# Programming Guide
# for Moxa Device Manager

**First Edition, December 2010**

**www.moxa.com/product**

# Programming Guide
# for Moxa Device Manager



## Copyright Notice



## Trademarks



## Disclaimer

Information in this document is subject to change without notice and does not represent a commitment on the part of Moxa.

Moxa provides this document as is, without warranty of any kind, either expressed or implied, including, but not limited to, its particular purpose. Moxa reserves the right to make improvements and/or changes to this manual, or to the products and/or the programs described in this manual, at any time.

Information provided in this manual is intended to be accurate and reliable. However, Moxa assumes no responsibility for its use, or for any infringements on the rights of third parties that may result from its use.

This product might include unintentional technical or typographical errors. Changes are periodically made to the information herein to correct such errors, and these changes are incorporated into new editions of the publication.

## Technical Support Contact Information

**www.moxa.com/support**

**Moxa Americas**
Toll-free: 1-888-669-2872
Tel: +1-714-528-6777
Fax: +1-714-528-6778

**Moxa China (Shanghai office)**
Toll-free: 800-820-5036
Tel: +86-21-5258-9955
Fax: +86-21-5258-5505

**Moxa Europe**
Tel: +49-89-3 70 03 99-0
Fax: +49-89-3 70 03 99-99

**Moxa Asia-Pacific**
Tel: +886-2-8919-1230
Fax: +886-2-8919-1231

# Table of Contents

# 1

# Introduction

The following topics are covered in this chapter:

- **Introduction**
- **System Architecture**
  - 3-tier System
  - 2-tier System
- **Software Module Design**
- **Message Structure**
- **MDM Message Definition**
- **Runtime Installation**
- **Runtime Operations**
- **Runtime Configuration**
- **MDM Development**

# Introduction

Moxa Device Manager (MDM for short) is an easy-to-use remote management tool for managing Moxa's ready-to-run embedded computers on the Internet. Moxa's embedded computers make excellent front-end computers at remote sites for on-site data collection and industrial control applications. MDM is designed to make it easy for system administrators to manage their remote embedded computers. One of the key benefits of MDM is that management tasks, such as configuring the network, managing and/or transmitting text and binary files, and monitoring and controlling processes, can be handled easily using a Windows-based user interface. In addition, MDM can be used to manage different embedded computer models and embedded computers that use different operating systems, all from one centrally located computer. As long as the individual embedded computers all have MDM Agent installed, they can be recognized and managed by the unified MDM Client onyour PC. MDM's features give system integrators an efficient tool for handling all remote devices from one computer.

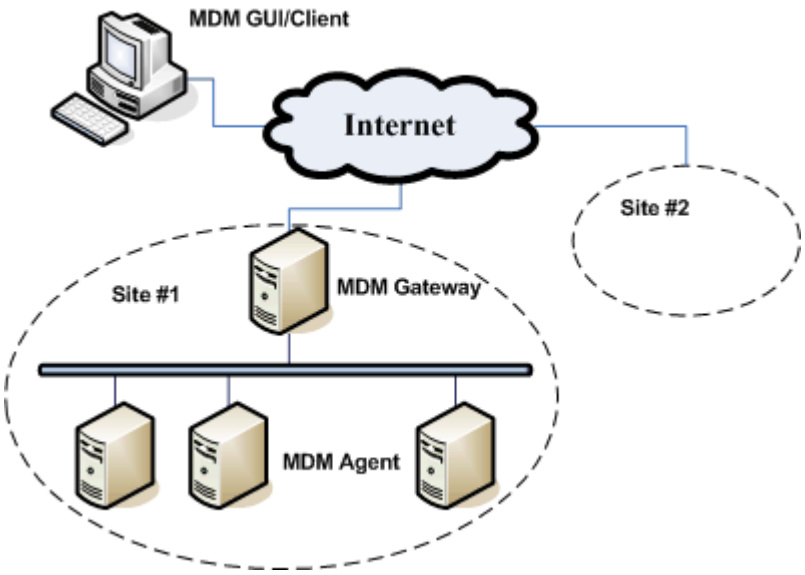
This version of MDM further provides a framework to help developers integrate their own programs with the MDM API to manage embedded computers with dynamically loaded functions.

