Monetra®

Application Interface Guide

Revision: 9.0.0
Publication date July 29, 2022

Application Interface Guide

Monetra Technologies, LLC

Revision: 9.0.0

Publication date July 29, 2022

Legal Notice

Table of Contents

1 Revision History

Version	Date	Changes
9.0.0	2022-05-23	• Removed legacy information • Reference ReST API for action documentation instead of duplicating here
8.17.0	2021-06-29	• Corrected `amount` description and format for recurring payments • Updated `nsf` request parameter to indicate that partial authorizations are automatically reversed when merchant does not explicitly allow them • Added Monetra Code `NSFAUTODENY` • Added customer flag `email_receipt_recurring` • Added BIN range format appendix section • Updated general formatting
v8.17.0	2021-04-20	• Changed revision history ordering to most recent first • Detailed fields received in datablocks for all actions in ???? • Specified that `merch_custom_fields`, `merch_customer_fields`, and `custom_customer_fields_spec` field names are lowercase • Updated push notification information in ????
v8.16.0	2021-04-15	• Updated formatting • Detailed fields received in datablocks for all actions in "Main Merchant Actions" and "Merchant Subuser Actions" • Added `descloc` and `cavvresp` response fields to ????, ????, and ???? reports • Added note on how to properly parse datablocks
v8.13.0	2019-12-06	• Complete rewrite
v8.0.0	2017-06-29	• Initial document re-write. Now includes information from legacy addendums: IP/SSL/Drop File, XML spec, DSS Storage and billing

2 Overview

2.1 Introduction

Monetra is a fast, efficient, and secure payment application that is certified [https://www.monetra.com/certifications] to connect many types of applications directly to any of the major North American-based payment processors. It's designed to scale from small, custom, embedded devices to fully-redundant payment servers processing thousands of transactions per minute. Trusted for over 15 years by thousands of merchants throughout North America, Monetra is the premier product of its type.

Monetra supports critical payment features such as:

- Extensive EMV Processor certifications across the US and Canada
- Robust Tokenized Card Storage and Recurring Billing
- P2PE Card Encryption proven to comply with PCI P2PE standards, including HSM support
- Flexible, developer-friendly integration options, including ReSTful APIs and iFrame integrations
- Clustering support for redundancy and load balancing

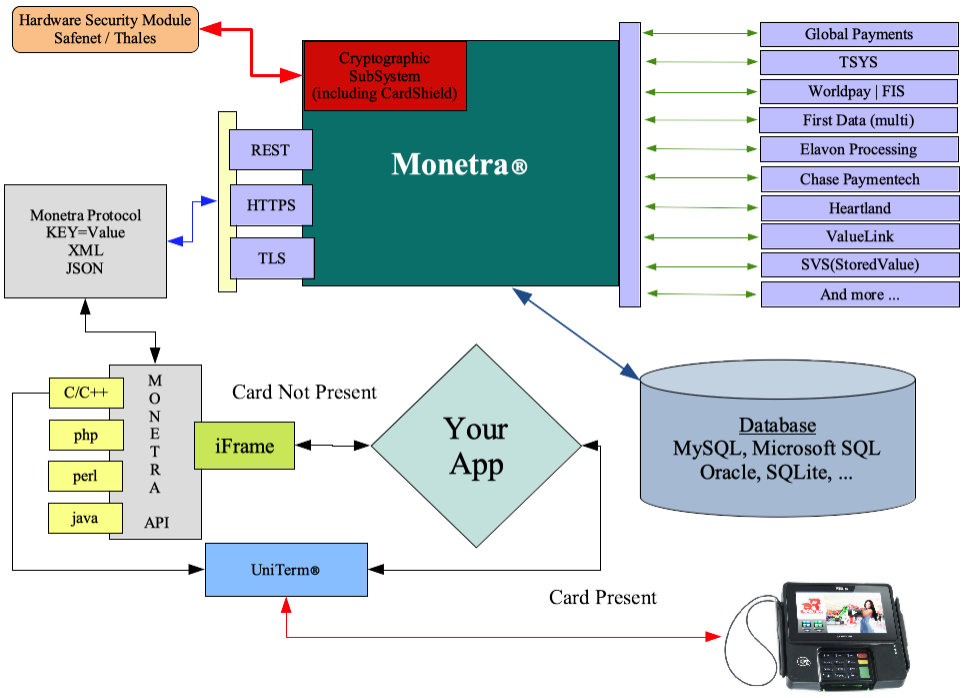
Designed from the ground up as a true Client/Server application, Monetra is written in 100% ANSI C89 to be compatible across all major and embedded operating systems, making it one of the most efficient, scalable, and portable payment engines available. It is built on a small payment core that houses all security and routing details, with all other features implemented via our flexible modular subsystem, which acts as an abstraction layer so that a change to the core does not impact a module and vice-versa. Inbound communications protocols, databases, processing institutions, etc. are all designed as separate modules to a common interface.

