutingNumber/>
        <achAccountNumber encrypted="Y"/>
        <!-- if data is NOT encrypted, set encrypted="N" or leave out the encrypted attribute -->
        <achAccountType/>
    </request>
</api-request>
```

Sample Response Code from Add Consumer with Save Card or Account on File

If the response code is **0**, the transaction was successful. (Indicators that the transaction succeeded are shown in **bold blue**.) The <token> sent back in the <result> tag is the payment token to be used in lieu of the credit card or ACH data on subsequent transactions with this customer.

```
<api-response>
    <errorCode> xs:string </errorCode> [1]
    <errorMessage> xs:string </errorMessage> [0..1]
    <command> xs:string </command> [0..1]
    <time> xs:string </time> [1]
    <result>
        <customerId> ... </customerId> [1]
        <token>I28P65RNklg=</token> [1]
    </result> [0..1]
    <firstName> ... </firstName> [0..1]
    <email> ... </email> [0..1]
    <alternatePhone> ... </alternatePhone> [0..1]
    <merchantKey> ... </merchantKey> [0..1]
    <customerIdExt> ... </customerIdExt> [0..1]
</api-response>
```

JSON Samples for Non-Transactions

Each JSON request sent to the Paymentsite Gateway servers and responses received from the servers have a standard format defined by a schema. The basic structure for non-transactions is shown below:

```
{
  "api-request": {
    "command": ...,
    "verification": {...},
    "request": {...}
```

```
  }
}
```

Response:

```
{
  "api-response": {
    ...
  }
}
```

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

```
{
  "api-error": {
    “errorCode”: ,
    “errorMsg”:””
  }
}
```

Json Non-Transaction Posting URL

The posting URL to use for **testing JSON** API **NON-transaction requests** is as follows. This URL is strictly for posting non-transaction requests such as add-consumer, add-card-on-file, token-update-credit-account-info, etc.

Posting URL for test NON-Transaction requests:

https://apiint.paymentsite.com/UniversalAPI/postAPI

The posting URL to use for **LIVE JSON** API **NON-transaction requests** is as follows. This URL is strictly for posting non-transaction requests such as add-consumer, add-card-on-file, token-update-credit-account-info, etc.

Posting URL for LIVE transaction requests:

https://api.paymentsite.com/UniversalAPI/postAPI

add-consumer

NOTE: It is **not** necessary to add a customer in order to create a Card OnFile (Token) in Paymentstie. Creation of a token can be done without associating that toke to a customer in the Paymentsite system, if an external system will house the relationship between a person and the card on file (or account on file).

```
{
```

```
  "api-request": {
    "command": "add-consumer",
    "verification": {
      "merchantId": "1111",
      "merchantKey": "key"
    },
    "request": {
      "customerIdExt": "746",
      "firstName": "Joe",
      "middleName": "",
      "lastName": "Consumer",
      "address1": "123 Main St.",
      "address2": "",
      "city": "Las Vegas",
      "state": "NV",
      "zip": "89044",
      "country": "US",
      "phone": "123-123-1234",
      "alternatePhone": "234-234-2345"
    }
  }
}
```

update-consumer

```
{
  "api-request": {
    "command": "update-consumer",
    "verification": {
      "merchantId": "1111",
      "merchantKey": "key"
    },
    "request": {
      "customerId": "23",
      "customerIdExt": "123456",
      "middleName": "A",
      "address1": "123 Main St.",
      "address2": "",
      "city": "Las Vegas",
      "state": "NV",
      "zip": "89044",
      "country": "US",
      "phone": "123-123-1234",
      "email": "test@test.com",
      "sex": "M",
      "alternatePhone": "234-234-2345"
    }
  }
}
```

add-card-onfile

NOTE: It is **not** necessary to add a customer in order to create a Card OnFile (Token) in Paymentstie. Creation of a token can be done without associating that toke to a customer in the Paymentsite

system, if an external system will house the relationship between a person and the card on file (or account on file).

```
{
   "api-request": {
     "command": "add-card-onfile",
      "verification": {
         "merchantId": "1111",
         "merchantKey": "key"
      },
      "request": {
         "customerId": "746",
         "creditCardNumber": "4111111111111111",
         "expirationMonth": "01",
         "expirationYear": "2025",
         "billingName": "Joe Consumer",
         "billingAddress1": "123 Main St.",
         "billingAddress2": "",
         "billingCity": "Las Vegas",
         "billingState": "NV",
         "billingZip": "89044",
         "billingCountry": "US",
         "billingPhone": "1231231234",
         "billingEmail": "test@test.com"
      }
   }
}
```

add-ach-onfile

```
{
   "api-request": {
     "command": "add-ach-onfile",
      "verification": {
         "merchantId": "11111",
         "merchantKey": "key"
      },
      "request": {
         "customerId": "746",
         "bankRoutingNumber": "121000248",
         "achAccountNumber": "123456789",
         "achAccountType": "PS",
         "billingName": "Joe Consumer",
         "billingAddress1": "123 Main St.",
         "billingAddress2": "",
         "billingCity": "Las Vegas",
         "billingState": "NV",
         "billingZip": "89044",
         "billingCountry": "US",
         "billingPhone": "1231231234",
         "billingEmail": "test@test.com"
      }
   }
}
```

token-update-credit-account-info

```
{
   "api-request": {
      "command": "token-update-credit-account-info",
      "verification": {
         "merchantId": "1111",
         "merchantKey": "key"
      },
      "request": {
         "token": "xzyabc123",
         "customerIdExt": "123123",
         "creditCardNumber": "4111111111111111",
         "expirationMonth": "12",
         "expirationYear": "2025",
         "billingName": "Joe Consumer",
         "billingAddress1": "123 street",
         "billingAddress2": "suite 108",
         "billingCity": "Las Vegas",
         "billingState": "NV",
         "billingZip": "89044",
         "billingCountry": "US",
         "billingPhone": "1231231234",
         "billingEmail": "test@test.com",
         "onFileEndDate": "12/31/2025",
         "onFilePermissions": "ongoing",
         "onFileComment": "test comment",
         "onFileMaxChargeAmount": "100"
      }
   }
}
```

All non-transaction operations are duplicated in the JSON API using the same tag names as are documented for XML.

9. Recurring Payments

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. Set up a recurring payment if you want Paymentsite to run Sale transactions for you automatically at a specified interval.

If your system manages transaction recurrence (you initiate the transactions each time) do not create a Recurring Payment in Paymentsite. In that case, include the flagAsRecurring tag in your XML to indicate that this is part of a recurring series.

(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>
    <recurringPayment>
      <referenceNum/>
        <ipAddress/>
        <invoiceNumber/>
        <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 transactionsreplace with ach fields for echeck 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/> <!-- Required if Heartland is the processor -->
              <deviceType/>
```

```
            <deviceKSN />
            <track1Data encrypted="Y"/>
              <!-- If the merchant is using an encrypted device for swiping the card, set encrypted = “Y” for all
track data that is sent. -->
            <track2Data encrypted="Y"/>
            <eCommInd/>
          </creditCard>
        </payType>
      </transactionDetail>
      <payment>
        <chargeTotal/>
        <salesTaxTotal/>
          <!-- sales tax on the order. If included, must contain a numeric value -->
        <shippingTotal/>
          <!-- shipping on the order. If included, must contain a numeric value -->
        <currencyCode/>
          <!-- if included, must contain a value. If NOT included, defaults to USD -->
      </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/>
      <processorReferenceNumber/>
      <processorTransactionID/>
<partiallyApprovedAmount/>
      <accountBalance/>
      <errorMessage/>
```

```
</transaction-response>
```
Or, if there is an error:
```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<api-error>
    <errorCode></errorCode>
    <errorMsg></errorMsg>
</api-error>
```

Modifying a Recurring Payment

Once you have created a scheduled payment (also known as a recurring transaction), if you need to change your scheduled payment, use the non-transactional posting URL and set the <command> tag to modify-recurring.

