Monetra®

C API (libmonetra) Developer Reference

LibMonetra Reference v7.0
Updated May 2008

Table of Contents

1 Revision History

Date	*Rev.*	*Notes*
05/08/08	V7.0	M_InitEngine_ex() function, M_Register_threadid() function, M_TransBinaryKeyVal() function, M_GetBinaryCell() function, and M_FreeBinaryCell() function. Update VB.Net and C#.Net examples.
09/03/05	v5.1	Initial re-layout. Delphi, VB.Net, C#.Net examples.

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 C API

The Monetra C API (libmonetra) 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 Libmonetra

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 and 64bit Windows, it may also be obtained in binary form from the same locations.

2.4 Compiling Applications with libmonetra

By default, in a Unix environment the libmonetra libraries are installed to /usr/local/lib and the headers are installed to / usr/local/include, so we will use those directories for the rest of this section. Obviously, if you've changed the base location by using the --prefix command in the ./configure script, you'll need to note the respective differences.

For windows, there is no default installation path, you must put the headers and libraries where you want them.

The first thing that needs to be mentioned is that every program that uses the libmonetra function must
```
#include <monetra.h>
```
There are no other required headers. OpenSSL library headers are not necessary.

If you are using Linux and have installed the files to the default locations, it is recommend that you ensure that your ld.so.conf file, located in the /etc/ directory, has been updated to include /usr/local/lib in its search path. Another option for Linux users that will work on Unix would be exporting the LD_LIBRARY_PATH environment variable so that it has that path on its list.

Depending on with which library you plan on linking (static or shared) and the operating system on which you have built, some differences in additional required libraries/compiler options exist.

2.4.1 Shared Library Requirements

Shared libraries should automatically link in any other required libraries. If you get link errors though, it's possible that you may need to link in the OpenSSL libraries or possibly others. Please see the section on Static Library Requirements for those possibilities.

2.4.2 Static Library Requirements

Static library requirements vary among operating systems.

If you've enabled SSL support, add -lssl and -lcrypto to the compile options. Some operating systems such as Solaris and SCO may require -lsocket, -lnsl, and -lresolv to be added to provide socket and dns services to libmonetra.

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. This is typically used for the M_CONN parameters which need to be passed by reference. Typically though, this parameter's address is not modified.

[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

4.3 Common Data Types

4.3.1 M_CONN

Definition: `typedef void * M_CONN;`

Description: Data blob that gets passed around by reference from function to function which stores such data as connection parameters, input/output buffers, transaction queues, etc.

4.3.2 M_int64

Definition:

(win32) `typedef signed __int64 M_int64;`

(unix w/stdint) `typedef int64_t M_int64;`

(unix w/o stdint) `typedef signed long long M_int64;`

Description: Some integers in Monetra are made to be 64bit to prevent duplicates on high-volume systems. This data type tries to guarantee 64bit integers on all systems.

4.3.3 M_uintptr

Definition:

(win32) `typedef unsigned long M_uintptr;`

(unix w/stdint) `typedef uintptr_t M_uintptr;`

(unix w/o stdint) `typedef unsigned long M_uintptr;`

Description: Pointer to a memory address which references where in the connection queue a particular transaction exists. The value of this variable should only be set by a libmonetra function. Improper use can cause undefined behavior.

4.4 Data Types for Thread Safety

4.4.1 M_Mutex_Lock

Definition: `typedef int (* M_Mutex_Lock)(void *);`
Description: Defines prototype for callback to lock a system mutex.
Return: The function should return 1 on success, 0 on failure

4.4.2 M_Mutex_Unlock

Definition: `typedef int (* M_Mutex_Unlock)(void *);`
Description: Defines prototype for callback to unlock a system mutex.
Return: The function should return 1 on success, 0 on failure

4.4.3 M_Register_Mutex

Definition: `typedef void *(* M_Register_Mutex)(void);`
Description: Defines prototype for callback to register a system mutex.
Return: The function should return a pointer to the mutex on success, or NULL on failure

4.4.4 M_Unregister_Mutex

Definition: `typedef int (* M_Unregister_Mutex)(void *);`
Description: Defines prototype for callback to unregister a system mutex.
Return: The function should return 1 on success, 0 on failure

4.4.5 M_ThreadID

Definition: `typedef unsigned long (* M_ThreadID)(void);`
Description: Defines prototype for callback to return the current threadid as an unsigned long. Introduced with libmonetra v5.4.
Return: The function should return the current threadid.

5 C API Functions

5.1 Initialization/Destruction of Library

5.1.1 M_DestroyEngine

Prototype: `void M_DestroyEngine();`
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_ex

Prototype: `int M_InitEngine_ex(enum m_ssllock_style lockstyle);`
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.
Introduced with libmonetra v5.4.
Return Value: 1 on success, 0 on failure

Parameter Descriptions:
[in] `lockstyle`:

M_SSLLOCK_EXTERNAL (1):	OpenSSL's locks are guaranteed to be initialized externally, OpenSSL itself would also be initialized externally.
M_SSLLOCK_INTERNAL (2):	LibMonetra is responsible for all OpenSSL code. OpenSSL should not be used outside of libmonetra calls in an application if this is used, especially in multi-threaded code. The OpenSSL locks are initialized on the first call to M_EnableThreadSafety(). Use this if not multi-threaded as well.

5.1.3 M_InitEngine (deprecated)

Prototype: `int M_InitEngine(const char *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.
NOTE: Use M_InitEngine_ex() instead.
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(M_CONN *myconn);`
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`: M_CONN structure passed by reference, or an allocated M_CONN pointer

5.2.2 M_ConnectionError

Prototype: `const char *M_ConnectionError(M_CONN *myconn);`
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`: M_CONN structure passed by reference, or an allocated M_CONN pointer

5.2.3 M_DestroyConn

Prototype: `void M_DestroyConn(M_CONN *myconn);`
Description: disconnects from Monetra and deallocates any memory associated with the M_CONN structure
Return Value: none

Parameters Descriptions:
[in/out] `myconn`: M_CONN structure passed by reference, or an allocated M_CONN pointer

5.2.4 M_InitConn

Prototype: `void M_InitConn(M_CONN *myconn);`
Description: allocates memory for the Connection Data Block and sets default values
Return Value: none

Parameter Descriptions:
[out] `myconn`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `tf`: 1 if blocking is desired, 0 if blocking is not desired

5.2.5 M_MaxConnTimeout

Prototype: `void M_MaxConnTimeout(M_CONN *myconn, int maxtime);`
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: none

Parameter Descriptions:
[in/out] `myconn`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `maxtime`: maximum amount of time in seconds to wait to establish IP/SSL connection

5.2.6 M_SetBlocking

Prototype: `int M_SetBlocking(M_CONN *myconn, 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`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `tf`: 1 if blocking is desired, 0 if blocking is not desired

5.2.7 M_SetDropFile

Prototype: `int M_SetDropFile(M_CONN *myconn, const char *df_location);`
Description: sets the M_CONN parameter to use the Drop-File method of communication NOTE: THIS FUNCTION IS DEPRECATED AND NOT RECOMMENDED
Return Value: 1 on success, 0 on failure

Parameter Descriptions:
[in/out] `myconn`: M_CONN structure passed by reference or an allocated M_CONN pointer
[in] `df_location`: directory to write transaction files

5.2.8 M_SetIP

Prototype: `int M_SetIP(M_CONN *myconn, const char *host, unsigned short 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`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `host`: hostname or ip address to establish IP connection
[in] `port`: port associated with ip/hostname

5.2.9 M_SetLogging

Prototype: `int M_SetLogging(M_CONN *myconn, int level);`
Description: tells Monetra to log all libmonetra communications to a file. The filename will always be /tmp/libmonetra-%d.log where %d is the pid of the running process.
Return Value: 1 on success, 0 on failure

Parameter Descriptions:
[in/out] `myconn`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `level`: Debug level to enable. Currently only '1' is used.