By providing a mature and feature-rich protocol, Monetra enables developers to build advanced payment-acceptance support directly into their application, including desktop, web, and mobile applications. Integrations can connect to Monetra through the ReST API. Other API method are available but they are considered legacy. These will continue to be supported so existing integrations do not need to update. However, all action and parameter documentation is contained in the ReST API documentation.

Monetra has been under constant development for over eighteen years while incorporating the functionality of our clients' best input directly into the production product along the way. We work hard to add more and more features and functionality to the core payment server every day. If there are any features you would like to see included, please feel free to contact sales@monetra.com.

2.2 Architecture

Figure 2.1.



3 Monetra Integration

3.1 Definitions

Identifier - A unique string of numbers and/or letters assigned to the transaction. This identifier may be repeated once a response has been given for the original transaction using the identifier.

Message - This is the actual data of a transaction that Monetra interprets and processes. Any data outside of this is strictly for transport/communication.

Start of Transaction - Indicator representing the start of a transaction, commonly known as `STX`. Hex value 0x02, decimal value: 2.

End of Transaction - Indicator representing the end of a transaction, commonly known as `ETX`. Hex value 0x03, decimal value: 3.

Field Separator - Character to separate one major portion of a message from another, commonly known as `FS`. Hex value 0x1C, decimal value: 28.

3.2 Communication and Transport

Monetra boasts a true Server/Client architecture, which allows for all system functionality to transpire either on the local machine or remotely across many dispersed systems. The connection methods that Monetra uses are completely modular by design, allowing for custom integration modules to be created and added easily. Modules are loaded at runtime. Currently, integrations are recommended to Monetra ReST with the ReST API. Legacy protocols include XML or JSON over HTTPS, and LibMonetra using TLS/SSL.

The ReST API documentation includes all key/value pairs that are used by all integration methods. The documentation for each action contains a section "libmonetra `KVS` equivalent for endpoint" which detail the key/value pairs necessary to preform the action using other integration methods. The key/value parameter pairs are the same and the same format for all methods.

3.2.1 TLS/SSL

A standard TLS/SSL connection should be made to the Monetra server. Transactions may be sent immediately upon successful connection.

For security-related segmentation purposes, each user group sends transactions to Monetra via a different port. These are the default port numbers for communication with Monetra:

Port	Type
8666	`GROUPMANAGE, PROFILEMANAGE, USERMANAGE, SYSADMIN, MERCHANT`
8665	`MERCHANT, USERMANAGE`

The basic LibMonetra transaction structure is as follows:
`[STX]`identifier**`[FS]`**message**`[ETX]`**

The identifier should be unique to the session, but it may be reused once a response is received from a transaction that shares the same identifier. The identifier may be any unique string of numbers and letters and is echoed in the response Monetra provides.

Both requests and responses share the same formatting requirements. The message portion of the data stream is formatted as detailed below.

If a connection to Monetra is lost while there are pending transactions for your connection, you must issue a Get Unsettled Transactions or Get Failed Transactions report in order to determine the status of your transaction. There is no recovery process to resume a failed session; therefore, a stable connection is strongly recommended.

We suggest using pre-existing free libraries to perform encryption and decryption with TLS/SSL. The most widely used libraries are available at http://www.openssl.org/. These libraries help perform all encryption "behind the scenes".

It is recommended to us the LibMonetra library with implements this format. This integration method has the highest performance. However, this should not be a factor in deciding between integration methods.

3.2.2 HTTPS

PCI Notice: Over public networks (like the Internet) or untrusted private networks, PCI Security requires the use of HTTPS instead of HTTP for communication to ensure card numbers and other critical data are not transmitted in plain text.

HTTP is the method of data transfer that web servers use. When you visit a website, that content is delivered to you using the HTTP protocol. HTTPS is simply HTTP tunneled over a TLS/SSL connection. The default port numbers for communication are `8665` for Merchant User actions and `8666` for Admin User actions.