# System Architecture

An MDM system is constructed by a set of MDM programs. An MDM Gateway acts as a TCP server as well as a UDP client. The MDM Client and a number of MDM Agents are TCP clients and UDP servers to the MDM Gateway. The MDM client runs at the central site and the MDM Agents run at the distributed sites. There are two different network deployment options to choose from, according to your network circumstances.

## 3-tier System

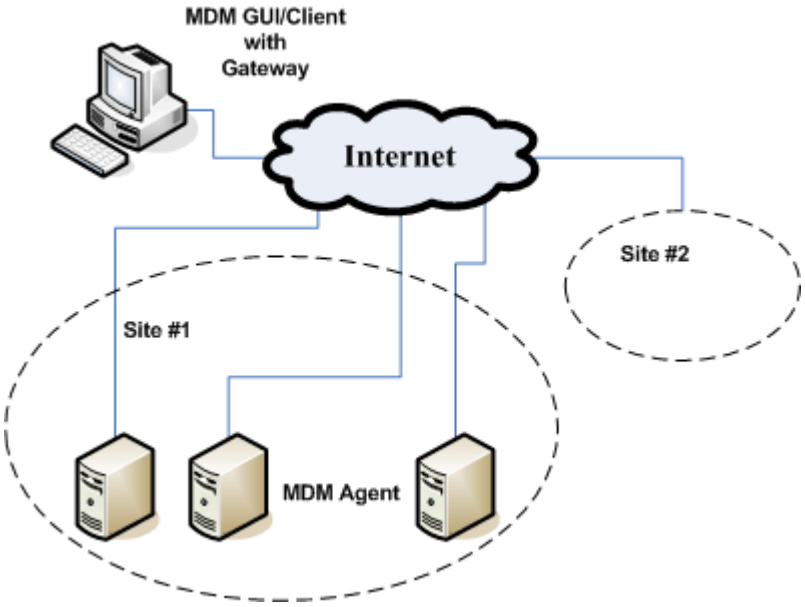
If your distributed or remote site has an internet-accessible Windows PC, you can use that PC to run MDM Gateway. In addition, install MDM Agent onto each Moxa computer which is in the same LAN as that PC. This deployment architecture, called a 3-tier system, is illustrated in the following figure. The advantage of this architecture is that none of the embedded computers in the site need internet accessibility. The MDM Client in the central site Windows PC communicates with the MDM Gateway.




Within the LAN at the remote site, MDM Gateway periodically sends a UDP broadcast message of its location to auto-discover MDM Agents in the embedded computers. The MDM Agents receive the UDP broadcast message and then make a TCP connection to the MDM Gateway for further message transmission.

## 2-tier System

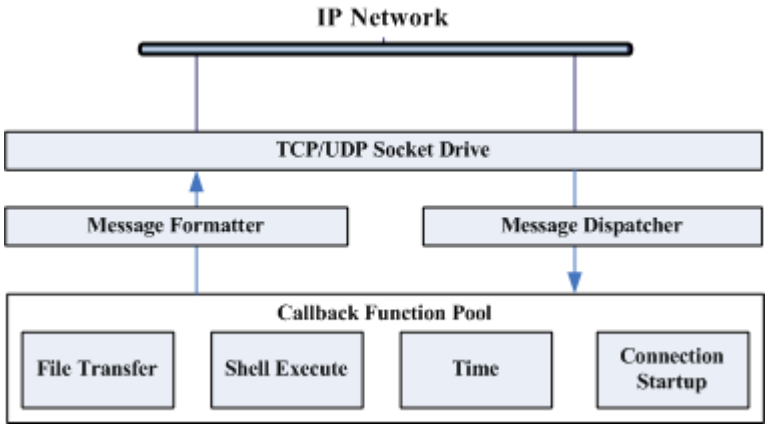
An alternative architecture is a 2-tier system, illustrated in the following figure. In this architecture, each of computers at the remote site must be connected to the internet. Each computer runs the MDM Agent program, and the MDM Gateway GUI/Client at the central site Windows PC communicates with them through the internet.