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

https://apiint.paymentsite.com/UniversalAPI/postAPI

Also, please include ONLY the tags you wish to update. If you send an empty tag (for example: <address1/>), the system will delete the data that was stored in that field (that is, the address1 field would be set to null).

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

INPUT XML Structure:

```
<api-request>
    <verification>
        <merchantId/>
        <merchantKey/>
    </verification>
    <command>modify-recurring</command>
    <request>
    <orderID/>
        <paymentInfo>
            <cardInfo>
                <creditCardNumber/>
                <expirationMonth/>
                <expirationYear/>
            </cardInfo>
            <!-- you can include one payment type only, so if you include credit card information, do NOT include ACH information (and vice versa). Also, note that if the payment information has not changed, there is no need to include it.
                <achInfo>
                    <bankRoutingNumber/>
                    <achAccountNumber/>
                </achInfo>
```

```
            -->
        </paymentInfo>
        <billingInfo>
            <name/>
            <!-- if name is included, this field MUST contain a value -->
            <address1/>
            <address2/>
            <city/>
            <zip/>
            <country/>
            <email/>
            <phone/>
        </billingInfo>
        <shippingInfo>
            <name/>
            <!-- if name is included, this field MUST contain a value -->
            <address1/>
            <address2/>
            <city/>
            <zip/>
            <country/>
            <email/>
            <phone/>
        </shippingInfo>
    </request>
</api-request>
```

OUTPUT XML Structure:

An errorCode of 0 means the operation was successful.

```
<api-response>
    <errorCode>0</errorCode>
    <errorMessage/>
    <command>modify-recurring</command>
    <time>1338857965625</time>
    <result/>
    <merchantKey>key</merchantKey>
</api-response>
```

Or, if there is an error:

```
<api-response>
    <errorCode>1</errorCode>
    <errorMessage>
    <![CDATA[Parser Error: URI=null Line=1: cvc-complex-type.2.4.d: Invalid content was found starting
with element 'billingInfo'. No child element is expected at this point.]]></errorMessage>
    <time>1338857906850</time>
    <merchantKey>key</merchantKey>
</api-response>
```

Cancelling a Recurring Payment

Once you have created a scheduled payment (also known as a recurring transaction), if you need to cancel it your scheduled payment, use the non-transactional posting URL and set the <command> tag to ***cancelrecurring***.

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

https://apiint.paymentsite.com/UniversalAPI/postAPI

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

INPUT XML Structure:

```
<api-request>
    <verification>
        <merchantId/>
        <merchantKey/>
    </verification>
    <command>cancel-recurring</command>
    <request>
        <orderID/>
    </request>
</api-request>
```

Output XML Structure:

An errorCode of 0 means the operation was successful.

```
<api-response>
    <errorCode>0</errorCode>
    <errorMessage/>
    <time>1338849076127</time>
    <merchantKey>key</merchantKey>
</api-response>
Or, if there is an error:
<api-response>
    <errorCode>1</errorCode>
    <errorMessage><![CDATA[Invalid order id.]]></errorMessage>
    <time>1338854820434</time>
    <merchantKey>key</merchantKey>
</api-response>
```

10. Gift Cards

Activating a Gift Card

Gift cards are only currently supported on the Chase Paymentech platform. If you are using any other processor, please do not attempt to send any gift card requests.

To activate a gift card for a specified amount, use the `<giftCardActivate>` method.

The amount that the gift card should have on it after it is activated should be sent in the `<chargetotal>` field.

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

INPUT XML Structure:

```
<transaction-request>
    <verification>
        <merchantId></merchantId>
        <merchantKey> </merchantKey>
    </verification>
    <order>
        <giftCardActivate>
            <referenceNum></referenceNum>
            <ipAddress></ipAddress>
            <transactionDetail>
                <payType>
                    <giftCard>
                        <number></number>
                        <cvvNumber></cvvNumber>
                    </giftCard>
                </payType>
            </transactionDetail>
            <payment>
                <chargeTotal></chargeTotal>
            </payment>
        </giftCardActivate>
    </order>
</transaction-request>
```

Response XML structure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<transaction-response>
    <authCode></authCode>
    <orderID></orderID>
    <referenceNum></referenceNum>
    <transactionID></transactionID>
    <transactionTimestamp></transactionTimestamp>
    <responseCode></responseCode>
    <responseMessage></responseMessage>
    <avsResponseCode />
    <cvvResponseCode>P</cvvResponseCode>
    <processorCode>00</processorCode>
```

```
    <processorMessage></processorMessage>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <partiallyApprovedAmount/>
    <accountBalance/>
    <errorMessage />
    <accountBalance></accountBalance>
</transaction-response>
```

Or

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

Adding Value to a Gift Card

Gift cards are only currently supported on the Chase Paymentech platform. If you are using any other processor, please do not attempt to send any gift card requests.

To add value to a gift card, use the `<giftCardAddValue>` method.

The **<chargetotal>** is the field used to specify the amount to add to the card.

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

INPUT XML Structure:

```
<transaction-request>
    <verification>
        <merchantId></merchantId>
        <merchantKey> </merchantKey>
    </verification>
    <order>
        <giftCardAddValue>
            <referenceNum></referenceNum>
            <ipAddress></ipAddress>
            <transactionDetail>
                <payType>
                    <giftCard>
                        <number></number>
                        <cvvNumber></cvvNumber>
                    </giftCard>
                </payType>
            </transactionDetail>
            <payment>
                <chargeTotal></chargeTotal>
            </payment>
        </giftCardAddValue>
```

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

Response XML structure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<transaction-response>
    <authCode></authCode>
    <orderID></orderID>
    <referenceNum></referenceNum>
    <transactionID></transactionID>
    <transactionTimestamp></transactionTimestamp>
    <responseCode></responseCode>
    <responseMessage></responseMessage>
    <avsResponseCode />
    <cvvResponseCode></cvvResponseCode>
    <processorCode></processorCode>
    <processorMessage></processorMessage>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <errorMessage />
    <accountBalance></accountBalance>
</transaction-response>
```

Or

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

Checking the Balance on a Gift Card

Gift cards are only currently supported on the Chase Paymentech platform. If you are using any other processor, please do not attempt to send any gift card requests.

To check the balance on a gift card, use the `<giftCardBalanceInquiry>`.

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

INPUT XML Structure:

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

```
            <referenceNum></referenceNum>
            <ipAddress></ipAddress>
            <transactionDetail>
                <payType>
                    <giftCard>
                        <number></number>
                        <cvvNumber></cvvNumber>
                    </giftCard>
                </payType>
            </transactionDetail>
        </giftCardBalanceInquiry>
    </order>
</transaction-request>
```

Response XML structure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<transaction-response>
    <authCode />
    <orderID></orderID>
    <referenceNum></referenceNum>
    <transactionID></transactionID>
    <transactionTimestamp></transactionTimestamp>
    <responseCode></responseCode>
    <responseMessage></responseMessage>
    <avsResponseCode />
    <cvvResponseCode></cvvResponseCode>
    <processorCode></processorCode>
    <processorMessage> </processorMessage>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <errorMessage />
    <accountBalance></accountBalance>
</transaction-response>
```

Or

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

Deactivating a Gift Card

Gift cards are only currently supported on the Chase Paymentech platform. If you are using any other processor, please do not attempt to send any gift card requests.

To deactivate a gift card, use the `<giftCardDeactivate>` method. When you deactivate a gift card, the account balance on the card will be set to $0.00.

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

INPUT XML Structure:

```
<transaction-request>
    <verification>
        <merchantId></merchantId>
        <merchantKey> </merchantKey>
    </verification>
    <order>
        <giftCardDeactivate>
            <referenceNum></referenceNum>
            <ipAddress></ipAddress>
            <transactionDetail>
                <payType>
                    <giftCard>
                        <number></number>
                        <cvvNumber></cvvNumber>
                    </giftCard>
                </payType>
            </transactionDetail>
        </giftCardDeactivate>
    </order>
</transaction-request>
```

Response XML structure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<transaction-response>
    <authCode></authCode>
    <orderID></orderID>
    <referenceNum></referenceNum>
    <transactionID></transactionID>
    <transactionTimestamp></transactionTimestamp>
    <responseCode></responseCode>
    <responseMessage></responseMessage>
    <avsResponseCode />
    <cvvResponseCode></cvvResponseCode>
    <processorCode></processorCode>
    <processorMessage></processorMessage>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <partiallyApprovedAmount/>
    <accountBalance/>
    <errorMessage />
</transaction-response>
```

Or

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

Reactivating a Gift Card

Gift cards are only currently supported on the Chase Paymentech platform. If you are using any other processor, please do not attempt to send any gift card requests.

