Monetra®

PHP API Developer Reference

Monetra PHP Reference v5.2
Updated November 2005

Table of Contents

1 Revision History

Date	***Rev.***	***Notes***
11/07/05	v5.2	Initial re-layout.

This page intentionally left blank.

2 Overview

2.1 Related Documentation

You will be required to review the Monetra Client Interface Protocol Specification (http://www.monetra.com/documentation.html) to cross-reference each transaction type, which will have multiple corresponding key/value pairs (ie. username, password, action, etc).

2.2 Introduction to the Monetra PHP API

The Monetra (MCVE) PHP API, which depends on libmonetra (C API), is designed to take advantage of all three of our "supported" communication methods, which include Drop-File, Unencrypted IP and Encrypted IP (SSL v3/TLS v1.0). Each method has its advantages and will be explained briefly below. Libmonetra is also the basis of the Perl, PHP and JAVA JNI modules, so the usage of those API's is nearly identical to Libmonetra itself, minus language semantics. In addition, this API was designed to be fully thread-safe and allows interleaving of transactions (streaming of transactions with out-of-order responses).

The Drop-File communication method is the most simplistic form of communication with Monetra. A transaction directory is specified where **.trn** (transaction) files are written, "picked up" and **.rtn** (response) files are written in reply. Advantages are the debug-ability and the fact that it does not require an IP stack to be present on the local machine. Although this method is not designed for networking, it is possible to share the transaction directory via NFS or SAMBA (for windows), to integrate with legacy applications. Because of security concerns, this should not be utilized for new integrations.

The unencrypted IP method is the fastest method of communication with Monetra. It requires the least amount of overhead and bypasses disk access. This method is perfect for locally "trusted" switched LANs or WANs, but should never be used on untrusted networks such as the Internet.

The encrypted IP (SSL) method is the most secure, requiring certificate verification and encryption to pass all data between the client and host. Most of the time, this is overkill for a local LAN or trusted WAN. SSL is most suitable for communication over the Internet or any untrusted network where the potential for eavesdropping is high. Newer monetra releases also support client certificate validation which is available in this API.

For any feature/anomaly, requests or support questions regarding libmonetra, feel free to contact our support staff via e-mail at support@mainstreetsoftworks.com .

2.3 Obtaining and Installing Libmonetra and the PHP API

Libmonetra may be obtained in source form from http://www.monetra.com/downloads.html or via ftp at ftp://ftp.monetra.com/pub/libmonetra. For 32bit Windows, it may also be obtained in binary form via a self-installing package from the same locations.

Please note: The PHP API for Monetra (MCVE) depends on Libmonetra 5.x and it must be installed **before** attempting to compile/install the PHP module. There are multiple ways to obtain the Monetra (MCVE) PHP API. For each method available, you must have the PHP development headers installed (for binary-based distributions, this is usually separated into a package typically named `php-dev`).

The easiest way to install the Monetra (MCVE) PHP API is to run these commands:

```
      pear upgrade pear       # May only be necessary on older
                              # PHP versions, v1.4.0RC2 or higher
                              # is required.  If you get errors, please
                              # upgrade any dependencies as well.
      pecl install mcve       # Please note this _does_ depend on
                              # libmonetra to have already been
                              # installed.
```

Those commands should compile and install the Monetra (MCVE) PHP API for you.

If you're using a release of PHP prior to 5.1, and you compile PHP by hand, you can alternatively, just add `--with-mcve` to your `./configure` arguments to add Monetra (MCVE) support.

Finally, you can always just download the latest release of the Monetra (MCVE) PHP API from the same location you downloaded Libmonetra. There should be a bundled ./configure script you can run. Then just install it as you would any other application in a Unix environment.

If you installed Monetra (MCVE) support as a loadable module, please modify your php.ini file to include this line:

```
      extension=mcve.so
```

Alternatively, you may call `dl("mcve.so")` from the script you wish to give Monetra (MCVE) support to, but it is often inconvenient to do so.

3 Using This Guide

LibMonetra only performs simple connection and transaction management facilities to the Monetra engine. Its API was created to be as minimalistic as possible, while being simple to use. It will pass the transaction set (a set of key/value pairs) to the Monetra engine and return to you a response. It provides additional parsing facilities for dealing with comma delimited responses as well. Please reference the Monetra Client Interface Protocol Specification located at http://www.monetra.com/documentation.html for the expected key/value pairs for each transaction and responses to those requests.