5.2.10 M_SetSSL

Prototype: `int M_SetSSL(M_CONN *myconn, const char *host, unsigned short 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`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `host`: hostname or ip address to establish SSL connection
[in] `port`: port associated with ip/hostname

5.2.11 M_SetSSL_CAfile

Prototype: `int M_SetSSL_CAfile(M_CONN *myconn, const char *path);`
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:` M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `path:` CA (Certificate Authority) file path

5.2.12 M_SetSSL_Files

Prototype: `int M_SetSSL_Files(M_CONN *myconn, const char *sslkeyfile, const char *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:` M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `sslkeyfile:` path of key file for client certificate
[in] `sslcertfile:` path of certificate file for client certificate

5.2.13 M_SetTimeout

Prototype: `int M_SetTimeout(M_CONN *myconn, long timeout);`
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:` M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `timeout:` maximum duration in seconds to wait for completion of transaction

5.2.14 M_ValidateIdentifier

Prototype: `int M_ValidateIdentifier(M_CONN * myconn, 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`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `tf`: 1 if verification of transaction identifiers is desired, 0 if not desired [default 0]
[in] `myconn`: M_CONN structure passed by reference, or an allocated M_CONN pointer

5.2.15 M_VerifyConnection

Prototype: `void M_VerifyConnection(M_CONN *myconn, 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`: M_CONN structure passed by reference or an allocated M_CONN pointer
[in] `tf`: 1 if SSL server certification verification is desired, 0 if not [default 0]

5.2.16 M_VerifySSLCert

Prototype: `void M_VerifySSLCert(M_CONN *myconn, 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`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[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(M_CONN *myconn, M_uintptr 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`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `identifier`: reference for transaction as returned by M_TransNew()

5.3.2 M_CompleteAuthorizations

Prototype:
```
long M_CompleteAuthorizations(M_CONN *myconn,
                              M_uintptr **listings);
```
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`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[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_CompleteAuthorizations_index

Prototype: `M_uintptr M_CompleteAuthorizations_index(`
`M_uintptr *listings, int num_listings,`
`int index);`

Description: Primarily for those languages that do not easily provide direct C array access such as VB .Net and Delphi, this function is used to return the specific element of the array. It is typically more efficient for native C applications to directly access the array rather than use this function. Index starts at 0.

Return Value: M_uintptr result, data stored at the indexed position.

Parameter Descriptions:

[in] `listings:` Array of listings returned from M_CompleteAuthorizations() (from second arg passed by reference)

[in] `num_listings:` Number of elements in array as returned by M_CompleteAuthorizations()

[in] `index:` Index from array to retrieve, starting at 0.

5.3.4 M_DeleteTrans

Prototype: `void M_DeleteTrans(M_CONN *myconn, M_uintptr identifier);`

Description: removes a transaction from the queue that was initialized with M_TransNew

Return Value: none

Parameter Descriptions:

[in/out] `myconn:` M_CONN structure passed by reference, or an allocated M_CONN pointer

[in] `identifier:` reference for transaction as returned by M_TransNew()

5.3.5 M_FreeCompleteAuthroizations

Prototype: `int M_FreeCompleteAuthorizations(M_uintptr *listings,`
`int num_listings);`

Description: Frees the memory associated with the array returned from M_CompleteAuthorizations(). This is primarily used by those languages that do not have direct access to the free() function. This function is equivalent to performing free(listings); in C.

Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in] `listings:` Array of listings returned from M_CompleteAuthorizations() (from second arg passed by reference)

[in] `num_listings:` Number of elements in the array, as returned by M_CompleteAuthorizations()

5.3.6 M_Monitor

Prototype: `int M_Monitor(M_CONN *myconn);`
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`: M_CONN structure passed by reference, or an allocated M_CONN pointer

5.3.7 M_TransInQueue

Prototype: `long M_TransInQueue(M_CONN *myconn);`
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`: M_CONN structure passed by reference, or an allocated M_CONN pointer

5.3.8 M_TransKeyVal

Prototype:
```
int M_TransKeyVal (M_CONN * myconn, M_uintptr identifier,
                   const char *key, const char *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`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[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

5.3.9 M_TransBinaryKeyVal

Prototype: `int M_TransKeyVal (M_CONN * myconn, M_uintptr identifier, const char *key, const char *value, int value_len);`

Description: adds a key/value pair for a transaction to be sent to Monetra. The value must be a binary format (such as an image upload), for a key which expects binary data. The value will be base64 encoded prior to being sent to Monetra.

Return Value: 1 on success, 0 on failure

Parameter Descriptions:

[in/out] `myconn`: M_CONN structure passed by reference, or an allocated M_CONN pointer

[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

5.3.10 M_TransNew

Prototype: `M_uintptr M_TransNew(M_CONN *myconn);`

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`: M_CONN structure passed by reference, or an allocated M_CONN pointer

5.3.11 M_TransactionsSent

Prototype: `int M_TransactionsSent(M_CONN *myconn);`

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`: M_CONN structure passed by reference, or an allocated M_CONN pointer

5.3.12 M_TransSend

Prototype: `int M_TransSend(M_CONN *myconn, M_uintptr 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`: M_CONN structure passed by reference, or an allocated M_CONN pointer

[in] `identifier`: reference for transaction as returned by M_TransNew()

5.4 Dealing with Responses from Monetra

5.4.1 M_FreeResponseKeys

Prototype: `int M_FreeResponseKeys(char **keys, int num_keys);`
Description: frees the returned result from M_ResponseKeys
Return Value: 1 on success, 0 on failure

Parameter Descriptions:
[in] `keys:` array as returned by M_ResponseKeys()
[in] `num_keys:` integer value returned from M_ResponseKeys() parameter

5.4.2 M_GetCell

Prototype:
```
const char *M_GetCell(M_CONN *myconn, M_uintptr identifier,
                      const char *column, long 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:` M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `identifier:` reference for transaction as returned by M_TransNew()
[in] `column:` text key (header) name for cell
[in] `row:` row number

5.4.3 M_GetBinaryCell

Prototype:
```
char *M_GetBinaryCell(M_CONN *myconn, M_uintptr identifier,
                      const char *column, long row,
                      int *out_len);
```
Description: gets a single binary cell from comma-delimited data (position independent) M_ParseCommaDelimited must be called first. If the cell exists, it will attempt to base64 decode it and return it as binary data. This function should only be used on known binary cells.
Return Value: data for particular cell. Must be free'd using free() or M_FreeBinaryCell()

Parameter Descriptions:
[in/out] `myconn:` M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `identifier:` reference for transaction as returned by M_TransNew()
[in] `column:` text key (header) name for cell
[in] `row:` row number
[out] `out_len:` length of binary data returned

5.4.4 M_GetCellByNum

Prototype: `const char *M_GetCellByNum(M_CONN *myconn, M_uintptr identifier, int column, long 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`: M_CONN structure passed by reference, or an allocated M_CONN pointer

[in] `identifier`: reference for transaction as returned by M_TransNew()

[in] `column`: integer value for column number

[in] `row`: row number

5.4.5 M_FreeBinaryCell

Prototype: `void M_FreeBinaryCell(char *cell);`

Description: Free's cell data returned from M_GetBinaryCell(). Currently this function only calls free(), so is mostly suited for use by non-C languages (such as C#, VB.Net, or Delphi).