To reactivate a gift card, use the `<giftCardReactivate>` method.

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

INPUT XML Structure:

```
<transaction-request>
    <verification>
        <merchantId></merchantId>
        <merchantKey> </merchantKey>
    </verification>
    <order>
        <giftCardReactivate>
            <referenceNum></referenceNum>
            <ipAddress></ipAddress>
            <transactionDetail>
                <payType>
                    <giftCard>
                        <number></number>
                        <cvvNumber></cvvNumber>
                    </giftCard>
                </payType>
            </transactionDetail>
            <payment>
                <chargeTotal></chargeTotal>
            </payment>
        </giftCardReactivate>
    </order>
</transaction-request>
```

Response XML structure:

```
<?xml version="1.0" encoding="UTF-8" ?>
<transaction-response>
    <authCode></authCode>
    <orderID></orderID>
    <referenceNum></referenceNum>
    <transactionID></transactionID>
```

```
    <transactionTimestamp></transactionTimestamp>
    <responseCode></responseCode>
    <responseMessage></responseMessage>
    <avsResponseCode />
    <cvvResponseCode> </cvvResponseCode>
    <processorCode></processorCode>
    <processorMessage></processorMessage>
    <processorReferenceNumber/>
    <processorTransactionID/>
    <partiallyApprovedAmount/>
    <accountBalance/>
    <errorMessage />
</transaction-response>
```

Or

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

Performing a Gift Card Sale

Gift cards are only currently supported on the Chase Paymentech platform. If you are using any other processor, please do not attempt to send any gift card requests.

To run a sale transaction on a gift card, use the `<giftCardSale>` method.

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

INPUT XML Structure:

```
<transaction-request>
    <verification>
        <merchantId></merchantId>
        <merchantKey></merchantKey>
    </verification>
    <order>
        <giftCardSale>
            <referenceNum></referenceNum>
            <ipAddress></ipAddress>
            <transactionDetail>
                <payType>
                    <giftCard>
                        <number></number>
                        <cvvNumber></cvvNumber>
                    </giftCard>
                </payType>
            </transactionDetail>
            <payment>
                <chargeTotal></chargeTotal>
```

```
            </payment>
        </giftCardSale>
    </order>
</transaction-request>
```

Response XML structure:

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

Or

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

11. Appendix

Frequenty Asked Questions

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

A: You need to use the API Integration Test Posting URL

(https://apiint.paymentsite.com/UniversalAPI/postXML) to post transactions from your application.

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: 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"?><apierror>
<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 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?

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"?><transactionresponse><uthCode>123456</authCode><orderID>C0A86E6F:011CC
02A270B:86E8:01417690</orderID><referenceNum>846392232</referenceNum><transactionID>1700</trans
actionID><transactionTimestamp>1222994700</transactionTimestamp>**<responseCode>0</responseCod
e><responseMessage>CAPTURED</responseMessage>**<avsResponseCode>YYY</avsResponseCode>
<cvvResponseCode>M</cvvResponseCode><processorCode>A</processorCode><processorMessage>APPR
OVED</processorMessage><errorMessage/></transactionresponse>

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

Table: Transaction Response Code 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
4096	COMMUNICATION ERROR	There was a communication error in the payment system

Input Parameter Descriptions

Parameter	Max Size	Description	Input / Output
<command>	N/A	For non-transactional functions like adding a customer or modify a recurring schedule of payments, use the command function to indicate the action to be taken. Set to one of the following values: add-consumer, delete-consumer, updateconsumer, add-card-onfile, add-ach-onfile, delete-card-onfile, add-consumer-onfile, modifyrecurring, cancel-recurring, submit-transactioninquiry, get-transaction-inquiry-status	I
<ach><bankRoutingNumber>	9	The Routing and Transit (ABA) Number of the bank on which the draft is drawn – used for an echeck Transaction. Required for ACH.	I

<ach><achAccountNumber>	17	The consumer's bank account number – used for an e-check transaction. Required for ACH	I
<ach><achEffectiveDate>	19	The ach transaction is held and will not be submitted to the bank until this date. Field is optional. Format is: YYYY-MM-DD HH:MM:SS	I
<ach><checkNumber>	8	The check identification number – used for an echeck transaction. Not required for ACH, but if the tag is present it must be populated.	I
<ach><MICR>	N/A	String, no length restriction. Optional	I
<ach><frontImage>	N/A	base64Binary image file of the front of the check. MIME type is image/tiff	I
<ach><backImage>	N/A	base64Binary image file of the back of the check. MIME type is image/tiff	I
<ach><achPaymentType>	3	String used to indicate whether the ACH authorization was written (SEC codes: ARC, POP, PPD), verbal (TEL), or electronic (WEB) authorization. Can be set to any of the following values: 'PPD' 'WEB' 'TEL' 'CCD' 'CTX' 'POP' or 'ARC'. Required for ACH	I
<ach><achAccountType>	2	String used to identify the type of account used in the transaction. Can be set to any of the following: 'PC' for personal checking, 'PS' for personal savings, 'PL', 'BC' for business checking, 'BS' for business savings, or 'BL'. Required for ACH	I
<rdc><MICR>	N/A	String, no length restriction	I

<rdc><frontImage>	N/A	base64Binary image file of the front of the check. MIME type is image/tiff	I
<rdc><backImage>	N/A	base64Binary image file of the back of the check. MIME type is image/tiff	I
<billing><address>	128	Billing address for the payment type being used. The numeric portion of the address (the house number) is typically used for address verification on credit/debit card transactions	I
<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
<payment><chargeTotal>	N/A	Total amount of the transaction, including sales tax, shipping, etc. This is the amount that is charged (or returned) to the customer's card. In the case of a return, it can be equal to or less than the	I

		initial transaction amount (to allow for partial returns).	
<payment><salesTaxTotal>	N/A	The total amount of sales tax that applies to this transaction. If the tag is included in the XML, it must contain a numeric value. 0.00 is allowed	I
<payment><shippingTotal>	N/A	The total shipping charges that apply to this transaction. If the tag is included in the XML, it must contain a numeric value. 0.00 is allowed.	I
<payment><cashBack>	N/A	For PIN debit transactions, the amount of cash the customer wants back. If the tag is included in the XML structure, it must contain a numeric value. 0.00 is allowed.	I
<payment><convenienceFee>	N/A	Optional field for use if a convenience fee is charged. This field is only used for processors that support a convenience fee. If the processor does not support the use of a convenience fee field, this field (if included) will be ignored. Contact customer support if you have questions about whether the processor you are using supports convenience fees. If the tag is included in the XML structure, it must contain a numeric value. 0.00 is allowed.	I
<payment><tipTotal>	N/A	For restaurant transactions, a field to pass the amount of the tip. The tipTotal + authorized amount = captured amount.	I
<payment><currencyCode>	3	Optional field to specify the currency for processing the transaction. Default is USD (US dollars). Supported ONLY by Chase Salem; if passed for any other processor, the tag will be	I