The basics for performing transactions for this guide include initializing the library, establishing a connection, structuring and sending one or more transactions, reading results and closing the connection/de-initializing the library.

You will note in this API that all parameters to functions are prefixed with an **[in]**, **[out]**, or **[in/out]** tag which indicates if you are receiving data into that parameter.

[in/out] means that the parameter's memory address may be updated upon return, but it must also have been initialized before being passed.

[out] means that the parameter's memory address will be updated upon return. You need to make sure this variable is passed by reference.

[in] means this is an input parameter used to tell the function what it needs to perform. This parameter should be passed normally (e.g. not by reference).

Please reference the examples in this document for basic API usage.

4 Constants and Data Types

4.1 Constants for M_ReturnStatus

4.1.1 M_ERROR

Value:	`-1`
Description:	Critical error. Status unknown

4.1.2 M_FAIL

Value:	`0`
Description:	Transaction or Audit Failed

4.1.3 M_SUCCESS

Value:	`1`
Description:	Transaction or Audit succeeded

4.2 Constants for M_CheckStatus

4.2.1 M_DONE

Value:	`2`
Description:	Transaction is complete

4.2.2 M_ERROR

Value:	`-1`
Description:	An error has occurred. Status unknown.

4.2.3 M_PENDING

Value:	`1`
Description:	Still waiting on transaction response from Monetra

5 PHP Functions

5.1 Initialization/Destruction of Library

5.1.1 M_DestroyEngine

Prototype: `void m_destroyengine(void)`
Description: frees any memory associated with the M_InitEngine call. Should be called just before a program terminates.
Return Value: none

5.1.2 M_InitEngine

Prototype: `int m_initengine(string location)`
Description: must be called before any other API calls. It is mainly used to initialize SSL calls, but on Windows, it also calls WSAStartup() to initialize BSD sockets.
[location] parameter should always be NULL. You should use M_SetSSL_CAfile to set the location on a per-connection basis.
Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in]	`location:`	SSL CA (Certificate Authority) file for verification remote SSL server (DEPRECATED, pass NULL, see notes above).

5.2 Initialization of Connections and Management

5.2.1 M_Connect

Prototype: `int m_connect(resource conn)`
Description: once all connection parameters have been set, this function establishes the connection to the Monetra daemon
Return Value: 1 on success, 0 on failure

Parameter Descriptions:
[in/out] `myconn`: Connection resource returned from M_InitConn()

5.2.2 M_ConnectionError

Prototype: `string m_connectionerror(resource conn)`
Description: if M_Connect returns a failure, this function may provide some text as an insight into what went wrong (such as timeout, or connection refused)
Return Value: textual error message associated with connection

Parameter Descriptions:
[in/out] `myconn`: Connection resource returned from M_InitConn()

5.2.3 M_DestroyConn

Prototype: `void m_destroyconn(resource conn)`
Description: disconnects from Monetra and deallocates any memory associated with the connection resource. This is not necessary in PHP, because it will automatically be destroyed upon script termination.
Return Value: none

Parameters Descriptions:
[in/out] `myconn`: Connection resource returned from M_InitConn()

5.2.4 M_InitConn

Prototype: `resource M_InitConn() (void)`
Description: allocates memory for the Connection Data Block and sets default values
Return Value: Resource for holding connection parameters

Parameter Descriptions:
N/A

5.2.5 M_MaxConnTimeout

Prototype: `bool m_maxconntimeout(resource conn, int secs)`
Description: sets how long libmonetra should try to connect to the Monetra server. This only has an effect when there are network problems and sets the socket into non blocking mode. Only relevant for IP or SSL connections.
Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `maxtime:`	maximum amount of time in seconds to wait to establish IP/SSL connection

5.2.6 M_SetBlocking

Prototype: `int m_setblocking(resource conn, int tf)`
Description: specifies whether to wait for a transaction to finish before returning from a M_TransSend or (legacy) M_Sale etc. (blocking), or to return immediately and make client check status (non-blocking)
Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `tf:`	1 if blocking is desired, 0 if blocking is not desired