Return Value: none

Parameter Descriptions:

[in] `cell`: Cell data returned by M_GetBinaryCell

5.4.6 M_GetCommaDelimited

Prototype: `const char *M_GetCommaDelimited(M_CONN *myconn, M_uintptr identifier);`

Description: gets the raw comma-delimited data

Return Value: raw transaction data returned by Monetra

Parameter Descriptions:

[in/out] `myconn`: M_CONN structure passed by reference, or an allocated M_CONN pointer

[in] `identifier`: reference for transaction as returned by M_TransNew()

5.4.7 M_GetHeader

Prototype: `const char *M_GetHeader(M_CONN *myconn, M_uintptr 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`: M_CONN structure passed by reference, or an allocated M_CONN pointer

[in] `identifier`: reference for transaction as returned by M_TransNew()

[in] `column_num`: column number to retrieve header name

5.4.8 M_IsCommaDelimited

Prototype: `int M_IsCommaDelimited(M_CONN *myconn, M_uintptr 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`: M_CONN structure passed by reference, or an allocated M_CONN pointer

[in] `identifier`: reference for transaction as returned by M_TransNew()

5.4.9 M_NumColumns

Prototype: `int M_NumColumns(M_CONN *myconn, M_uintptr 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`: M_CONN structure passed by reference, or an allocated M_CONN pointer

[in] `identifier`: reference for transaction as returned by M_TransNew()

5.4.10 M_NumRows

Prototype: `long M_NumRows(M_CONN *myconn, M_uintptr 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`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `identifier`: reference for transaction as returned by M_TransNew()

5.4.11 M_ParseCommaDelimited

Prototype:
```
int M_ParseCommaDelimited(M_CONN *myconn,
                          M_uintptr 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`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `identifier`: reference for transaction as returned by M_TransNew()

5.4.12 M_ResponseKeys

Prototype:
```
char **M_ResponseKeys(M_CONN *myconn, M_uintptr identifier,
                      int *num_keys);
```
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`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `identifier`: reference for transaction as returned by M_TransNew()
[out] `num_keys`: pass integer by reference. Returns number of keys in array.

5.4.13 M_ResponseKeys_index

Prototype: `const char *M_ResponseKeys_index(char **keys, int num_keys, int index);`

Description: This function is primarily for those languages such as VB.Net or Delphi that do not easily support direct C array access. It is typically more efficient to directly access the array member. The index starts at position 0.

Return Value: the string stored in the index of the array specified, or NULL if invalid

Parameter Descriptions:

[in] `keys:` Returned array from M_ResponseKeys

[in] `num_keys:` Number of keys returned in array (value passed-by-reference to M_ResponseKeys)

[in] `index:` Index of element you are retrieving, starting at 0

5.4.14 M_ResponseParam

Prototype: `const char *M_ResponseParam(M_CONN *myconn, M_uintptr identifier, const char *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:` M_CONN structure passed by reference, or an allocated M_CONN pointer

[in] `identifier:` reference for transaction as returned by M_TransNew()

[in] `key:` Response Parameter key as defined in the Monetra Client Interface Specification

5.4.15 M_ReturnStatus

Prototype: `int M_ReturnStatus(M_CONN *myconn, M_uintptr 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:` M_CONN structure passed by reference, or an allocated M_CONN pointer

[in] `identifier:` reference for transaction as returned by M_TransNew()

5.5 Miscellaneous Functions

5.5.1 M_SSLCert_gen_hash

Prototype: `char *M_SSLCert_gen_hash(const char *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(unsigned long length);`
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)

5.6 Thread Safety Functions

5.6.1 M_EnableThreadSafety

Prototype: `int M_EnableThreadSafety(M_CONN *myconn);`
Description: After registering mutex callbacks, you must enable the use of them by calling this function. This function must be called before M_Connect(). Enabling thread safety is important if you have more than one thread that wishes to act upon a single connection at the same time. Most programs do not have this need, but if you do, make sure you register the callbacks, and enable this function.
Return Value: 1 on success, 0 on failure

Parameter Descriptions:
[in/out] `myconn`: M_CONN structure passed by reference, or an allocated M_CONN pointer

5.6.2 M_Register_mutexdestroy

Prototype:
```
int M_Register_mutexdestroy(M_CONN *myconn,
                            M_Unregister_Mutex reg);
```
Description: Register a callback to destroy the mutex
Return Value: 1 on success, 0 on failure

Parameter Descriptions:
[in/out] `myconn`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `reg`: pointer to function to call when doing a mutex destroy

5.6.3 M_Register_mutexinit

Prototype:
```
int M_Register_mutexinit(M_CONN *myconn,
                         M_Register_Mutex reg);
```
Description: Register a callback to intialize a mutex
Return Value: 1 on success, 0 on failure

Parameter Descriptions:
[in/out] `myconn`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `reg`: pointer to function to call when doing a mutex initialization

5.6.4 M_Register_mutexlock

Prototype: `int M_Register_mutexlock(M_CONN *myconn, M_Mutex_Lock reg);`
Description: Register a callback to lock the mutex
Return Value: 1 on success, 0 on failure

Parameter Descriptions:
[in/out] `myconn`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `reg`: pointer to function to call when doing a mutex lock

5.6.5 M_Register_mutexunlock

Prototype:
```
int M_Register_mutexunlock(M_CONN *myconn,
                           M_Mutex_Unlock reg);
```
Description: Register a callback to unlock the mutex
Return Value: 1 on success, 0 on failure

Parameter Descriptions:
[in/out] `myconn`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `reg`: pointer to function to call when doing a mutex unlock

5.6.6 M_Register_threadid

Prototype:
```
int M_Register_threadid(M_CONN *myconn,
                        M_ThreadID     reg);
```
Description: Register a callback to the current threadid
Return Value: 1 on success, 0 on failure

Parameter Descriptions:
[in/out] `myconn`: M_CONN structure passed by reference, or an allocated M_CONN pointer
[in] `reg`: pointer to function to call when doing a threadid lookup

6 Examples

6.1 Full Transaction Examples

6.1.1 Basic Sale Transaction Code

```
#include <stdio.h>
#include <string.h>
#include <monetra.h>

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

int main(int argc, char **argv)
{
        M_CONN conn = NULL;
        M_uintptr identifier = 0;
        const char *temp = NULL;
        char **response_keys = NULL;
        int num_response_keys = 0, i;