		ignored. Must be set to one of the 3-digit current codes from ISO standard #4217. See the currency table at the end of this document for valid currency values. Chase Salem does not support retail, PIN debit transactions. Level 2, Level 3 and gift card transactions on Chase Salem are US only.	
<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. Required for credit card transactions if Heartland is the processor.	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
<deviceType>	N/A	For swiped card transactions where an encrypted device is used. Set to 'iDynamo', 'MagneSafeV5’, or 'Rambler', depending on which device the merchant is using for swiping cards. Also required when adding a card or account on file if the card or account data is encrypted.	I
<deviceKSN>	20	For swiped card transactions where an encrypted device is used. Device key serial number. Used for decrypting the card	I

		track data. Also required when adding a card or account on file if the card or account data is encrypted.	
<creditCard><track1Data encrypted="Y">	512	Credit Card track data. One of the two tracks (either track1 or track2) is required for cardpresent swiped transactions. If using an encrypted device to swipe the card, set encrypted = "Y". If not, there is no need to set the "encrypted" flag.	I
<creditCard><track2Data encrypted="Y">	512	Credit Card track data. One of the two tracks (either track1 or track2) is required for card present swiped transactions. If using an encrypted device to swipe the card, set encrypted = "Y". If not, there is no need to set the "encrypted" flag.	I
<creditCard><trackData>	512	Credit Card track data. You can send all track data in the trackData field vs. sending track1Data and track2Data separately.	I
<crebitCard><eCommInd>	10	e-commerce Indicator. This is a field required by payment processors for both credit and debit transactions to indicate whether the customer is present and the card was swiped through a reader, whether the customer provided payment info via mail or telephone, or if the information was received via the internet. Set to "retail" for face - to- face transactions, "moto" for mail or telephone orders, "eci" for internet or email orders, or "restaurant" for restaurant transactions. Sending a value of "restaurant" allows the captured transaction amount to be more than the authorization amount.	I

		Captured Amount = Authorized Amount + Tip Amount.	
<debitCard><track1Data>	512	For PIN debit transactions: Debit Credit Card track data. One of the two tracks (either track1 or track2) is required for card-present swiped transactions	I
<debitCard><track2Data>	512	For PIN debit transactions: Debit Card track data. One of the two tracks (either track1 or track2) is required for card-present swiped transactions.	I
<debitCard><trackData>	512	For PIN debit transactions: Debit Card track data. You can send all track data in the trackData field vs. sending track1Data and track2Data separately.	I
<debitCard><pinBlock>	32	For PIN debit transactions: The PIN data block	I
<debitCard><keySerialNo>	32	For PIN debit transactions: the key serial number data block	I
<debitCard><eCommInd>	10	e-commerce Indicator. This is a field required by payment processors for both credit and debit transactions to indicate whether the customer is present and the card was swiped through a reader, whether the customer provided payment info via mail or telephone, or if the information was received via the internet. For PIN debit transactions, the e-commerce indicator should be set to “retail”.	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	I

		present for one-time transactions and is not needed for recurring payments scheduled via the Paymentsite system.	
<firstRecurring/>	0	If tag is present, indicates this is the first recurring transaction in what will be a series. Do not pass a value with this tag, merely include <firstRecurring/>. Should only be present if recurringPaymentsFlag is also present.	I
<clientData><comments/>	19	A field to pass any comments related to this transaction	I
<clientData><customField1/>	100	A field to pass custom data related to this transaction	I
<clientData><customField2/>	100	A field to pass custom data related to this transaction	I
<clientData><customField3/>	100	A field to pass custom data related to this transaction	I
<clientData><customField4/>	100	A field to pass custom data related to this transaction	I
<clientData><customField5/>	100	A field to pass custom data related to this transaction	I
<consumerPrimaryId/>	64	Primary customer number. Merchant should populate this field with the customerID number returned from an add-consumer method.	I
<consumerExternalId/>	64	Alternate customer number assigned by the merchant	I
<ipAddress>	16	The IP Address of the consumer's computer system. The merchant's IP Address is automatically collected. Required for JHA transactions.	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
<processorID>	2	For merchants who are set up for multiple card or check processors, this field is used to identify which processor this transaction should be processed with. Used primarily in the South America implementation at this point in time. Can be numeric values from 1 – 18. Contact support for processor mapping values.	I
<onfile><customerId>	20	The Customer's unique identification number. Used to save a customer's information on file or when the customer's payment information is already on file.	I
<onfile><token>	100	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 ***new***, ***cancel*** or ***modify***.	I
<recurring><startDate>	12	The date to start the recurring transaction. Format is YYYY-MM-DD	I

<recurring><period>	10	The interval of time between transactions. Valid values are ***daily, week, month,*** and ***quarter***.	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 ***week*** and the <frequency> to ***2***.	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 must fail before the system begins sending recurring failure notifications	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
<invoiceNumber>	100	The invoice number associated with this transaction. Typically used for business-to business transactions. This field goes in the <order> tag and can be used for auth, sale, capture, recurringPayment, return, credit/debitSale, debitReturn, fsaSale, fsaReturn transactions for credit card, debit card, check/ACH, and onFile payment types	I
<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
<customerToken>	64	Used for identifying the customer when saving a card or bank account on file in conjunction with a payment transaction. Populate this field with the customerId that was returned when you added the customer using the add-consumer method.	
<onFileEndDate>	12	The last date (in mm/dd/yyyy format) that the merchant is authorized by the customer to use the card (or account) on file.	I
<onFilePermissions/>	10	The type of permission that the merchant has for a customer's card (or account) on file. May be set to 'ongoing' if the merchant has permission to use the card (or account) for some period of time or 'use_once' if the	I

		merchant may use the card on file one time only.	
<onFileComment/>	1024	Optional comments about the customer's card (or account) on file	I
<onFileMaxChargeAmount/>	10	The maximum amount (in US dollars) that the merchant is authorized to charge to the card (or account) on file at any one time. Must be a numerical value.	I
<customerIdExt>	64	Optional External Customer Id field, passed during creation of consumer.	I
<firstName>	64	Required First name field, passed during creation of consumer	I
<middleName>	64	Middle name field, passed during creation/modification of consumer	I
<lastName>	64	Last name field, passed during creation of consumer	I
<address1>	128	First line address of the consumer, passed during creation/modification of consumer details	I
<address2>	128	Second line address of the consumer, passed during creation/modification of consumer details	I
<city>	64	City of the consumer, passed during creation/modification of consumer details	I
<state>	2	2-digit state abbreviation for the Consumer's address. Used for adding/modifying a consumer.	I
<phone>	32	Consumer's primary phone number. Used for adding/modifying a consumer.	I

<email>	128	Consumer's e-mail address. Used for adding/modifying a consumer.	I
<dob>	12	Consumer's date of birth, in mm/dd/yyyy format. Used for adding/modifying a consumer.	I
<ssn>		Consumer's social security number. Used for adding/modifying a consumer.	I
<sex>	1	Consumer's gender. Can be set to M for male or F for female. Used for adding/modifying a consumer.	I
<approvedTxnEmail>	1	A value 'Y' would enable the consumer receiving emails for approved transactions. Used for adding/modifying a consumer.	I
<declinedTxnEmail>	1	A value 'Y' would enable the consumer receiving emails for declined transactions. Used for adding/modifying a consumer.	I
<recAdvNoticeEmail>	1	A value 'Y' would enable the consumer receiving advance emails for recurring transactions. Used for adding/modifying a consumer	I
<recAdvNoticeThreshold>	4	A value to indicate the threshold days before recurring expiry for email notification. Uses for adding/modifying a consumer.	I
<ccOnFileExpiryEmail>	1	A value 'Y' would enable the consumer receiving advance emails for card on file expiry. Used for adding/modifying a consumer	I
<ccOnFileExpiryThreshold>	4	A value to indicate the threshold days before card on file expiry for email notification. Uses for adding/modifying a consumer	I