With the goal of keeping it simple, Monetra only supports a small subset of the HTTP protocol. An HTTPS request to Monetra should be formatted as a simple HTTP 1.0 or 1.1 POST command. A Content-Length descriptor is also required to allow the Monetra parser to efficiently identify the start and end of the data being sent. Please verify that the host and port numbers reference your configuration (IP address of Monetra service, and configured ports for HTTPS).

Most programming languages provide HTTP routines to simplify the programming process. You may also wish to look into `curl` as an alternative (http://curl.haxx.se).

HTTP 1.0 is defined in RFC 1945 (http://www.faqs.org/rfcs/rfc1945.html), and HTTP 1.1 is defined in RFC 2616 (http://www.faqs.org/rfcs/rfc2616.html)

3.3 Transaction Structures

The basic protocol used to communicate with Monetra consists of key/value pairs. Beyond that, Monetra can accept transactions formatted as XML or JSON, as outlined below, or sent using the ReST API [https://developers.monetra.com/].

3.3.1 XML Transactions

Each set of XML transactions must begin and end with `<MonetraTrans>` tags. Within these tags is one or more transactions. Each transaction is wrapped in `<Trans>` tags, which must also specify an `identifier` attribute of letters and/or numbers for uniquely identifying that transaction. The sub tags for each transaction are the key/value pairs found in this specification. The order of the key/value data pairs is irrelevant, and Monetra does not return the response key/value pairs in any particular order.

XML responses are encapsulated in `<MonetraResp>` tags, and a corresponding `<Resp>` tag exists for every `<Trans>` tag sent (assuming the structure was initially correct). The `<Resp>` tag has an `identifier` attribute that echoes the identifier sent with each `<Trans>` tag. An additional `<DataTransferStatus>` tag indicating whether or not the request was successfully parsed is always sent with a `code` attribute, which has a value of `SUCCESS` or `FAIL`. On `FAIL`, an error message is enclosed in the `<DataTransferStatus>` tag, as seen in Section 3.3.1.3: XML Error Example.

Every `<Resp>` tag has a `<Code>` tag indicating the overall outcome of the transaction. For comma-separated responses (like you receive when running reports), there is a `<DataBlock>` tag that holds the response data. Otherwise, the response tags are those as defined in this guide.

Note: Some transactional characters might need to be encoded when inserting them into the XML data stream, such as these: `'>'` needs to become `'&gt;'`, `'<'` needs to become `'&lt;'` and `'&'` needs to become `'&amp;'`.

Note: The fields in a `<DataBlock>` tag are guaranteed to not be removed or renamed, but fields can be added or rearranged at any time. Integrations need to parse the header first to find the correct column index for the desired field before reading out the value in that field and should never blindly assume that the data will always be at a specific column index.

3.3.1.1 XML Request Example

Below is an example of an XML request.

```
1
2       <MonetraTrans>
3        <Trans identifier='1'>
4         <username>vitale</username>
5         <password>test</password>
6         <action_trans>sale</action_trans>
7         <account>4012888888881</account>
8         <expdate>0512</expdate>
9         <amount>12.00</amount>
```

```
 </Trans>
 <Trans identifier="can be a string too">
  <username>vitale</username>
  <password>test</password>
  <action_trans>sale</action_trans>
  <account>5454545454545454</account>
  <expdate>0512</expdate>
  <amount>11.00</amount>
 </Trans>
 <Trans identifier='3'>
  <username>vitale</username>
  <password>test</password>
  <action_admin>report_tran</report_tran>
  <txnstatus>CAPTURED|UNCAPTURED</txnstatus>
 </Trans>
</MonetraTrans>
```

3.3.1.2 XML Response Example

Below is an example of an XML response.

```
<MonetraResp>
 <DataTransferStatus code='SUCCESS'/>
 <Resp identifier="can be a string too">
  <code>AUTH</code>
  <verbiage>APPROVAL 123456</verbiage>
  <batch>1</batch>
  <item>1</item>
  <avs>STREET</avs>
  <cv>GOOD</cv>
  <ttid>112</ttid>
 </Resp>
 <Resp identifier='3'>
  <code>SUCCESS</code>
  <DataBlock>
   ttid,type,capture, ...
   1,SALE,1, ...
   5,PREAUTH,0, ...
   7,RETURN,1, ...
  </DataBlock>
 </Resp>
 <Resp identifier='1'>
  <code>DENY</code>
  <verbiage>CVV2 MISMATCH</verbiage>
  <avs>GOOD</avs>
  <cv>BAD</cv>
  <ttid>3842</ttid>
 </Resp>
</MonetraResp>
```

3.3.1.3 XML Error Example

Below is an example of an XML response when the transaction was not structured properly.

```
<MonetraResp>
 <DataTransferStatus code='FAIL'>My descriptive failure reason</DataTransferStatus>
</MonetraResp>
```

3.3.2 JSON Transactions

Each set of JSON transactions must begin with `"MonetraTrans"`. Within this block is one or more transactions. Each transaction begins with a unique identifier of letters and/or numbers. The data for each transactions consists of the key/value pairs found in this specification. The order of the key/value data pairs is irrelevant, and Monetra does not return the response key/value pairs in any particular order.