        /* Initialize Engine, we're not using threads so
         * M_SSLLOCK_INTERNAL is appropriate */
        if (!M_InitEngine(M_SSLLOCK_INTERNAL)) {
                printf("Failed to initialize libmonetra\r\n");
                return(1);
        }

        /* Initialize Connection */
        M_InitConn(&conn);

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

```
        return(1);
}

/* 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)) {
        printf("Could not set non-blocking mode\r\n");
        /* Free memory associated with conn */
        M_DestroyConn(&conn);
        return(-1);
}

/* Set a timeout to be appended to each transaction
 * sent to Monetra */
if (!M_SetTimeout(&conn, 30)) {
        printf("Could not set timeout\r\n");
        /* Free memory associated with conn */
        M_DestroyConn(&conn);
        return(-1);
}

/* Connect to Monetra */
if (!M_Connect(&conn)) {
        printf("Connection failed: %s\r\n",
                M_ConnectionError(&conn));
        /* Free memory associated with conn */
        M_DestroyConn(&conn); // free memory
        return(-1);
}

/* 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)) {
        printf("Transaction improperly structured, possibly"
                "not enough info\r\n");
        /* Free memory associated with conn */
        M_DestroyConn(&conn); /* free memory associated wit*/
        return(-1);
}

/* 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 */
                printf("Unexpected disconnect: %s\r\n",
                        M_ConnectionError(&conn));
        }
                M_uwait(20000); /* Microsecond sleep timer */
}

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

/* Get results */
response_keys = M_ResponseKeys(&conn, identifier,
                                        &num_response_keys);
```

```
        printf("Response Keys: Values\r\n");
        for (i=0; i<num_response_keys; i++) {
                printf("%s: %s\r\n", response_keys[i],
                         M_ResponseParam(&conn, identifier,
                                                 response_keys[i]));
        }
        M_FreeResponseKeys(response_keys, num_response_keys);

        /* 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();
        return(0);
}
```

6.1.2 Requesting and Interpreting Reports

```
#include <stdio.h>
#include <string.h>
#include <monetra.h>

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

int main(int argc, char **argv)
{
        M_CONN conn = NULL;
        M_uintptr identifier = 0;
        int rows=0, columns=0, i, j;

        /* Initialize Engine, we're not using threads,
         * so M_SSLLOCK_INTERNAL is appropriate */
        if (!M_InitEngine_ex(M_SSLLOCK_INTERNAL)) {
                printf("Failed to initialize libmonetra\r\n");
                return(1);
        }

        /* Initialize Connection */
        M_InitConn(&conn);

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

        /* 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)) {
                printf("Could not set non-blocking mode\r\n");
                /* Free memory associated with conn */
                M_DestroyConn(&conn);
                return(-1);
        }

        /* Set a timeout to be appended to each transaction
```

```
         * sent to Monetra */
        if (!M_SetTimeout(&conn, 30)) {
                printf("Could not set timeout\r\n");
                /* Free memory associated with conn */
                M_DestroyConn(&conn);
                return(-1);
        }

        /* Connect to Monetra */
        if (!M_Connect(&conn)) {
                printf("Connection failed: %s\r\n",
                         M_ConnectionError(&conn));
                /* Free memory associated with conn */
                M_DestroyConn(&conn); // free memory
                return(-1);
        }

        /* 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)) {
                printf("Communication Error: %s\r\n",
                        M_ConnectionError(&conn));
                /* Free memory associated with conn */
                M_DestroyConn(&conn);
                return(1);
        }

        /* We do not have to perform the M_Monitor() loop
         * because we are in blocking mode */
        if (M_ReturnStatus(&conn, identifier) != M_SUCCESS) {
                printf("Audit failed\r\n");
                M_DestroyConn(&conn);
                return(1);
        }
        if (!M_IsCommaDelimited(&conn, identifier)) {
                printf("Not a comma delimited response!\r\n");
                M_DestroyConn(&conn);
                return(1);
        }

        /* Print the raw, unparsed data */
        printf("Raw Data:\r\n%s\r\n",
                 M_GetCommaDelimited(&conn, identifier));

        /* Tell the API to parse the Data */
        if (!M_ParseCommaDelimited(&conn, identifier)) {
                printf("Parsing comma delimited data failed");
                M_DestroyConn(&conn);
                return(1);
        }

        /* 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) printf("|");
                printf("%s", M_GetHeader(&conn, identifier, i));
        }
        printf("\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) printf("|");
```

```
                        printf("%s", M_GetCellByNum(&conn, identifier,
                                                            i, j));
                }
                printf("\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();
        return(0);
}
```

6.2 Setting Up Thread Safety Example

6.2.1 PThread Mutex Linux/Unix example

```
#include <stdio.h>
#include <stdlib.h>
#include <monetra.h>
#include <pthread.h>

/* Return NULL on error, pointer to mutex on success */
static void *pthread_register_mutex(void)
{
        pthread_mutex_t *mutex = NULL;
        mutex = (pthread_mutex_t *)malloc(sizeof(pthread_mutex_t));
        if (mutex == NULL) return(NULL);
        pthread_mutex_init(mutex);
        return((void *)mutex);
}

/* Return 1 on success, 0 on failure */
static int pthread_unregister_mutex(void *mutex)
{
        pthread_mutex_destroy((pthread_mutex_t *)mutex);
        free(mutex);
        return(1);
}

/* Return 1 on success, 0 on failure */
static int pthread_dolock_mutex(void *mutex)
{
        if (pthread_mutex_lock((pthread_mutex_t *)mutex) != 0)
                return(0);
        return(1);
}

/* Return 1 on success, 0 on failure */
static int pthread_dounlock_mutex(void *mutex)
{
        if (pthread_mutex_unlock((pthread_mutex_t *)mutex) != 0)
                return(0);
        return(1);
}

static unsigned long pthread_thread_id(void)
{
        return((unsigned long)pthread_self());
}

int main(int argc, char **argv)
{
        M_CONN conn;
        M_InitEngine_ex(M_SSLLOCK_INTERNAL);
        M_InitConn(&conn);
        M_Register_mutexinit(&conn,
                (M_Register_Mutex)pthread_register_mutex);
        M_Register_mutexdestroy(&conn,
                (M_Unregister_Mutex)pthread_unregister_mutex);
        M_Register_mutexlock(&conn, (M_Mutex_Lock)pthread_dolock_mutex);
        M_Register_mutexunlock(&conn,
                (M_Mutex_Unlock)pthread_dounlock_mutex);
        M_Register_threadid(&conn, (M_ThreadID)pthread_thread_id);
        M_EnableThreadSafety(&conn);

        /* Perform remaining connection setup info here ... */

        /* ... */

        /* Cleanup */
        M_DestroyConn(&conn);
        M_DestroyEngine();
```

```
        return(1);
}
```