<purchaseCardDetail> **<sdMerchantName>**	25	Used with purchasing card transactions. Field used to identify the merchant name to the cardholder. This field should be set to a name that is most recognizable to the cardholder [Company name or trade name].	I
<purchaseCardDetail> **<SDProductDescription>**	18	Used with purchasing card transactions to provide an accurate product description.	I
<purchaseCardDetail> **<SDMerchantCity>**	13	Used with purchasing card transactions to identify the city that a retail merchant is located in.	I
<purchaseCardDetail> **<AmexTransactionAdviceAddendum1>**	40	AmEx Purchasing Card Data – American Express Transaction Advice Addendum #1. This record provides additional purchase information for American Express purchasing card transactions. It is also used to provide specific details about the transaction to the cardholder for tracking purposes. Information entered in this field should be as specific as possible. MERCHANDISE, for example, is unacceptable. APPLE MACINTOSH is acceptable. The text must be in uppercase. Transaction Advice Addendum fields must be presented in sequence.	I
<purchaseCardDetail> **<AmexTransactionAdviceAddendum2>**	40	AmEx Purchasing Card Data – American Express Transaction Advice Addendum #2.	I
<purchaseCardDetail> **<AmexTransactionAdviceAddendum3>**	40	AmEx Purchasing Card Data – American Express Transaction Advice Addendum #3.	I
<purchaseCardDetail> **<AmexTransactionAdviceAddendum4>**	40	AmEx Purchasing Card Data – American Express Transaction Advice Addendum #4.	I
<PCLevel>	1	Required with purchasing card transactions to identify the	I

		purchase card level. Set to “2” for purchasing card level 2 or “3” for purchasing card level 3.	
<PCOrderNumber>	17	Required with purchasing card transactions to identify the purchase order number or the Order number from the customer.	I
<TaxIndicator>	1	Required with purchasing card transactions to define the tax type. 0 – Not provided 1 – Included 2 – Non-Taxable	I
<PC3AlternateTaxAmount>		Used with purchasing card level 3 transactions to define the total amount of alternate tax associated with this transaction.	I
<PC3AlternateTaxID>	15	Used with purchasing card level 3 transactions to define the tax ID number for the alternate tax associated with this transaction.	I
<PC3VatTaxRate>		Used with purchasing card level 3 transactions to define the VAT or other tax rate (expressed in percentage terms).	I
<PC3VatTaxAmount>		Used with purchasing card level 3 transactions to define the total amount of VAT or other tax included in this transaction	I
<PC3DiscountAmount>		Used with purchasing card level 3 transactions to define the total amount of discount applied to the transaction by the merchant	I
<PC3ShipFromZip>	10	Used with purchasing card level 3 transactions to identify the zip/postal code of the location	I

		from which the goods are shipped	
<PC3DestCountryCode>	3	Required with purchasing card level 3 transactions to identify the ISO-assigned 3-digit code of the country to which the goods are shipped.	I
<PC3DutyAmount>		Used with purchasing card level 3 transactions to identify the total charges for any import and/or export duties included in this transaction	I
<PC3FreightAmount>		Used with purchasing card level 3 transactions to identify the total freight or shipping and handling charges	I
<itemlist itemCount>	2	Required with purchasing card level 3 transactions to indicate the number of line items included. Example: <itemList itemCount="2"><item> …</item> <item> … </item></itemlist>	I
<item><itemIndex>	2	Required with purchasing card level 3 sale and post-authorization transactions to indicate the sequential number of the line item	I
<item><itemDescription>	35	Required with purchasing card level 3 transactions to provide a text description of the line item	I
<item><itemProductCode>	12	Required with purchasing card level 3 transactions to provide the product code of the item purchased	I
<item><itemQuantity>		Required with purchasing card level 3 transactions to provide the number of units or quantity of the item purchased	I
<item><itemUnitOfMeasurement>	3	Required with purchasing card level 3 transactions to provide	I

		the unit of measure code used for this line item. Cannot be left blank.	
<item><itemTaxAmount>		Required with purchasing card level 3 transactions to provide the tax amount for this item.	I
<item><itemTaxRate>		Required with purchasing card level 3 transactions to provide the tax rate applied to this item.	I
<item><itemDiscountAmount>		Required with purchasing card level 3 transactions to provide the amount of the discount applied to the line item. Must be set to 0 or a positive amount-cannot be left blank.	I
<item><itemCommodityCode>	3	Used with purchasing card level 3 transactions to provide the commodity code associated with this line item	I
<item><itemUnitCost>		Required with purchasing card level 3transactions to provide the unit cost of the line item.	I
<item><itemTaxIncluded>	1	Required with purchasing card level 3 transactions. Indicates whether the tax amount is included in the item amount Y = Item amount includes tax amount N = Item amount does not include tax amount	I
<item><itemTaxType>	4	Used with purchasing card level 3 transactions to provide the type of tax being applied for this item	I
<item><itemDebitIndicator>	1	Used with purchasing card level 3 transactions to indicate whether the item extended amount is a debit or credit. Cannot be left blank. Valid Values:	I

		D = Item extended amount is a debit. C = Item extended amount is a credit.	
inactivationFlag	1	Used with token-inactivation command. Cannot be left blank. Valid Values: Y = Make token Inactive. N = Make token active.	I
FraudScore	3	Numeric	

Output Parameters

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. Will always return no value for authForce and saleForce transactions.	O
<cvvResponseCode>	4	Response code from the processor indicating whether the card code information supplied matches the card code on file at the Card Issuer. Will always return no value for authForce and saleForce transactions.	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,	O

		something is wrong with the XML.	
<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
<gatewayDebitNetworkID>	6	Response from the processor and used by some Debit Merchants who use the value for settlement purpose	O
<processorCode>	64	A code from the processor indicating the status of the transaction. Some processors will populate this field; others will not.	O
<processorMessage>	255	A message from the processor indicating the status of the transaction. Some processors will populate this field; others will not.	O
<processorTransactionID>	128	If the payment processor sends back a transaction id, it will be provided in this tag. Merchants can use this identifier to help locate a transaction from a processor report.	O
<processorReferenceNumber>	64	If the payment processor sends back a transaction reference number, it will be provided in this tag.	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
<result><status>	32	The status of the batch request. Possible values that may be returned are: “Processed”, “Queued”, “Processing”, “Not Found”	O

<result><batchID>	32	The unique batch ID assigned to the batch by the system.	O
<result><noOfTransactions>	N/A	Numeric response indicating the number of transactions in the batch.	O
<result><TotalAmount>	N/A	Decimal value showing the total sum of all transactions included in the settled batch.	O
<transactionTimestamp>	13	Transaction creation date and time in GMT, in seconds since Jan 1, 1970.	O
<time>	13	Numeric	

EMV Specific Tags

Field Name	Size	Description	Input/ Output
<cardSequenceNumber>	3	Card sequence number, 000 - 999	I
<emvDataRequest>	999	EMV card chip request data in base64 encoded format.	I
<isFallBackToSwipe>	1	Flag for fall back to swipe transaction in case of chip read error, 0=no OR 1=yes.	I
<systemTraceNumber>	6	This number needs to be a unique number per Merchant and Terminal ID. If multiple terminals are used at a merchant's location, the system trace number must be different for each terminal.The system trace number must increment from 000001 to 999999 and not reset until it reaches 999999 .	I
<processorMid>	16	Merchant Id assigned by processor.	I