5.2.7 M_SetDropFile

Prototype: `int m_setdropfile(resource conn, string directory)`
Description: sets the M_CONN parameter to use the Drop-File method of communication
Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `df_location:`	directory to write transaction files

5.2.8 M_SetIP

Prototype: `int m_setip(resource conn, string host, int port)`
Description: sets the M_CONN parameter to use the IP method of communication
Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `host:`	hostname or ip address to establish IP connection
[in] `port:`	port associated with ip/hostname

5.2.9 M_SetSSL

Prototype: `int m_setssl(resource conn, string host, int port)`
Description: sets the M_CONN parameter to use the SSL method of communication
Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `host:`	hostname or ip address to establish SSL connection
[in] `port:`	port associated with ip/hostname

5.2.10 M_SetSSL_CAfile

Prototype: `int m_setssl_cafile(resource conn, string cafile)`
Description: sets the CA file location before establishing a connection to a running Monetra engine. Used to verify the remote host's SSL certificate.
Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `path:`	CA (Certificate Authority) file path

5.2.11 M_SetSSL_Files

Prototype: `int m_setssl_files(resource conn, string sslkeyfile, string sslcertfile)`
Description: sets the client certificate and key used for verification if the remote Monetra engine has client SSL certificate verification enabled
Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `sslkeyfile:`	path of key file for client certificate
[in] `sslcertfile:`	path of certificate file for client certificate

5.2.12 M_SetTimeout

Prototype: `int m_settimeout(resource conn, int seconds)`
Description: sets the maximum amount of time a transaction can take before timing out. This values gets sent to the Monetra engine, the engine sends the TIMEOUT response to libmonetra, libmonetra never times out (and SHOULD NOT)
Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `timeout:`	maximum duration in seconds to wait for completion of transaction

5.2.13 M_ValidateIdentifier

Prototype: `int m_validateidentifier(resource conn, int tf)`
Description: tells the API whether or not the identifiers used for transactions should be validated from within the connection structure before assuming they are correct. Since the transaction identifier is actually a pointer address, passing an incorrect address can cause segmentation faults without verification. This is usually not necessary for C programs, but is helpful for writing modules for other languages.
Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `tf:`	1 if verification of transaction identifiers is desired, 0 if not desired [default 0]

5.2.14 M_VerifyConnection

Prototype: `bool m_verifyconnection(resource conn, int tf)`
Description: tells Monetra whether or not to send a PING request to the Monetra server once a connection has been established. Default is TRUE. This will probably only be used if trying to connect to a Monetra version < 2.1.
Return Value: none

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `tf:`	1 if SSL server certification verification is desired, 0 if not [default 0]

5.2.15 M_VerifySSLCert

Prototype: `bool m_verifysslcert(resource conn, int tf)`
Description: tells Monetra whether or not to verify that the SSL certificate provided by Monetra has been signed by a proper CA. Obviously, this is only applicable if using SSL connectivity.
Return Value: none

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `tf:`	1 if SSL server certification verification is desired, 0 if not [default 0]

5.3 Sending Transactions to Monetra

5.3.1 M_CheckStatus

Prototype: `int m_checkstatus(resource conn, int identifier)`
Description: returns the state of the transaction, whether or not processing has been complete or is still pending
Return Value: M_PENDING (1) if still being processed, M_DONE (2) if complete, <= 0 on critical failure

Parameter Descriptions:
[in/out] `myconn`: Connection resource returned from M_InitConn()
[in] `identifier`: reference for transaction as returned by M_TransNew()

5.3.2 M_CompleteAuthorizations

Prototype: `int m_completeauthorizations(resource conn, int &array)`
Description: gets how many transactions have been completed and loads the list of identifiers into listings
Return Value: number of transactions in the current connection queue which are complete (fully processed)

Parameter Descriptions:
[in/out] `myconn`: Connection resource returned from M_InitConn()
[out] `listings`: returns an identifier for each listing which is complete. Should free () the array returned here. Must not be NULL.