6.2.2 Win32 Mutex example

```
#include <stdio.h>
#include <stdlib.h>
#include <monetra.h>
#include <windows.h>

/* Return NULL on error, pointer to mutex on success */
static void *win32_register_mutex(void)
{
        void *mutex = NULL;
        mutex = malloc(sizeof(CRITICAL_SECTION));
        InitializeCriticalSection((LPCRITICAL_SECTION)mutex);
        return(mutex);
}

/* Return 1 on success, 0 on failure */
static int win32_unregister_mutex(void *mutex)
{
        DeleteCriticalSection((LPCRITICAL_SECTION)mutex);
        free(mutex);
        return(1);
}

/* Return 1 on success, 0 on failure */
static int win32_lock_mutex(void *mutex)
{
        EnterCriticalSection((LPCRITICAL_SECTION)mutex);
        return(1);
}

/* Return 1 on success, 0 on failure */
static int win32_unlock_mutex(void *mutex)
{
        LeaveCriticalSection((LPCRITICAL_SECTION)mutex);
        return(1);
}

static unsigned long win32_thread_id(void)
{
        return((unsigned long)GetCurrentThreadId());
}

int main(int argc, char **argv)
{
        M_CONN conn;
        M_InitEngine_ex(M_SSLLOCK_INTERNAL);
        M_InitConn(&conn);
        M_Register_mutexinit(&conn,
                (M_Register_Mutex)win32_register_mutex);
        M_Register_mutexdestroy(&conn,
                (M_Unregister_Mutex)win32_unregister_mutex);
        M_Register_mutexlock(&conn, (M_Mutex_Lock)win32_lock_mutex);
        M_Register_mutexunlock(&conn,
                (M_Mutex_Unlock)win32_unlock_mutex);
        M_Register_threadid(&conn, (M_ThreadID)win32_thread_id);
        M_EnableThreadSafety(&conn);

        /* Perform remaining connection setup info here ... */

        /* ... */

        /* Cleanup */
        M_DestroyConn(&conn);
        M_DestroyEngine();
        return(1);
}
```

A Appendix: Calling LibMonetra from other Languages

A.1 Borland Delphi

The example below demonstrates how to call libmonetra.dll from Borland Delphi (Tested with version 2005). To use this example, create a console project name Project1 and use the code below as Project1.pas . When you execute the test, it will connect over the Internet to our test server at testbox.monetra.com and run the transaction. Notice each function from libmonetra must be prototyped for Delphi. Please reference the C prototypes in the previous sections to see how different variable types map.

```
program Project1;

{$APPTYPE CONSOLE}

{$R *.RES}

type
        M_CONN = Pointer;
        M_UINTPTR = LongWord;

var
        conn: M_CONN;
        id: M_UINTPTR;

const
        host = 'testbox.monetra.com';
        port = 8333;

function M_InitEngine(CAfile: PChar): Boolean; stdcall; external 'libmonetra.dll';
function M_InitConn(var conn: M_CONN): Boolean; stdcall; external 'libmonetra.dll';
function M_SetIP(var conn: M_CONN; host: PChar; port: Integer): Boolean; stdcall; external
'libmonetra.dll';
function M_SetBlocking(var conn: M_CONN; tf: Integer): Boolean; stdcall; external 'libmonetra.dll';
function M_ConnectionError(var conn: M_CONN): PChar; stdcall; external 'libmonetra.dll';
function M_Connect(var conn: M_CONN): Boolean; stdcall; external 'libmonetra.dll';
function M_TransNew(var conn: M_CONN): M_UINTPTR; stdcall; external 'libmonetra.dll';
function M_TransKeyVal(var conn: M_CONN; id: M_UINTPTR; key: PChar; val: PChar): Boolean; stdcall;
external 'libmonetra.dll';
function M_TransSend(var conn: M_CONN; id: M_UINTPTR): Boolean; stdcall; external 'libmonetra.dll';
procedure M_DestroyConn(var conn: M_CONN); stdcall; external 'libmonetra.dll';
procedure M_DestroyEngine(); stdcall; external 'libmonetra.dll';
function M_ResponseParam(var conn: M_CONN; id: M_UINTPTR; key: PChar): PChar; stdcall; external
'libmonetra.dll';

function Int2Str(Number : Int64) : String;
var Minus : Boolean;
begin
        {
        SysUtils is not in the Uses clause so I can
        not use IntToStr() so I have to
        define an Int2Str function here
        }
        Result := '';
        if Number = 0 then
                Result := '0';
        Minus := Number < 0;
        if Minus then
                Number := -Number;
        while Number > 0 do
        begin
```

```
                Result := Char((Number mod 10) + Integer('0')) + Result;
                Number := Number div 10;
        end;
        if Minus then
                Result := '-' + Result;
end;

begin // main program begin
        M_InitEngine(nil);
        M_InitConn(conn);
        M_SetIP(conn, PChar(host), port);
        M_SetBlocking(conn, 1);
        Writeln('Connecting to ' + host + ':' + Int2Str(port) + '...');
        if not M_Connect(conn) then
        begin
                Writeln('Could not connect to ' + host + ':' +
                                Int2Str(port));
                M_DestroyConn(conn);
                M_DestroyEngine();
                Halt;
        end;
        id := M_TransNew(conn);

        M_TransKeyVal(conn, id, PChar('username'), PChar('vitale'));
        M_TransKeyVal(conn, id, PChar('password'), PChar('test'));
        M_TransKeyVal(conn, id, PChar('action'), PChar('sale'));
        M_TransKeyVal(conn, id, PChar('account'),
                        PChar('4012888888881'));
        M_TransKeyVal(conn, id, PChar('expdate'), PChar('0512'));
        M_TransKeyVal(conn, id, PChar('amount'), PChar('12.00'));

        Writeln('Running Transaction...');

        if not M_TransSend(conn, id) then
        begin
                Writeln('Connectivity Error: ' + M_ConnectionError(conn));
                M_DestroyConn(conn);
                M_DestroyEngine();
                Halt;
        end;

        Writeln('Code    : ' + M_ResponseParam(conn, id,
                        PChar('code')));
        Writeln('Verbiage: ' + M_ResponseParam(conn, id,
                        PChar('verbiage')));
        Writeln('Item    : ' + M_ResponseParam(conn, id,
                        PChar('item')));
        Writeln('Batch   : ' + M_ResponseParam(conn, id,
                        PChar('batch')));
        Writeln('TTID    : ' + M_ResponseParam(conn, id,
                        PChar('ttid')));
        M_DestroyConn(conn);
        M_DestroyEngine();
        Halt;
end.
```

A.2 Microsoft Visual C# .Net

Visual C# .Net can call libmonetra directly utilizing the InteropServices. Note each function must be prototyped before use, and since libmonetra is a C library, it must be marked as 'unsafe', simply stating that improper use can cause unexpected behavior. Most types map closely to their C counterparts. This program has been tested with Visual Studio .Net 2003 and 2005.