<processorTid>	16	Terminal Id assigned by processor.	I
<MessageAuthCode>	16	Message Authentication Code (MAC) generated by terminal.	I
<MessageAuthCheckDigits>	4	The check digits are validation characters associated with the message authentication working key .	I
<debitAccountType>	10	Debit account type, 'checking' or 'saving'	I
<fileDownloadName>	10	File name indicator, 'EMV2KEY'	I/O
<fileBlockSequence>	4	Use this field to keep track the current downloading block, 0001 - 9999.	I/O
<fileBlockMaxSize>	3	Maximum block size, 100 – 500 .	I
<fileOffset>	7	Indicate the file offset, want to download from. First download request should have 0s as offset, subsequent download request should use provided —next file download offset provided in the response message.	I
<emvDataResponse>	999	EMV response data received from processor in base64 encoded format.	O
<fileUpdateName>	10	File update name indicator if received from host, 'EMV2KEY'	O
<fileUpdateCreateDate>	14	File update create date received from host in format of Mmddyyyyhhmmss (zeroes if not available)	O
<fileUpdateSize>	5	Total size of file in bytes	O

<fileUpdateCRC>	4	The check digits are validation characters associated with the message authentication working key .	O
<fileUpdateFCode>	1	File update function code, 'S' = File transfer successful 'N' = No update available 'U' = Update available	O
<gatewayMessageAuthCode>	16	Message Authentication Code (MAC) received from host.	O
<gatewayMessageAuthCheckDigits>	4	The check digits received from host are validation characters and associated with the message authentication working key .	O
<TMACKey>	16	Message Authentication Working Key, received from host.	O
<TKPEKey>	16	PIN Encryption Working Key, received from host.	O
<TKMEKey>	16	Message Encryption Working Key, received from host.	O
<fileBlockSize>	3	The ACTUAL size of the following file data block, may differ from the MAX block size send in request message, 100 – 500 .	O
<fileNextOffset>	7	Provide the next file offset that should be downloaded from in subsequent download request.	O
<fileDownloadStatusCode>	1	File download status code from host 'C' – Continuation Block 'L' – Last block 'E' – Error	O

<fileBlockData>	500	File block data, length may vary; total size is the —Provided File Block Size.	O

School Specific Input Tags

Field Name	Size	Description	Input/ Output
<parentId>	64	Field used in the Schools Implementation ONLY. Parent's login ID	I
<parentAccountNumber>	64	Field used in the Schools Implementation ONLY. Parent account number. Should be unique for each parent.	I
<clientData><program>	100	Field used in the Schools Implementation ONLY. School program that this transaction relates to.	I
<clientData><category>	100	Field used in the Schools Implementation ONLY. Category of payment. A category is related to a program. Each program can have multiple categories.	I
<clientData><subCategory>	100	Field used in the Schools Implementation ONLY. Sub-category of payment. A sub-category is related to a category. Each category can have multiple sub-categories.	I
<clientData><parentName>	100	Field used in the Schools Implementation ONLY. Parent's name. You may include first, middle, and/or last name in this field.	I
<clientData><studentAccountNumber>	100	Field used in the Schools Implementation ONLY. Student ID / account	I

		number. This field must be unique for each student.	
<clientData><studentName>	100	Field used in the Schools Implementation ONLY. Student name. You may include first, middle, and/or last name in this field.	I
<clientData><studentId>	100	Field used in the Schools Implementation ONLY. Alternate student ID number. Some schools have two tracking numbers for a student—this field allows the school to send both fields.	I

Property Management Specific Tags

Field Name	Size	Description	Input / Output
<tenantId>	64	Field used in the Property Management Implementation ONLY. Tenant login ID. Populate this field with the customerId that was returned when you added the tenant as a customer using the add-consumer method.	I
<tenantAccountNumber>	64	Field used in the Property Management Implementation ONLY. Tenant account number. Should be unique for each tenant.	I
<clientData><property>	100	Field used in the Property Management Implementation ONLY. Name of the property associated with this tenant.	I
<clientData><department>	100	Field used in the Property Management Implementation ONLY. Department to associate with this transaction.	I
<clientData><customerPaying>	100	Field used in the Property Management Implementation ONLY. Method by which the customer is paying. Enumerated data field must be set to one of the following values: *onsite*, *lockbox*, *mail*, *phone*, *billpay*, *other.*	I

		Transaction fees are charged differently by the card associations depending on the method used to pay. *onsite* and *lockbox* are coded as retail transactions. *mail* and *phone* are coded as MOTO transactions. *billpay* and *other* are coded as eci transactions.	
<clientData><tenantApartmentNumber>	100	Field used in the Property Management Implementation ONLY. Tenant apartment or rental unit number.	I
<clientData><tenantName>	100	Field used in the Property Management Implementation ONLY. Tenant's name.	I
<clientData><tenantStatus>	100	Field used in the Property Management Implementation ONLY. Tenant status. Enumerated data field which must be set to one of the following values: *current*, *future*, *vacant*, *past*, *eviction*, *notice*, *notaccept*	I
<clientData><leaseRent>	100	Field used in the Property Management Implementation ONLY. Amount of lease or rent due from the tenant.	I
<clientData><balanceForward>	100	Field used in the Property Management Implementation ONLY. Balance due from this tenant for previous billing cycles.	I
<clientData><parkingGarage>	100	Field used in the Property Management Implementation ONLY. Parking garage fees for this tenant.	I
<clientData><storage>	100	Field used in the Property Management Implementation ONLY. Storage fees for this tenant.	I
<clientData><fitnessCenter>	100	Field used in the Property Management Implementation ONLY. Fitness center fees for this tenant.	I
<clientData><delinquincyLateFee>	100	Field used in the Property Management Implementation ONLY. Delinquency or late fees owed by this tenant.	I

<clientData><utilityServices>	100	Field used in the Property Management Implementation ONLY. Amount due from this tenant for utilities.	I
<clientData><deposit>	100	Field used in the Property Management Implementation ONLY. Deposit due from this tenant.	I
<clientData><applicationProcessing>	100	Field used in the Property Management Implementation ONLY. Application processing fee due from this tenant.	I
<clientData><petFee>	100	Field used in the Property Management Implementation ONLY. Pet fees due from this tenant.	I
<clientData><nsfFee>	100	Field used in the Property Management Implementation ONLY. NSF fees due from this tenant.	I
<clientData><specialFeeOther>	100	Field used in the Property Management Implementation ONLY. Special fees/other fees due from this tenant.	I
<clientData><taxes>	100	Field used in the Property Management Implementation ONLY. Taxes due from this tenant.	I
<clientData><customField1>	100	Field used in the Property Management Implementation ONLY. Field for sending a custom amount (that doesn't fit in any of the above categories) due from the tenant.	I
<clientData><customField2>	100	Field used in the Property Management Implementation ONLY. Field for sending a custom amount (that doesn't fit in any of the above categories) due from the tenant.	I
<clientData><customField3>	100	Field used in the Property Management Implementation ONLY. Field for sending a custom amount (that doesn't fit in any of the above categories) due from the tenant.	I
<clientData><customField4>	100	Field used in the Property Management Implementation ONLY. Field for sending a custom amount (that doesn't fit in	I

		any of the above categories) due from the tenant.	
<clientData><customField5>	100	Field used in the Property Management Implementation ONLY. Field for sending a custom amount (that doesn't fit in any of the above categories) due from the tenant.	I

12. Possible Response Values

AVS Response Code Result Values

The following table lists the possible results for AVS checking and their meaning. This data is returned in the <avsResponseCode> tag.

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

CVV Response Code Result Values

The following table lists the possible results for CVV checking and their meaning. This data is returned in the <cvvResponseCode> tag.

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

Gateway Response Code Result Values

The following table lists the possible results that will be returned on a request and their meaning. This data is returned in the <responseCode> tag.

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 Paymentsite

4096	COMMUNICATION ERROR	There was a communication error in the Paymentsite payment system

13. Additional Information

Functions not Universally Supported

Some functions are not supported by all processors. Listed in the table below are the functions that are not universally supported.

Parameter/function	Description
<convenienceFee>	Used for passing the amount of the convenience fee to the processor. Not currently supported by WorldPay or TSYS.
Purchasing card transactions	Purchasing card transactions level 2 is supported for First Data Nashville, Chase Paymentech Orbital, Chase Paymentech Tampa and Global. Purchase Card Level 3 is currently supported by Chase Paymentech Orbital ONLY.
<currencyCode>	Used to specify the currency for the transaction. Default is USD (United States dollars). Supported by Chase Paymentech Orbital ONLY.