JSON responses begin with `"MonetraResp"`. Within this block is a transaction response block for each transaction sent. The attribute title of each response block matches the corresponding transaction that was sent. Additionally, a `"DataTransferStatus"` block is always sent with a `code` attribute, which has a value of `SUCCESS` or `FAIL`. On `FAIL`, an error message is sent back in the `"verbiage"` attribute.

Every transaction response block has a `"code"` attribute indicating the overall outcome of the transaction. For comma-separated responses (like you receive when running reports), there is a `"DataBlock"` attribute that holds the response data. Otherwise, the response attribute pairs are those as defined in this guide.

Note: The fields in a `"DataBlock"` tag are guaranteed to not be removed or renamed, but fields can be added or rearranged at any time. Integrations need to parse the header first to find the correct column index for the desired field before reading out the value in that field and should never blindly assume that the data will always be at a specific column index.

3.3.2.1 JSON Request Example

Below is an example of a JSON request.

```
{
 "MonetraTrans": {
  "1": {
   "action": "ping"
  },

  "2": {
   "username": "test_retail:public",
   "password": "publ1ct3st",
   "action_trans": "sale",
   "amount": "2.00",
   "account": "5454545454545454",
   "expdate": "0219",
   "zip": "32606"
  },

  "doesn't have to be a number": {
```

```
          "username": "test_retail:public",
          "password": "publ1ct3st",
          "action_trans": "return",
          "amount": "3.00",
          "account": "5454545454545454",
          "expdate": "0219",
          "zip": "32606",
          "cvv": "123"
         },

         "numbers are typically easier than words or sentences": {
          "username": "test_retail:public",
          "password": "publ1ct3st",
          "action_admin": "report_tran",
          "txnstatus": "CAPTURED|UNCAPTURED"
         }
        }
       }

```

3.3.2.2 JSON Response Example

Below is an example of a JSON response.

```

       {
        "MonetraResp" : {
         "DataTransferStatus" : {
          "code" : "SUCCESS"
         },

         "1" : {
          "verbiage" : "AUTHENTICATION FAILED",
          "msoft_code" : "ACCT_AUTHFAILED",
          "phard_code" : "UNKNOWN",
          "code" : "DENY"
         },

         "2" : {
          "ttid" : "423542",
          "auth" : "214833",
          "timestamp" : "1442607738",
          "phard_code" : "SUCCESS",
          "avs" : "GOOD",
          "verbiage" : "APPROVED",
          "account" : "XXXXXXXXXXXX5454",
          "batch" : "393",
          "code" : "AUTH",
          "pclevel" : "0",
          "rcpt_host_ts" : "091815162218",
          "rcpt_entry_mode" : "M",
          "cardtype" : "MC",
          "item" : "5237",
          "msoft_code" : "INT_SUCCESS"
         },

         "doesn't have to be a number" : {
```

```
          "msoft_code" : "INT_SUCCESS",
          "cardtype" : "MC",
          "account" : "XXXXXXXXXXXX5454",
          "phard_code" : "UNKNOWN",
          "verbiage" : "SUCCESS",
          "pclevel" : "0",
          "code" : "AUTH",
          "rcpt_entry_mode" : "M",
          "item" : "5238",
          "timestamp" : "1442607738",
          "rcpt_host_ts" : "091815162218",
          "ttid" : "423543",
          "batch" : "393"
         },

         "numbers are typically easier than words or sentences" : {
          "code" : "SUCCESS",
          "DataBlock" : "ttid,type,capture, ...
           1,SALE,1, ...
           5,PREAUTH,0, ...
           7,RETURN,1, ..."
         }
        }
       }

```

3.3.2.3 JSON Error Example

Below is an example of a JSON response when the transaction was not structured properly.

```

       {
        "MonetraResp" : {
         "DataTransferStatus" : {
          "code" : "FAIL",
          "verbiage" : "My descriptive failure reason"
         }
        }
       }

```

3.3.3 ReST API

For information and details on the ReST API, please see the online documentation at https://developers.monetra.com/.