To test this code, you will need to create a new console application, using the contents below as the code. Please note we provide a convenient class you may use in your own code at the top of this example. If you would like the most recent example source, please download the libmonetra 32bit or 64bit binaries and look in the examples directory of the archive.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;

public class Monetra
{
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_InitEngine(string cafile);
        [DllImport("libmonetra.dll")]
                unsafe public static extern void M_DestroyEngine();
        [DllImport("libmonetra.dll")]
                unsafe public static extern void M_InitConn(IntPtr *conn);
        [DllImport("libmonetra.dll")]
                unsafe public static extern void M_DestroyConn(IntPtr* conn);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_SetIP(IntPtr* conn, string host, ushort port);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_SetSSL(IntPtr* conn, string host, ushort port);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_SetSSL_CAfile(IntPtr* conn, string path);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_SetSSL_Files(IntPtr* conn, string sslkeyfile,
                        string sslcertfile);
        [DllImport("libmonetra.dll")]
                unsafe public static extern void M_VerifySSLCert(IntPtr* conn, int tf);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_SetBlocking(IntPtr *conn, int tf);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_Connect(IntPtr *conn);
        [DllImport("libmonetra.dll")]
                unsafe public static extern IntPtr M_TransNew(IntPtr *conn);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_TransKeyVal(IntPtr *conn, IntPtr id,
                        string key, string val);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_TransSend(IntPtr *conn, IntPtr id);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_Monitor(IntPtr *conn);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_CheckStatus(IntPtr *conn, IntPtr id);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_ReturnStatus(IntPtr *conn, IntPtr id);
        [DllImport("libmonetra.dll")]
                unsafe public static extern void M_DeleteTrans(IntPtr *conn, IntPtr id);
        [DllImport("libmonetra.dll", EntryPoint="M_ResponseParam")]
                unsafe public static extern IntPtr M_ResponseParam_int(IntPtr *conn, IntPtr id,
                        string key);
        [DllImport("libmonetra.dll", EntryPoint="M_ConnectionError")]
                unsafe public static extern IntPtr M_ConnectionError_int(IntPtr *conn);
        [DllImport("libmonetra.dll")]
                unsafe public static extern IntPtr M_ResponseKeys(IntPtr *conn, IntPtr id,
                        int *num_keys);
        [DllImport("libmonetra.dll", EntryPoint="M_ResponseKeys_index")]
                unsafe public static extern IntPtr M_ResponseKeys_index_int(IntPtr keys,
```

```
                int num_keys, int idx);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_FreeResponseKeys(IntPtr keys, int num_keys);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_IsCommaDelimited(IntPtr *conn, IntPtr id);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_ParseCommaDelimited(IntPtr *conn, IntPtr id);
        [DllImport("libmonetra.dll", EntryPoint="M_GetCell")]
                unsafe public static extern IntPtr M_GetCell_int(IntPtr *conn, IntPtr id,
                        string column, int row);
        [DllImport("libmonetra.dll", EntryPoint="M_GetCellByNum")]
                unsafe public static extern IntPtr M_GetCellByNum_int(IntPtr *conn, IntPtr id,
                        int column, int row);
        [DllImport("libmonetra.dll", EntryPoint="M_GetHeader")]
                unsafe public static extern IntPtr M_GetHeader_int(IntPtr *conn, IntPtr id,
                        int column);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_NumRows(IntPtr *conn, IntPtr id);
        [DllImport("libmonetra.dll")]
                unsafe public static extern int M_NumColumns(IntPtr *conn, IntPtr id);

        unsafe public static string M_ResponseParam(IntPtr *conn, IntPtr id, string key)
        {
                return (Marshal.PtrToStringAnsi(M_ResponseParam_int(conn, id, key)));
        }
        unsafe public static string M_ConnectionError(IntPtr *conn)
        {
                return (Marshal.PtrToStringAnsi(M_ConnectionError_int(conn)));
        }
        unsafe public static string M_ResponseKeys_index(IntPtr keys, int num_keys, int idx)
        {
                return (Marshal.PtrToStringAnsi(M_ResponseKeys_index_int(keys, num_keys, idx)));
        }
        unsafe public static string M_GetCell(IntPtr *conn, IntPtr id, string column, int row)
        {
                return(Marshal.PtrToStringAnsi(M_GetCell_int(conn, id, column, row)));
        }
        unsafe public static string M_GetCellByNum(IntPtr *conn, IntPtr id, int column, int row)
        {
                return (Marshal.PtrToStringAnsi(M_GetCellByNum_int(conn, id, column, row)));
        }
        unsafe public static string M_GetHeader(IntPtr *conn, IntPtr id, int column)
        {
                return (Marshal.PtrToStringAnsi(M_GetHeader_int(conn, id, column)));
        }

        /* M_ReturnStatus codes */
        public const int M_SUCCESS       = 1;
        public const int M_FAIL          = 0;
        public const int M_ERROR  = -1;

        /* M_CheckStatus codes */
        public const int M_DONE          = 2;
        public const int M_PENDING       = 1;
}

namespace libmonetra_csharp
{
        public class Program
        {
                unsafe static int RunTrans(IntPtr *conn)
                {
                        int retval;
                        IntPtr id;
                        IntPtr keys;
                        string key;
                        int num_keys;
                        int i;

                        Console.WriteLine("Running transaction...");
                        id = Monetra.M_TransNew(conn);
                        Monetra.M_TransKeyVal(conn, id, "username", "vitale");
                        Monetra.M_TransKeyVal(conn, id, "password", "test");
                        Monetra.M_TransKeyVal(conn, id, "action", "sale");
                        Monetra.M_TransKeyVal(conn, id, "account", "4012888888881");
                        Monetra.M_TransKeyVal(conn, id, "expdate", "0512");
```

```
                Monetra.M_TransKeyVal(conn, id, "amount", "12.00");
                if (Monetra.M_TransSend(conn, id) == 0) {
                        Console.WriteLine("Failed to send trans: " +
                                Monetra.M_ConnectionError(conn));
                        return(1);
                }
                retval = Monetra.M_ReturnStatus(conn, id);
                if (retval != Monetra.M_SUCCESS) {
                        Console.WriteLine("Bad return status: " + retval);
                }
                /* print results ... */
                Console.WriteLine("code: " + Monetra.M_ResponseParam(conn, id, "code"));
                Console.WriteLine("verbiage: " +
                        Monetra.M_ResponseParam(conn, id, "verbiage"));

                keys = Monetra.M_ResponseKeys(conn, id, &num_keys);
                Console.WriteLine("All " + num_keys + " repsonse parameters:");
                for (i=0; i<num_keys; i++) {
                        key = Monetra.M_ResponseKeys_index(keys, num_keys, i);
                        Console.WriteLine(key + "=" +
                                Monetra.M_ResponseParam(conn, id, key));
                }
                Monetra.M_FreeResponseKeys(keys, num_keys);
                Monetra.M_DeleteTrans(conn, id);
                Console.WriteLine("Transaction Done");
                return(0);
        }

        unsafe static int RunReport(IntPtr *conn)
        {
                IntPtr id;
                int retval;
                int rows, cols, i, j;
                string line;

                Console.WriteLine("Running report....");
                id = Monetra.M_TransNew(conn);
                Monetra.M_TransKeyVal(conn, id, "username", "vitale");
                Monetra.M_TransKeyVal(conn, id, "password", "test");
                Monetra.M_TransKeyVal(conn, id, "action", "admin");
                Monetra.M_TransKeyVal(conn, id, "admin", "gut");
                if (Monetra.M_TransSend(conn, id) == 0) {
                        Console.WriteLine("Failed to send trans: " +
                                Monetra.M_ConnectionError(conn));
                        return(1);
                }
                retval = Monetra.M_ReturnStatus(conn, id);
                if (retval != Monetra.M_SUCCESS) {
                        Console.WriteLine("Bad return status: " + retval);
                }
                Monetra.M_ParseCommaDelimited(conn, id);

                rows = Monetra.M_NumRows(conn, id);
                cols = Monetra.M_NumColumns(conn, id);