Currency Codes

If processing transactions in currency other than US dollars, the <currencyCode> tag is used to specify which currency to use for the transaction. The code passed must be one of the 3-digit codes from ISO 4217, as specified in the table below. The currencies supported are indicated with a Y in the "supported by Chase" column. Chase Salem is the only processor who currently supports multi-currency.

Code	Currency	Supported by Chase?
AFN	Afghan afghani	
ALL	Albanian lek	

DZD	Algerian dinar	Y
AOA	Angolan kwanza	
ARS	Argentine peso	Y
AMD	Armenian dram	Y
AWG	Aruban guilder	Y
AUD	Australian dollar	Y
AZN	Azerbaijani manat	Y
BSD	Bahamian dollar	Y
BHD	Bahraini dinar	
BDT	Bangladeshi taka	Y
BBD	Barbados dollar	Y
BYR	Belarusian ruble	Y
BZD	Belize dollar	Y
BMD	Bermudian dollar (customarily known as Bermuda dollar)	Y
BTN	Bhutanese ngultrum	
BOV	Bolivian Mvdol (funds code)	
BOB	Boliviano	Y
BAM	Bosnia and Herzegovina convertible mark	
BWP	Botswana pula	Y
BR L	Brazilian real	Y
GBP	British Pound	Y
BND	Brunei dollar	Y
BGN	Bulgarian lev	Y

BIF	Burundian franc	Y
KHR	Cambodian riel	Y
CAD	Canadian dollar	Y
CVE	Cape Verde escudo	Y
KYD	Cayman Islands dollar	Y
XOF	CFA Franc BCEAO	Y
XAF	CFA franc BEAC	Y
XPF	CFP franc	Y
CLP	Chilean peso	Y
CNY	Chinese yuan	Y
XTS	Code reserved for testing purposes	Y
COP	Colombian peso	Y
KMF	Comoro franc	Y
CDF	Congolese franc	
CRC	Costa Rican colon	Y
HRK	Croatian kuna	
CUC	Cuban convertible peso	
CUP	Cuban peso	
CZK	Czech koruna	Y
DKK	Danish krone	Y
DJF	Djiboutian franc	Y
DOP	Dominican peso	Y
XCD	East Caribbean dollar	Y

EGP	Egyptian pound	
SVC	El Salvador Colon	Y
ERN	Eritrean nakfa	
EEK	Estonian Kroon	Y
ETB	Ethiopian birr	Y
EUR	Euro	Y
XBA	European Composite Unit(EURCO) (bond market unit)	
XBB	European Monetary Unit(E.M.U.-6) (bond market unit)	
XBD	European Unit of Account 17(E.U.A.-17) (bond market unit)	
XBC	European Unit of Account 9(E.U.A.-9) (bond market unit)	
FKP	Falkland Islands pound	Y
FJD	Fiji dollar	Y
GMD	Gambian dalasi	Y
GEL	Georgian lari	Y
GHC	Ghanaian Cedi	Y
GHS	Ghanaian cedi	
GIP	Gibraltar pound	Y
XAU	Gold (one troy ounce)	
GTQ	Guatemalan quetzal	Y
GWP	Guinea-Bissau Peso	Y
GNF	Guinean franc	Y
GYD	Guyanese dollar	Y
HTG	Haitian gourde	Y

HNL	Honduran lempira	Y
HKD	Hong Kong dollar	Y
HUF	Hungarian forint	Y
ISK	Icelandic króna	Y
INR	Indian rupee	Y
IDR	Indonesian rupiah	Y
IRR	Iranian rial	
IQD	Iraqi dinar	
ILS	Israeli new sheqel	Y
JMD	Jamaican dollar	Y
JPY	Japanese yen	Y
JOD	Jordanian dinar	
KZT	Kazakhstani tenge	Y
KES	Kenyan shilling	Y
KWD	Kuwaiti dinar	
KGS	Kyrgyzstani som	Y
LAK	Lao kip	Y
LVL	Latvian lats	Y
LBP	Lebanese pound	Y
LSL	Lesotho loti	
LRD	Liberian dollar	
LYD	Libyan dinar	
LTL	Lithuanian litas	Y

MOP	Macanese pataca	Y
MKD	Macedonian denar	
MGA	Malagasy ariary	
MGF	Malagasy Franc	Y
MWK	Malawian kwacha	Y
MYR	Malaysian ringgit	Y
MVR	Maldivian rufiyaa	Y
MRO	Mauritanian ouguiya	Y
MUR	Mauritian rupee	Y
MXN	Mexican peso	Y
MXV	Mexican Unidad de Inversion(UDI) (funds code)	
MDL	Moldovan leu	Y
MNT	Mongolian tugrik	Y
MAD	Moroccan dirham	Y
MZM	Mozambique Metical	Y
MMK	Myanma kyat	
NAD	Namibian dollar	Y
NPR	Nepalese rupee	Y
ANG	Netherlands Antillean guilder	Y
TWD	New Taiwan dollar	Y
NZD	New Zealand dollar	Y
NIO	Nicaraguan córdoba	Y
NGN	Nigerian naira	Y

XXX	No currency	
KPW	North Korean won	
NOK	Norwegian krone	Y
OMR	Omani rial	
PKR	Pakistani rupee	Y
XPD	Palladium (one troy ounce)	
PAB	Panamanian balboa	Y
PGK	Papua New Guinean kina	Y
PYG	Paraguayan guaraní	Y
PEN	Peruvian nuevo sol	Y
PHP	Philippine peso	Y
XPT	Platinum (one troy ounce)	
PLN	Polish złoty	Y
QAR	Qatari riyal	Y
ROL	Romanian new leu	Y
RUB	Russian rouble	Y
RWF	Rwandan franc	Y
SHP	Saint Helena pound	Y
WST	Samoan tala	Y
STD	São Tomé and Príncipe dobra	Y
SAR	Saudi riyal	Y
RSD	Serbian dinar	
SCR	Seychelles rupee	Y

SLL	Sierra Leonean leone	Y
XAG	Silver (one troy ounce)	
SGD	Singapore dollar	Y
SBD	Solomon Islands dollar	Y
SOS	Somali shilling	Y
ZAR	South African rand	Y
KRW	South Korean won	Y
SSP	South Sudanese pound	
XDR	Special drawing rights	
LKR	Sri Lankan rupee	Y
SDG	Sudanese pound	
SRD	Surinamese dollar	
SZL	Swazi lilangeni	Y
SEK	Swedish krona/kronor	Y
CHF	Swiss franc	Y
SYP	Syrian pound	
TJS	Tajikistani somoni	
TZS	Tanzanian shilling	Y
THB	Thai baht	Y
TOP	Tongan pa‘anga	Y
TTD	Trinidad and Tobago dollar	Y
TND	Tunisian dinar	
TRY	Turkish lira	Y

TMT	Turkmenistani manat	
UGX	Ugandan shilling	Y
XFU	UIC franc (special settlement currency)	
UAH	Ukrainian hryvnia	Y
CLF	Unidad de Fomento (funds code)	
COU	Unidad de Valor Real	
AED	United Arab Emirates dirham	Y
USD	United States dollar	Y
UYI	Uruguay Peso en Unidades Indexadas (URUIURUI) (funds code)	
UYU	Uruguayan peso	Y
UZS	Uzbekistan som	Y
VUV	Vanuatu vatu	Y
VEB	Venezuelan Bolivar	Y
VEF	Venezuelan bolívar fuerte	
VND	Vietnamese đồng	Y
CHE	WIR Euro (complementary currency)	
CHW	WIR Franc (complementary currency)	
YER	Yemeni rial	Y
ZMK	Zambian kwacha	Y
ZWL	Zimbabwe dollar	Y

Units of Measurement

When performing a purchasing card level 3 post-authorization or sale transaction, one of the required fields is <item><itemUnitOfMeasurement>, which must be set to a 3-digit code that indicates the units of measurement for the item. The table below contains the 3-digit codes and their meanings.