All MDM Agents need to know the location of the MDM Gateway. That is to say, at each remote computer, you need to specify the IP address and the TCP listening port of the MDM Gateway in the configuration file.

# Software Module Design

The following figure shows the software module design of the MDM programs.




For any MDM program, a dispatcher is implemented within an infinite loop. It manages/handles all established TCP/UDP connections and those data packets received from these connections. The dispatcher digests one message at a time. By checking a pool of managed callback functions, it calls the associated function to handle the message.

MDM also includes an application programming interface (API) for programmers to format messages to be transmitted over TCP/UDP connections.

# Message Structure

MDM programs are driven by messages over UDP/TCP connections. A message is used when one MDM computer needs to communicate with another. An MDM message contains at least two fields: the message tag and data field. The message tag field indentifies the message and is a positive number. The data field contains data for the associated message ID.

# MDM Message Definition

The MDM design has defined message protocols to support all features. For details, please refer to Appendix A: MDM Protocol. When programming for MDM, you need to use MDM protocols and the symbolic constants defined in the header file *mdm_message.h*.

# Runtime Installation

The MDM program on each embedded computer can be either an MDM Agent or MDM Gateway. For your specific product model, download the compressed MDM program from Moxa web site and upload it to the embedded computer. Then select a directory such /home in Linux and C:\ in Windows XPe and uncompress the compressed MDM program into the directory.

In Linux, the MDM working directory *mdm* includes the following files:

| | |
|---|---|
| *mdm* | working directory |
| *dll* | DLL directory |
| *libfmngr.so* | File Manager shared library |
| *liblaunch.so* | Auto-launch shared library |
| *libnet.so* | Network Management shared library |
| *libproc.so* | Process Management shared library |
| *libshell.so* | Shell Execution shared library |
| *libsystem.so* | System Information shared library |
| *libtime.so* | RTC and System Time shared library |
| *libmdm.so* | MDM API shared library |
| *mdm-main* | MDM main program using MDM API shared library |
| *mdm_start.sh* | shell script to start the MDM program |
| *mdm_stop.sh* | shell script to stop the MDM program |

In Windows, the MDM working folder *mdm* includes the following files:

| | |
|---|---|
| *mdm* | working directory |
| *dll* | DLL directory |
| *fmngr.dll* | File Manager DLL |
| *launch.dll* | Auto-launch DLL |
| *net.dll* | Network Management DLL |
| *proc.dll* | Process Management DLL |
| *shell.dll* | Shell Execution DLL |
| *system.dll* | System Information DLL |
| *time.dll* | RTC and System Time DLL |
| *mdm.dll* | MDM API DLL |
| *mdm-main.exe* | MDM main program using MDM API DLL |

Under the MDM working directory *mdm*, there are an MDM API Linux shared library (*libmdm.so*) or Windows dynamic-link library (*mdm.dll*), an MDM main program for MDM Agent or MDM Gateway, and a subdirectory *dll* where seven Moxa featured dynamic-link libraries or shared libraries are provided. The dynamic-link libraries or shared libraries under *dll* are loaded into the memory space of the MDM Gateway or Agent when it is started.

Follow these steps to add auto-launch support for MDM Agent in Windows XPe,

1. Execute Autolaunch.exe to install the AutoLaunch service.
2. Start the AutoLaunch service immediately (**Start menu → Control Panel → Administrative Tools → Services → Auto Launch**), or start AutoLaunch the next time Windows boots up.

# Runtime Operations

To start an MDM Agent, run the MDM main program in Windows and *mdm_start.sh* in Linux with or without the *-a* argument. To stop it, kill its process in Windows and run *mdm_stop.sh* in Linux. The Windows MDM main program is a console program and it may be stopped gracefully by pressing the Ctrl and Break keys at the same time.

To start an MDM Gateway, run the MDM main program in Windows and *mdm_start.sh* in Linux with the *-g* argument. To stop it, kill its process in Windows and run *mdm_stop.sh* in Linux.

To start an MDM Gateway and MDM Agent simultaneously, run the MDM main program in Windows and *mdm_start.sh* in Linux with *-g* and *-a* arguments. To stop it, kill its process in Windows and run *mdm_stop.sh* in Linux.