5.3.3 M_DeleteTrans

Prototype: `bool m_deletetrans(resource conn, int identifier)`
Description: removes a transaction from the queue that was initialized with M_TransNew
Return Value: none

Parameter Descriptions:
[in/out] `myconn`: Connection resource returned from M_InitConn()
[in] `identifier`: reference for transaction as returned by M_TransNew()

5.3.4 M_Monitor

Prototype: `int m_monitor(resource conn)`
Description: Performs all communication with the Monetra server. If this function never gets called, no transactions will be processed. Function is non-blocking, meaning it will return immediately if there is nothing to be done.
Return Value: 1 on success (connection alive), 0 on disconnect, -1 on critical error

Parameter Descriptions:
[in/out] `myconn`: Connection resource returned from M_InitConn()

5.3.5 M_TransInQueue

Prototype: `int m_transinqueue(resource conn)`
Description: returns the total number of transactions in the queue, no matter what state they are in or if they have been sent or not.
Return Value: number of transactions in the current connection queue

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()

5.3.6 M_TransKeyVal

Prototype: `int m_transkeyval(resource conn, long identifier, string key, string value)`
Description: adds a key/value pair for a transaction to be sent to Monetra
Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `identifier:`	reference for transaction as returned by M_TransNew()
[in] `key:`	key as referenced in the Monetra Client Interface Specification
[in] `value:`	value as referenced for key in Monetra Client Interface Specification

5.3.7 M_TransNew

Prototype: `int m_transnew(resource conn)`
Description: starts a new transaction. This must be called to obtain an identifier before any Transaction Parameters may be added.
Return Value: reference for transaction

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()

5.3.8 M_TransactionsSent

Prototype: `int m_transactionssent(resource conn)`
Description: checks to make sure the SEND queue for IP and SSL connections is empty. Useful for determining connection problems to Monetra.
Return Value: number of transactions sent to Monetra from this connection (that have not already been deleted).

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()

5.3.9 M_TransSend

Prototype: `int m_transsend(resource conn, long identifier)`
Description: finalizes a transaction and sends it to the Monetra server
Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `identifier:`	reference for transaction as returned by M_TransNew()

5.4 Dealing with Responses from Monetra

5.4.1 M_GetCell

Prototype:	`string m_getcell(resource conn, int identifier, string column, int row)`
Description:	gets a single cell from comma-delimited data (position independent M_ParseCommaDelimited must be called first.
Return Value:	data for particular cell

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `identifier:`	reference for transaction as returned by M_TransNew()
[in] `column:`	text key (header) name for cell
[in] `row:`	row number

5.4.2 M_GetCellByNum

Prototype:	`string m_getcellbynum(resource conn, int identifier, int column, int row)`
Description:	gets a single cell from comma-delimited data (position dependent M_ParseCommaDelimited must be called first.
Return Value:	data for particular cell

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `identifier:`	reference for transaction as returned by M_TransNew()
[in] `column:`	integer value for column number
[in] `row:`	row number

5.4.3 M_GetCommaDelimited

Prototype:	`string m_getcommadelimited(resource conn, int identifier)`
Description:	gets the raw comma-delimited data
Return Value:	raw transaction data returned by Monetra

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `identifier:`	reference for transaction as returned by M_TransNew()

5.4.4 M_GetHeader

Prototype: `string m_getheader(resource conn, int identifier, int column_num)`
Description: retrieval of a header by column number from comma-delimited data. M_ParseCommaDelimited must be called first.
Return Value: text name for column header

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `identifier:`	reference for transaction as returned by M_TransNew()
[in] `column_num:`	column number to retrieve header name

5.4.5 M_IsCommaDelimited

Prototype: `int m_iscommadelimited(resource conn, int identifier)`
Description: a quick check to see if the response that has been returned is comma-delimited or a standard response
Return Value: 1 if response is comma-delimited, 0 if not

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `identifier:`	reference for transaction as returned by M_TransNew()

5.4.6 M_NumColumns

Prototype: `int m_numcolumns(resource conn, int identifier)`
Description: the number of columns in a comma-delimited responseM_ParseCommaDelimited must be called first.
Return Value: number of columns for comma-delimited data

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `identifier:`	reference for transaction as returned by M_TransNew()

5.4.7 M_NumRows

Prototype: `int m_numrows(resource conn, int identifier)`
Description: the number of rows in a comma-delimited response. M_ParseCommaDelimited must be called first.
Return Value: number of rows for comma delimited data

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `identifier:`	reference for transaction as returned by M_TransNew()