                Console.WriteLine("Report Data (" + rows + " rows, " + cols + " cols):");
                line = "";
                for (i=0; i<cols; i++) {
                        if (i != 0) line = line + "|";
                        line = line + Monetra.M_GetHeader(conn, id, i);
                }
                Console.WriteLine(line);
                for (i=0; i<rows; i++) {
                        line = "";
                        for (j=0; j<cols; j++) {
                                if (j != 0) line = line + "|";
                                line = line + Monetra.M_GetCellByNum(conn, id, j, i);
                        }
                        Console.WriteLine(line);
                }
                Monetra.M_DeleteTrans(conn, id);
                Console.WriteLine("Report Done");
                return (0);
        }

        unsafe static int Main(string[] args)
```

```
                {
                        IntPtr conn;

                        Monetra.M_InitEngine(null);
                        Monetra.M_InitConn(&conn);
                        Monetra.M_SetSSL(&conn, "testbox.monetra.com", 8444);
                        Monetra.M_SetBlocking(&conn, 1);
                        if (Monetra.M_Connect(&conn) == 0) {
                                Console.WriteLine("Failed to connect: " +
                                        Monetra.M_ConnectionError(&conn));
                                return(1);
                        }

                        RunTrans(&conn);
                        RunReport(&conn);

                        Monetra.M_DestroyConn(&conn);
                        Monetra.M_DestroyEngine();
                        return (0);
                }
        }
}
```

A.3 Microsoft Visual Basic .Net

Below is an example of how to call libmonetra.dll from a Visual Basic .Net application. Please note this will NOT work with Visual Basic 6 or lower, as libmonetra is a _cdecl DLL. Please reference the C prototypes above to see how they map to Visual Basic data types. You will find that each function must be prototyped for Visual Basic before it can be used. The example below prototypes the primary libmonetra functions and shows you how to use them. You will notice the class at the top of the example can be easily copied into your own project and be used.

A special note about Visual Basic .Net and string handling. You will notice that '.ToInt32' is called to check for 0 (aka NULL) before converting the IntPtr to a string via marshalling. If this is not done, an exception will be raised about a NULL pointer dereference. We have wrapped the functions which return ANSI C Strings to show you the proper way to accomplish this. The wrapped functions return Strings that are usable anywhere within VB.

To test this code, you will need to create a new Visual Basic console application, using the code below. This has been tested with Visual Basic .Net 2003 and 2005.

```
Option Explicit On
Option Strict On
Imports System
Imports System.String
Imports System.Runtime.InteropServices

Public Class Monetra
    Declare Ansi Function M_InitEngine Lib "libmonetra.dll" (ByVal cafile As String) As Integer
    Declare Ansi Sub M_DestroyEngine Lib "libmonetra.dll" ()
    Declare Ansi Sub M_InitConn Lib "libmonetra.dll" (ByRef conn As IntPtr)
    Declare Ansi Sub M_DestroyConn Lib "libmonetra.dll" (ByRef conn As IntPtr)
    Declare Ansi Function M_SetIP Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal host As String, ByVal port As Short) As Integer
    Declare Ansi Function M_SetSSL Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal host As String, ByVal port As Short) As Integer
    Declare Ansi Function M_SetSSL_CAfile Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal path As String) As Integer
    Declare Ansi Function M_SetSSL_Files Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal sslkeyfile As String, ByVal sslcertfile As String) As Integer
    Declare Ansi Function M_VerifySSLCert Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal tf As Integer) As Integer
    Declare Ansi Function M_SetBlocking Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal tf As Integer) As Integer
    Declare Ansi Function M_Connect Lib "libmonetra.dll" (ByRef conn As IntPtr) As Integer
    Declare Ansi Function M_TransNew Lib "libmonetra.dll" (ByRef conn As IntPtr) As IntPtr
    Declare Ansi Function M_TransKeyVal Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal id As IntPtr, ByVal key As String, ByVal val As String) As Integer
    Declare Ansi Function M_TransSend Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal id As IntPtr) As Integer
    Declare Ansi Function M_Monitor Lib "libmonetra.dll" (ByRef conn As IntPtr) As Integer
    Declare Ansi Function M_CheckStatus Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal id As IntPtr) As Integer
    Declare Ansi Function M_ReturnStatus Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal id As IntPtr) As Integer
    Declare Ansi Function M_DeleteTrans Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal id As IntPtr) As Integer
    ' map to different name so we can wrap it
    Declare Ansi Function M_ResponseParam_int Lib "libmonetra.dll" Alias "M_ResponseParam" _
        (ByRef conn As IntPtr, ByVal id As IntPtr, ByVal key As String) As IntPtr
    ' map to different name so we can wrap it
    Declare Ansi Function M_ConnectionError_int Lib "libmonetra.dll" Alias "M_ConnectionError" _
        (ByRef conn As IntPtr) As IntPtr
    Declare Ansi Function M_ResponseKeys Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal id As IntPtr, ByRef num_keys As Integer) As IntPtr
    ' map to different name so we can wrap it
```

```
    Declare Ansi Function M_ResponseKeys_index_int Lib "libmonetra.dll" Alias _
        "M_ResponseKeys_index"  (ByVal keys As IntPtr, ByVal num_keys As Integer, _
        ByVal idx As Integer) As IntPtr
    Declare Ansi Function M_FreeResponseKeys Lib "libmonetra.dll" (ByVal keys As IntPtr, _
        ByVal num_keys As Integer) As Integer
    Declare Ansi Function M_IsCommaDelimited Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal id As IntPtr) As Integer
    Declare Ansi Function M_ParseCommaDelimited Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal id As IntPtr) As Integer
    ' map to different name so we can wrap it
    Declare Ansi Function M_GetCell_int Lib "libmonetra.dll" Alias "M_GetCell" (_
        ByRef conn As IntPtr, ByVal id As IntPtr, ByVal column As String, ByVal row As Integer) _
        As IntPtr
    Declare Ansi Function M_GetCellByNum_int Lib "libmonetra.dll" Alias "M_GetCellByNum" ( _
        ByRef conn As IntPtr, ByVal id As IntPtr, ByVal column As Integer, ByVal row As Integer) _
        As IntPtr
    Declare Ansi Function M_GetHeader_int Lib "libmonetra.dll" Alias "M_GetHeader" ( _
        ByRef conn As IntPtr, ByVal id As IntPtr, ByVal column As Integer) As IntPtr
    Declare Ansi Function M_NumRows Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal id As IntPtr) As Integer
    Declare Ansi Function M_NumColumns Lib "libmonetra.dll" (ByRef conn As IntPtr, _
        ByVal id As IntPtr) As Integer


    ' Wrap this function to check for a NULL pointer being
    ' returned to avoid an exception
    Public Shared Function M_GetCell(ByRef conn As IntPtr, ByVal id As IntPtr, _
        ByVal column As String, ByVal row As Integer) As String
        Dim ret As IntPtr
        ret = M_GetCell_int(conn, id, column, row)
        If ret.ToInt64 = 0 Then
            Return ""
        End If
        Return Marshal.PtrToStringAnsi(ret)
    End Function