# Runtime Configuration

By default, the MDM configuration files is not in the release package and MDM operates using default values. The MDM configuration file must be named *config*. If the configuration file *config* is needed, it should be in the MDM working directory *mdm*. This file defines system parameters used by the MDM programs. These parameters are presented in a "key = value" format. Examples are as follows.

```
# For MDM Gateway, default as 16888
tcp_server_port = 1234
# For MDM Gateway, default as 16889
udp_server_port = 1235
# For MDM Agent or GUI/client, specifically assign the location of the gateway
gateway_hosts = tcp:201.63.45.2:1234
```

The program must restart for any changes to take effect. For the system architecture where MDM Gateway and MDM Agent are cross-Internet, the *gateway_hosts* parameter must be specified in the configuration file in order to let MDM Agent know the location of the MDM Gateway.

# MDM Development

MDM development files are used to develop MDM Client programs.

In linux, the MDM API directory *mdm_api* includes the following files:

| | |
|---|---|
| *mdm_api* | MDM API directory |
| *inc* | header file directory |
| *mdm_api.h* | MDM API header file |
| *mdm_message.h* | MDM message definition file |
| *lib* | library directory |
| *libmdm.so* | MDM API shared library |
| *mdm-test* | MDM Client demo program source directory |

In windows, the MDM API folder *mdm_api* includes the following files:

| | |
|---|---|
| *mdm_api* | MDM API folder |
| *inc* | header file folder |
| *mdm_api.h* | MDM API header file |
| *mdm_message.h* | MDM message definition file |
| *lib* | library folder |
| *mdm.dll* | MDM API DLL |
| *mdm.lib* | MDM API LIB |
| *mdm-test* | MDM Client demo program source folder |

You can refer to files in the *mdm-test* directory to help write your own MDM Client programs.

# 2

# MDM Programming

This chapter describes the procedure of initializing an MDM program and the use of APIs and data structures in the C language. This chapter will also illustrate how to program callback functions and how to handle messages.

The following topics are covered in this chapter:

- **Initializing An MDM Program**
- **Connections and Users**
- **Sending Messages**
- **Splitting a Formatted Message**
- **Programming a GUI Tool**

# Initializing An MDM Program

To initialize an MDM program, call function *mdm_start* with specifying a program type for MDM Client, MDM Gateway, or MDM Agent.

```
int mdm_start(unsigned int type, unsigned int thread);
#define PROGRAM_TYPE_DEVICE          1          /* MDM Agent */
#define PROGRAM_TYPE_GATEWAY   (1<<1)   /* MDM Gateway */
#define PROGRAM_TYPE_CLIENT    (1<<2)   /* MDM Client */
```

For example, use *mdm_start* to call one of the following:

```
mdm_start(PROGRAM_TYPE_DEVICE, 0);
mdm_start(PROGRAM_TYPE_GATEWAY, 0);
mdm_start(PROGRAM_TYPE_CLIENT, 1);
```

Note that when using this function, the MDM client and MDM gateway can be started together, as in the example below:

```
mdm_start(PROGRAM_TYPE_GATEWAY | PROGRAM_TYPE_CLIENT, 1);
```

Use argument *thread* to initialize MDM in thread mode. That means you can add your code and have one thread operating MDM.

To check the life of the MDM thread, please use the following API.

```
int mdm_alive(void);
```

To stop the MDM thread, call the following API.

```
void mdm_stop(void);
```

# Connections and Users

Each UDP/TCP connection in MDM is defined as follows.

```
typedef struct _MDMCONN      MDMCONN;
/* A connection could be a device, a gateway, or a client. Messages are transmitted over it. */
struct _MDMCONN
{
unsigned int type;          /* type of connection to the system */
      void  *hndl;          /* pointer to the low level connection */
MDMUSER *user;              /* what user is   */
char *peer;                 /* peer is talking */
void *data;                 /* private data, careful use */
};
```

Do not change the values of these elements: *type*, *hndl* and *user*. You can utilize the element *data* to carry meaningful private data.