5.4.8 M_ParseCommaDelimited

Prototype: `int m_parsecommadelimited(resource conn, int identifier)`
Description: tells libmonetra to use its internal parsing commands to parse the comma delimited response. This MUST be called before calls to M_GetCell, M_GetCellByNum, M_GetHeader, M_NumRows, and M_NumColumns.
Note: If you call M_ParseCommaDelimited, you can no longer call MonetraGetCommaDelimited because ParseCommaDelimited destroys the data.
Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `identifier:`	reference for transaction as returned by M_TransNew()

5.4.9 M_ResponseKeys

Prototype: `array m_responsekeys(resource conn, long identifier)`
Description: retrieves the response keys (parameters) returned from the Monetra engine for the particular transaction. Useful so you can pull the value using M_ResponseParam() for each key.
Return Value: array of strings which are the available keys in the response

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `identifier:`	reference for transaction as returned by M_TransNew()

5.4.10 M_ResponseParam

Prototype: `string m_responseparam(resource conn, long identifier, string key)`
Description: This function is used to retrieve the response key/value pairs from the Monetra Engine, as specified in the Monetra Client Interface Protocol Specification.
Return Value: value associated with the key requested. NULL if not found.

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `identifier:`	reference for transaction as returned by M_TransNew()
[in] `key:`	Response Parameter key as defined in the Monetra Client Interface Specification

5.4.11 M_ReturnStatus

Prototype:	`int m_returnstatus(resource conn, int identifier)`
Description:	returns a success/fail response for every transaction. If a detailed code is needed, please see M_ReturnCode
Return Value:	1 if transaction successful (authorization), 0 if transaction failed (denial)

Parameter Descriptions:

[in/out] `myconn:`	Connection resource returned from M_InitConn()
[in] `identifier:`	reference for transaction as returned by M_TransNew()

5.5 Miscellaneous Functions

5.5.1 M_SSLCert_gen_hash

Prototype:	string M_SSLCert_gen_hash(string filename)
Description:	generates hash content of client certificate for adding restrictions to user connections
Return Value:	certificate hash, or NULL on error

Parameter Descriptions:

[in] `filename:`	path to client certificate file

5.5.2 M_uwait

Prototype:	`int m_uwait(long microsecs)`
Description:	a microsecond sleep timer that uses select() with a NULL set to obtain a more efficient, cross-platform, thread-safe usleep();
Return Value:	1 on success, 0 on failure

Parameter Descriptions:

[in] `length:`	time in micro seconds to delay (1/1000000)