    ' Wrap this function to check for a NULL pointer being
    ' returned to avoid an exception
    Public Shared Function M_GetCellbyNum(ByRef conn As IntPtr, ByVal id As IntPtr, _
        ByVal column As Integer, ByVal row As Integer) As String
        Dim ret As IntPtr
        ret = M_GetCellByNum_int(conn, id, column, row)
        If ret.ToInt64 = 0 Then
            Return ""
        End If
        Return Marshal.PtrToStringAnsi(ret)
    End Function

    ' Wrap this function to check for a NULL pointer being
    ' returned to avoid an exception
    Public Shared Function M_GetHeader(ByRef conn As IntPtr, ByVal id As IntPtr, _
        ByVal column As Integer) As String
        Dim ret As IntPtr
        ret = M_GetHeader_int(conn, id, column)
        If ret.ToInt64 = 0 Then
            Return ""
        End If
        Return Marshal.PtrToStringAnsi(ret)
    End Function

    ' Wrap this function to check for a NULL pointer being
    ' returned to avoid an exception
    Public Shared Function M_ResponseKeys_index(ByVal keys As IntPtr, _
        ByVal num_keys As Integer, ByVal idx As Integer) As String
        Dim ret As IntPtr
        ret = M_ResponseKeys_index_int(keys, num_keys, idx)
        If ret.ToInt64 = 0 Then
            Return ""
        End If
        Return Marshal.PtrToStringAnsi(ret)
    End Function

    ' Wrap this function to check for a NULL pointer being
    ' returned to avoid an exception
    Public Shared Function M_ResponseParam(ByRef conn As IntPtr, ByVal id As IntPtr, _
        ByVal key As String) As String
```

```
        Dim ret As IntPtr
        ret = M_ResponseParam_int(conn, id, key)
        If ret.ToInt64 = 0 Then
            Return ""
        End If
        Return Marshal.PtrToStringAnsi(ret)
    End Function
    ' Wrap this function to check for a NULL pointer being
    ' returned to avoid an exception
    Public Shared Function M_ConnectionError(ByRef conn As IntPtr) As String
        Dim ret As IntPtr
        ret = M_ConnectionError_int(conn)
        If ret.ToInt64 = 0 Then
            Return ""
        End If
        Return Marshal.PtrToStringAnsi(ret)
    End Function
    ' M_ReturnStatus returns
    Public Const M_SUCCESS As Integer = 1
    Public Const M_FAIL As Integer = 0
    Public Const M_ERROR As Integer = -1
    ' M_CheckStatus returns
    Public Const M_DONE As Integer = 2
    Public Const M_PENDING As Integer = 1
End Class

Module Module1
    Sub RunTrans(ByVal conn As IntPtr)
        Dim retval As Integer
        Dim id As IntPtr
        Dim i As Integer
        Dim key As String
        Dim num_keys As Integer
        Dim keys As IntPtr
        Console.WriteLine("Running transaction....")
        id = Monetra.M_TransNew(conn)
        Monetra.M_TransKeyVal(conn, id, "username", "vitale")
        Monetra.M_TransKeyVal(conn, id, "password", "test")
        Monetra.M_TransKeyVal(conn, id, "action", "sale")
        Monetra.M_TransKeyVal(conn, id, "account", "4012888888881")
        Monetra.M_TransKeyVal(conn, id, "expdate", "0512")
        Monetra.M_TransKeyVal(conn, id, "amount", "12.00")
        If Monetra.M_TransSend(conn, id) = 0 Then
            Console.WriteLine("Failed to send trans")
            Return
        End If
        retval = Monetra.M_ReturnStatus(conn, id)
        If retval <> Monetra.M_SUCCESS Then
            Console.WriteLine("Bad return status: " + retval.ToString())
        End If
        Console.WriteLine("code: " + Monetra.M_ResponseParam(conn, id, "code"))
        Console.WriteLine("verbiage: " + Monetra.M_ResponseParam(conn, id, "verbiage"))
        keys = Monetra.M_ResponseKeys(conn, id, num_keys)
        Console.WriteLine("All " + num_keys.ToString() + " response parameters:")
        For i = 0 To num_keys - 1
            key = Monetra.M_ResponseKeys_index(keys, num_keys, i)
            Console.WriteLine(key + "=" + Monetra.M_ResponseParam(conn, id, key))
        Next
        Monetra.M_FreeResponseKeys(keys, num_keys)
        Monetra.M_DeleteTrans(conn, id)
        Console.WriteLine("Transaction Done")
    End Sub

    Sub RunReport(ByVal conn As IntPtr)
        Dim id As IntPtr
        Dim retval As Integer
        Dim rows As Integer
        Dim cols As Integer
        Dim i As Integer
        Dim j As Integer
        Dim line As String

        Console.WriteLine("Running report....")
        id = Monetra.M_TransNew(conn)
        Monetra.M_TransKeyVal(conn, id, "username", "vitale")
        Monetra.M_TransKeyVal(conn, id, "password", "test")
```

```
        Monetra.M_TransKeyVal(conn, id, "action", "admin")
        Monetra.M_TransKeyVal(conn, id, "admin", "gut")
        If Monetra.M_TransSend(conn, id) = 0 Then
            Console.WriteLine("Failed to send trans")
            Return
        End If
        retval = Monetra.M_ReturnStatus(conn, id)
        If retval <> Monetra.M_SUCCESS Then
            Console.WriteLine("Bad return status: " + retval.ToString())
        End If
        Monetra.M_ParseCommaDelimited(conn, id)

        rows = Monetra.M_NumRows(conn, id)
        cols = Monetra.M_NumColumns(conn, id)

        Console.WriteLine("Report Data (" + rows.ToString() + " rows, " + cols.ToString() + _
                " cols):")

        line = ""
        For i = 0 To cols - 1
            If Not i = 0 Then
                line = line + "|"
            End If
            line = line + Monetra.M_GetHeader(conn, id, i)
        Next
        Console.WriteLine(line)
        For i = 0 To rows - 1
            line = ""
            For j = 0 To cols - 1
                If Not j = 0 Then
                    line = line + "|"
                End If
                line = line + Monetra.M_GetCellbyNum(conn, id, j, i)
            Next
            Console.WriteLine(line)
        Next

        Monetra.M_DeleteTrans(conn, id)
        Console.WriteLine("Report Done")
    End Sub

    Public Sub Main()
        Dim conn As IntPtr

        Monetra.M_InitEngine("")
        Monetra.M_InitConn(conn)
        Monetra.M_SetSSL(conn, "testbox.monetra.com", 8444)
        Monetra.M_SetBlocking(conn, 1)
        If Monetra.M_Connect(conn) = 0 Then
            Console.WriteLine("M_Connect() failed" + Monetra.M_ConnectionError(conn))
            Return
        End If

        RunTrans(conn)
        RunReport(conn)

        Monetra.M_DestroyConn(conn)
        Monetra.M_DestroyEngine()
        Return
    End Sub

End Module
```