Additionally, the following struct type is used to identify a connection. Element *nick* is a string with a format of ip:port, for example, 201.34.56.12:3476. It is the IP address followed by the local port of the connection.

```
/* An MDM program maintains a list of users. A user could be a device, a gateway, or a client. */
typedef struct _MDMUSER
{
      char *nick;               /* a system-wide yet unique name */
      char *pass;               /* not used */
      MDMCONN *con;             /* connection bound to the user */
      unsigned int flags;       /* reserved for future use */
} MDMUSER;
```

Do not change the values of the elements: *nick, pass* and *con*. You can utilize the element *flags* to carry meaningful private data.

# Sending Messages

Inside a callback function, you can utilize either of the following APIs to send a message to a connection.

```
void send_cmd (MDMCONN *con, unsigned int tag, const char *fmt, ...);
void send_peer(MDMCONN *con, char *peer, unsigned int tag, const char *fmt, ...);
void send_cmd_data (MDMCONN *con, unsigned int tag, const char *data, unsigned int len);
void send_peer_data(MDMCONN *con, char *peer, unsigned int tag, const char *data, unsigned int len);
```

An MDM program uses function *send_cmd* to send a formatted message over a one-hop connection to another MDM program, e.g., an MDM Client to an MDM Gateway and vice versa or an MDM Agent to an MDM Gateway and vice versa. The *send_cmd* function is formatted just like a *sprintf* function. When there is more than one data field in the message, remember to use a space as a delimiter. If there is a space or two in a data field, use a pair of double quotes on it.

When you transmit a formatted message, unless it is a file transfer message between an MDM Agent and an MDM Client, use *send_peer* to send it over two-hop connections. The message is first sent to the MDM Gateway which then forwards it to a named peer.

Function *send_cmd_data* is the non-formatted version of *send_cmd* while s*end_peer_data* is the non-formatted version of *send_peer.*

# Splitting a Formatted Message

When a formatted message arrives, your user-defined function can use the following API function to split the message into string fields:

```
int split_line (char **av, int max, char *pkt);
```

Or use the following API function to parse a string field at a time:

```
char* next_arg (char **pkt);
```

# Programming a GUI Tool

When you program an MDM Client, you can also build a GUI tool. Leave those messages to the tool. That means no callback functions are defined in the client. Messages for these functions will not be handled by the client. Instead, you define a function of the following type to handle them in with GUI components. An example of such a function would be:

```
typedef void (*mdm_message_t) (MDMCONN *con, int tag, int argc, char **argv);
```

In this function, the first argument contains information of connected MDM Gateway. The connected MDM Gateway session ID can be obtained from `con->user->nick`. The second argument denotes the message ID of the received message. The third argument denotes the number of fields in the received message for the field data in the fourth argument.

```
/* This function is provided by the GUI and is called when there is
message coming and is not intercepted by the client.
*/
static void
mngr_message (MDMCONN *con, int tag, int argc, char **argv)
{
    char *nick;
    int i;
```

```
        printf("mngr_message : [%d]", tag);
        for (i=0; i < argc; i++)
        printf(" %s", argv[i]);
        printf("\n");

        nick = argv[0];
        switch(tag)
        {
              /* periodically gets the information of a device: */
              case MSG_DEVICE_MEMORY_INFO: /* 3: <nick> <freeRAM> <totalRAM> */
              break;
        …
        }
…
}
```

To send messages to the MDM Gateway or MDM Agent outside this function, the MDM Gateway session ID and the associated MDMCONN argument value must be saved when message ID 16 is received on MDM Clients connected to the MDM Gateway. The session to MDM Gateway is useless when message ID 21 is received on MDM Clients disconnected from the MDM Gateway. Furthermore, in order to send messages to MDM Agent outside this function, the MDM Agent session ID must also be saved for the connected MDM Gateway when message ID 17 is received. The session ID of MDM Agent is useless when message ID 18 is received.

To make the function effective, call the following API *mdm_set_mngr_function* to add it before calling API *mdm_start* in the main routine.

```
void mdm_set_mngr_function(mdm_message_t msg);
…
mdm_set_mngr_function(mngr_message);
/* start MDM */
if (mdm_start(p_type, 1) != 0)
{
      printf("Failed to start MDM\n");
      return -2;
}
…
```

# 3

# Programming Examples

This chapter introduces several examples of programming for an MDM Client.