6 Examples

6.1 Full Transaction Examples

6.1.1 Basic Sale Transaction Code

```
<?php

$MYHOST=        "localhost";
$MYPORT=        8444;
$MYUSER=        "test-user";
$MYPASS=        "test-pass";
$MYMETHOD=      "SSL";
$MYCAFILE=      "/usr/local/monetra/CAfile.pem";
$MYVERIFYSSL=   1;


/* Initialize Engine */
if (!M_InitEngine(NULL)) {
        echo "Failed to initialize libmonetra\r\n";
        return;
}

/* Initialize Connection Resource */
$conn = M_InitConn();
if ($conn === FALSE) {
        echo "Failed to initialize connection resource\r\n";
        return;
}

if (strcasecmp($MYMETHOD, "SSL") == 0) {
        /* Set up SSL Connection Location */
        if (!M_SetSSL($conn, $MYHOST, $MYPORT)) {
                echo "Could not set method to SSL";
                /* Free memory associated with conn */
                M_DestroyConn($conn);
                return;
        }
        /* Set up information required to verify certificates */
        if ($MYVERIFYSSL) {
                if (!M_SetSSL_CAfile($conn, $MYCAFILE)) {
                        echo "Could not set SSL CAFile.  " .
                             "Does the file exist?\r\n";
                        M_DestroyConn($conn);
                        return;
                }
                M_VerifySSLCert($conn, 1);
        }
} else if (strcasecmp($MYMETHOD, "IP") == 0) {
        /* Set up IP Connection Location */
        if (!M_SetIP($conn, $MYHOST, $MYPORT)) {
                echo "Could not set method to IP\r\n";
                /* Free memory associated with conn */
                M_DestroyConn($conn);
                return;
        }
} else {
        echo "Invalid method '" . $MYMETHOD . "' specified!\r\n";
        /* Free memory associated with conn */
        M_DestroyConn($conn);
        return;
}

/* Set to non-blocking mode, means we must do
 * a M_Monitor() loop waiting on responses
 * Please see next example for blocking-mode */
if (!M_SetBlocking($conn, 0)) {
        echo "Could not set non-blocking mode\r\n";
        /* Free memory associated with conn */
        M_DestroyConn($conn);
        return;
}

/* Set a timeout to be appended to each transaction
 * sent to Monetra */
if (!M_SetTimeout($conn, 30)) {
        echo "Could not set timeout\r\n";
        /* Free memory associated with conn */
        M_DestroyConn($conn);
        return;
}

/* Connect to Monetra */
if (!M_Connect($conn)) {
        echo "Connection failed: " . M_ConnectionError($conn) .
             "\r\n";
        /* Free memory associated with conn */
        M_DestroyConn($conn); // free memory
        return;
}

/* Allocate new transaction */
$identifier=M_TransNew($conn);

/* User credentials */
M_TransKeyVal($conn, $identifier, "username", $MYUSER);
M_TransKeyVal($conn, $identifier, "password", $MYPASS);
/* Transaction Type */
M_TransKeyVal($conn, $identifier, "action", "sale");
/* Transaction Data */
M_TransKeyVal($conn, $identifier, "account", "4012888888881");
M_TransKeyVal($conn, $identifier, "expdate", "0512");
M_TransKeyVal($conn, $identifier, "amount", "12.00");
M_TransKeyVal($conn, $identifier, "ptrannum", "99999");

/* Add transaction to outgoing buffer */
if (!M_TransSend($conn, $identifier)) {
        echo "Transaction improperly structured, possibly" .
             "not enough info\r\n";
        /* Free memory associated with conn */
        M_DestroyConn($conn); /* free memory associated wit*/
        return;
}

/* Communication loop with Monetra. Loop until transaction
 * is complete */
while (M_CheckStatus($conn, $identifier) == M_PENDING) {
        if (M_Monitor($conn) != 1) {
                /* Disconnect has occurred, or other critical
                 * error */
                echo "Unexpected disconnect: " .
                     M_ConnectionError($conn) . "\r\n";
        }
        M_uwait(20000); /* Microsecond sleep timer */
}

/* Check success or Fail */
if (M_ReturnStatus($conn, $identifier) == M_SUCCESS) {
        echo "Transaction successful!\r\n";
} else if (M_ReturnStatus($conn, $identifier) == M_FAIL) {
        echo "Transaction failed!\r\n";
}

/* Get results */
$response_keys = M_ResponseKeys($conn, $identifier);
echo "Response Keys: Values\r\n";
for ($i=0; $i<count($response_keys); $i++) {
        echo $response_keys[$i] . " : " .
                M_ResponseParam($conn, $identifier,
                 $response_keys[$i]) . "\r\n";
}


/* Optionally clean up transaction memory, this is
 * automatically free()'d when the connection is destroyed */
M_DeleteTrans($conn, $identifier);

/* Clean up connection, and library instance */
M_DestroyConn($conn);
M_DestroyEngine();

?>
```