3-digit code	Description
ACR	Acre
ASM	Alcoholic strength by mass
ASV	Alcoholic strength by volume
AMP	AMP
AMH	Ampere-hour (3,6 kC)
ARE	Are (100 m2)
BAR	Bar
BLL	Barrel (petroleum) (158,987 dm3)
BFT	Board foot
BQL	Becquerel
BIL	Billion EUR
MLD	Billion US
BHP	Brake horse power (245,7 watts)
BTU	British thermal unit (1,055 kilojoules)
BUA	Bushel (35,2391 dm3)
BUI	Bushel (36,36874 dm3)
CDL	Candela
CCT	Carrying capacity in metric tones
CNT	Cental GB (45,359237 kg)
CGM	Centigram
CLT	Centiliter
CMT	Centimeter
DTN	Centner, metric (100 kg)
WCD	Cord (3,63 m3)
COU	Coulomb

CKG	Coulomb per kilogram
CMQ	Cubic centimeter
DMQ	Cubic decimeter
FTQ	Cubit foot
INQ	Cubic inch
MTQ	Cubic meter
MQH	Cubic meter per hour
MQS	Cubic meter per second
MMQ	Cubic millimeter
YDQ	Cubic yard
CUR	Curie
DAY	Day
DAA	Decare
DLT	Deciliter
DMT	Decimeter
DTN	Deciton
CEL	Degree Celsius
FAH	Degree Fahrenheit
DPT	Displacement tonnage
DZN	Dozen
DZP	Dozen packs
DZR	Dozen pairs
DCP	Dozen pieces
DRL	Dozen rolls
DRM	Drachm GB (3,887935 g)
DRI	Dram GB (1,771745 g)
DRA	Dram US (3,887935 g)

BLD	Dry barrel (115,627 dm3)
GLD	Dry gallon (4,404884 dm3)
PTD	Dry pint (0,55061 dm3)
QTD	Dry quart (1,101221 dm3)
FAR	Farad
OZI	Fluid ounce (28,413 cm3)
OZA	Fluid ounce (29,5735 cm3)
FOT	Foot (0,3048 m)
GLI	Gallon (4,546092 dm3)
GBQ	Gigabecquerel
GWH	Gigawatt-hour (1 million kW/h)
GII	Gill (0,142065 dm3)
GIA	Gill (11,8294 cm3)
GRN	Grain GB, US (64,798910 mg)
GRM	Gram
GFI	Gram of fissile isotopes
GGR	Great gross (12 gross)
GRO	Gross
GRT	Gross (register) ton
SAN	Half year (six months)
HAR	Hectare
HBA	Hectobar
HGM	Hectogram
DTH	Hectokilogram
HLT	Hectoliter
HPA	Hectoliter of pure alcohol
HMT	Hectometer

HTZ	Hertz
HUR	Hour
CEN	Hundred
BHX	Hundred boxes
HIU	Hundred international units
CLF	Hundred leaves
CNP	Hundred packs
CWA	Hundredweight US (45,3592 kg)
INH	Inch (25,4 mm)
JOU	Joule
KEL	Kelvin
KBA	Kilobar
KGM	Kilogram
KPH	Kilogram of caustic potash
KSH	Kilogram of caustic soda
KNS	Kilogram of named substance
KNI	Kilogram of nitrogen
KPP	Kilogram of phosphonic anhydride
KPP	Kilogram of phosphorus pentoxide
KPH	Kilogram of potassium hydroxide
KPO	Kilogram of potassium oxide
KSH	Kilogram of sodium hydroxide
KSD	Kilogram of substance 90% dry
KUR	Kilogram of uranium
KMQ	Kilogram per cubic meter

KGS	Kilogram per second
KHZ	Kilohertz
KJO	Kilojoule
KMT	Kilometer
KMH	Kilometer per hour
KPA	Kilopascal
KTN	Kilotonne
KVR	Kilovar
KVT	Kilovolt
KVA	Kilovolt-ampere
KWT	Kilowatt
KWH	Kilowatt-hour
KNT	Knot (1 nautical mile per hour)
LEF	Leaf
GLL	Liquid gallon (3,78541 dm3)
PTL	Liquid pint (0,473176 dm3)
QTL	Liquid quart (0,946353 dm3)
LTR	Liter (1dm3)
LPA	Liter of pure alcohol
CWI	(Long) hundredweight GB (50,802345 kg)
LTN	Long ton GB, US (1,0160469 t)
LUM	Lumen
LUX	Lux
MHZ	Megahertz
MAL	Megaliter
MAM	Megameter

MPA	Megapascal
MVA	Megavolt-ampere (1000 KVA)
MAW	Megawatt
MWH	Megawatt-hour (100 kW/h)
MTR	Meter
MTS	Meter per second
MSK	Meter per second squared
CTM	Metric carat (200 mg = 2.10-4 kg)
TNE	Metric ton (1000 kg)
MLD	Milliard
MBR	Millibar
MCU	Millicurie
MGM	Milligram
MLT	Milliliter
MMT	Millimeter
MIO	Million
HMQ	Million cubic meters
MIU	Million international units
MIN	Minute
MON	Month
NMI	Nautical mile (1852 m)
NTT	Net (register) ton
NEW	Newton
NMB	Number
NAR	Number of articles
NBB	Number of bobbins

NCL	Number of cells
NIU	Number of international units
NMP	Number of packs
NMR	Number of pairs
NPL	Number of parcels
NPT	Number of parts
NRL	Number of rolls
OHM	Ohm
ONZ	Ounce GB, US (28,349523 g)
APZ	Ounce GB, US (31,10348 g)
PAL	Pascal
DWT	Pennyweight GB, US (1,555174 g)
PCE	Piece
PTI	Pint (0,568262 dm3)
LBR	Pound GB, US (0,45359237 kg)
PGL	Proof gallon
QTI	Quart
QAN	Quarter (of a year)
QTR	Quarter, GB (12,700586 kg)
DTN	Quintal, metric (100 kg)
RPM	Revolution per minute
RPS	Revolution per second
SCO	Score
SCR	Scruple GB, US (1,295982 g)
SEC	Second
SET	Set

SHT	Shipping ton
SST	Short standard
STN	Short ton GB, US (0,90718474t)
SIE	Siemans
CMK	Square centimeter
DMK	Square decimeter
FTK	Square foot
INK	Square inch
KMK	Square kilometer
MTK	Square meter
MIK	Square mile
MMK	Square millimeter
TDK	Square yard
WSD	Standard
ATM	Standard atmosphere (101325 Pa)
SMI	(Statute) mile (1609,344 m)
STI	Stone GB (6,350293 kg)
ATT	Technical atmosphere (98066,5 Pa)
DAD	Ten days
TPR	Ten pairs
MIL	Thousand
TAH	Thousand ampere-hour
MBF	Thousand board feet (2,36 m3)
TQD	Thousand cubic meters per day
MBE	Thousand standard brick equivalent

TSH	Ton of steam per hour
TNE	Tonne (1000 kg)
TSD	Tonne of substance 90% dry
TRL	Trillion EUR
BIL	Trillion US
APZ	Troy Ounce
LBT	Troy pound, US (373,242 g)
VLT	Volt
WTT	Watt
WHR	Watt-hour
WEB	Weber
WEE	Week
YRD	Yard
ANN	Year

14. Version History

Version	Date	Summary of Changes
3.5.2.9	Aug 9, 2018	Added JSON API information for non-transactions; corrected information on non-transaction XML required fields
3.5.2.8	Nov 10, 2017	Added convenience fee references,
3.5.2.7	Jun 8, 2017	Reformatted document, removed EBT transaction info