6.1.2 Requesting and Interpreting Reports

```
<?php

$MYHOST=        "localhost";
$MYPORT=        8444;
$MYUSER=        "test-user";
$MYPASS=        "test-pass";
$MYMETHOD=      "SSL";
$MYCAFILE=      "/usr/local/monetra/CAfile.pem";
$MYVERIFYSSL=   1;


/* Initialize Engine */
if (!M_InitEngine(NULL)) {
        echo "Failed to initialize libmonetra\r\n";
        return;
}

/* Initialize Connection */
$conn = M_InitConn();
if ($conn === FALSE) {
        echo "Failed to initialize connection resource\r\n";
        return;
}

if (strcasecmp($MYMETHOD, "SSL") == 0) {
        /* Set up SSL Connection Location */
        if (!M_SetSSL($conn, $MYHOST, $MYPORT)) {
                echo "Could not set method to SSL\r\n";
                /* Free memory associated with conn */
                M_DestroyConn($conn);
                return;
        }
        /* Set up information required to verify certificates */
        if ($MYVERIFYSSL) {
                if (!M_SetSSL_CAfile($conn, $MYCAFILE)) {
                        echo "Could not set SSL CAFile.  " .
                             "Does the file exist?\r\n";
                        M_DestroyConn($conn);
                        return;
                }
                M_VerifySSLCert($conn, 1);
        }
} else if (strcasecmp($MYMETHOD, "IP") == 0) {
        /* Set up IP Connection Location */
        if (!M_SetIP($conn, $MYHOST, $MYPORT)) {
                echo "Could not set method to IP\r\n";
                /* Free memory associated with conn */
                M_DestroyConn($conn);
                return;
        }
} else {
        echo "Invalid method '" . $MYMETHOD . "' specified!\r\n";
        /* Free memory associated with conn */
        M_DestroyConn($conn);
        return;
}

/* Set to blocking mode, means we do not have to
 * do a M_Monitor() loop, M_TransSend() will do this for us */
if (!M_SetBlocking($conn, 1)) {
        echo "Could not set non-blocking mode\r\n";
        /* Free memory associated with conn */
        M_DestroyConn($conn);
        return;
}

/* Set a timeout to be appended to each transaction
 * sent to Monetra */
if (!M_SetTimeout($conn, 30)) {
        echo "Could not set timeout\r\n";
        /* Free memory associated with conn */
        M_DestroyConn($conn);
        return;
}

/* Connect to Monetra */
if (!M_Connect($conn)) {
        echo "Connection failed: " .
                M_ConnectionError($conn) . "\r\n";
        /* Free memory associated with conn */
        M_DestroyConn($conn); // free memory
        return;
}

/* Allocate new transaction */
$identifier=M_TransNew($conn);

/* User credentials */
M_TransKeyVal($conn, $identifier, "username", $MYUSER);
M_TransKeyVal($conn, $identifier, "password", $MYPASS);
/* Transaction Type */
M_TransKeyVal($conn, $identifier, "action", "admin");
M_TransKeyVal($conn, $identifier, "admin", "GUT");
/* Additional Auditing parameters may be specified
 * Please consult the Monetra Client Interface Protocol */

if (!M_TransSend($conn, $identifier)) {
        echo "Communication Error: " .
                M_ConnectionError($conn) . "\r\n";
        /* Free memory associated with conn */
        M_DestroyConn($conn);
        return;
}
```

```
/* We do not have to perform the M_Monitor() loop
 * because we are in blocking mode */
if (M_ReturnStatus($conn, $identifier) != M_SUCCESS) {
        echo "Audit failed\r\n";
        M_DestroyConn($conn);
        return;
}
if (!M_IsCommaDelimited($conn, $identifier)) {
        echo "Not a comma delimited response!\r\n";
        M_DestroyConn($conn);
        return;
}

/* Print the raw, unparsed data */
echo "Raw Data:\r\n" . M_GetCommaDelimited($conn, $identifier) .
        "\r\n";

/* Tell the API to parse the Data */
if (!M_ParseCommaDelimited($conn, $identifier)) {
        echo "Parsing comma delimited data failed";
        M_DestroyConn($conn);
        return;
}

/* Retrieve each number of rows/columns */
$rows=M_NumRows($conn, $identifier);
$columns=M_NumColumns($conn, $identifier);

/* Print all the headers separated by |'s */
for ($i=0; $i<$columns; $i++) {
        if ($i != 0) echo "|";
        echo M_GetHeader($conn, $identifier, $i);
}
echo "\r\n";

/* Print one row per line, each cell separated by |'s */
for ($j=0; $j<$rows; $j++) {
        for ($i=0; $i<$columns; $i++) {
                if ($i != 0) echo "|";
                echo M_GetCellByNum($conn, $identifier, $i, $j);
        }
        echo "\r\n";
}

/* Use M_GetCell instead of M_GetCellByNum if you need a
 * specific column, as the results will allow for position-
 * independent searching of the results. The ordering of
 * returned headers may be different between Monetra versions,
 * so that is _highly_ recommended */


/* Optionally free transaction, though M_DestroyConn() will
```

```
 * do this for us */
M_DeleteTrans($conn, $identifier);

/* Clean up and close */
M_DestroyConn($conn);
M_DestroyEngine();

?>
```