The following topics are covered in this chapter:

- **Uploading a File**
- **Executing an Executable**
- **A Test Client Program**
- **Three GUI MDM Client Programs**

# Uploading a File

By utilizing *send_cmd*, MDM allows a client program to upload a local file to an agent. The messages related to file transfer are defined in Appendix A. In your client program, call the following—

```
send_cmd(con, MSG_MXFTP_3TIER_CLI2DEV, "%s \"%s\" \"%s\" \"%s\"",
devNick, devDir, fName, cliDir);
```

—where *devNick* is the session ID of MDM Agent, *fName* is the name of the file being uploaded, *cliDir* is its source directory, and *devDir* is its destination directory in the embedded computer.

After initiating a file upload session, the client program continues receiving a progress status, i.e., MSG_MXFTP_UPLOAD_STATUS messages. When the session completes, the client program receives a MSG_MXFTP_UPLOAD_COMPLETE message.

# Executing an Executable

The MDM allows a client program to remotely execute an executable file in an embedded computer. Use the following code—

```
send_peer(con, MSG_SYSTEM_EXECUTE, "%s \"%s\" \"%s\"", nick, path, args);
```

—where *nick* is the session ID of MDM Agent, *path* is the path of an executable file, and *args* is its arguments.

# A Test Client Program

You can find a Windows Visual Studio project or Linux Makefile and the source code of a console program in the directory *mdm-test*. The *mdm-test* MDM Client sends MDM message requests from a file in a specific format and receives replies. It can be used to test all MDM messages.

# Three GUI MDM Client Programs

You can find three Windows Visual Studio projects and their source code in the directories *MDM-System*, *MDM-Proc* and *MDM-Shell*. The *MDM-System* program displays the system information of the selected device. The *MDM-Proc* program displays a list of running programs on the selected device and can stop a selected running program. The *MDM-Shell* program can run a non-interactive program in the selected device and get its output to display.

# A

# MDM Protocol

The MDM Protocol defines message specifications for each MDM message. There are two types of MDM protocols. The first are messages supported by MDM Core and the others are ones supported by seven prebuilt MDM Agent libraries. Both are described in the following sections. Each function of message specifications are described together in items of message type, message ID, message data and the description of arguments in message data. The message type of each message is request or reply. If the message ID for a message is in bold font, then it means multiple requests or replies could be sent or received. Message examples of each reply are also given. For each message request with the first data argument *devNick* except file transfer messages, use the function *send_peer* or *send_peer_data* to send requests. For other requests, use the function *send_cmd* or *send_cmd_data* to send them.

# Messages Supported by MDM Core

MDM Core provides six categories of messages.

**1. Unsolicited Messages**

When MDM Client is connected to MDM Gateway, there are five types of unsolicited messages.

A. MDM Client accepted by MDM Gateway notification. This message is the first message received by the MDM Client and sent from the MDM Gateway after connection to the MDM Gateway is established.

| **Type** | **ID** | **Data** | **Description** |
|---|---|---|---|
| Reply | 16 | gwLocation cliNick | gwLocation: connected MDM Gateway location in IP address and TCP port<br>cliNick: session ID of MDM Client accepted by connected MDM Gateway |

For example: 16 192.168.30.44:16888 192.168.30.44:2312

B. MDM Client disconnected from MDM Gateway notification. There is no data in the message.

| **Type** | **ID** | **Data** | **Description** |
|---|---|---|---|
| Reply | 21 | | |

C. MDM Agent session ID notification. After this message is received the MDM Client can manage the corresponding MDM Agent.

| **Type** | **ID** | **Data** | **Description** |
|---|---|---|---|
| Reply | 17 | DevNick | devNick: session ID of MDM Agent managed by connected MDM Gateway |

For example: 17 192.168.30.81:1029

D. MDM Agent disconnected notification.

| **Type** | **ID** | **Data** | **Description** |
|---|---|---|---|
| Reply | 18 | devNick | devNick: session ID of MDM Agent which is disconnected from connected MDM Gateway |

For example: 18 192.168.30.81:1029

E. Error replies sent by MDM Agent

| Type | ID | Data | Description |
|---|---|---|---|
| Reply | 1 | devNick errMsg | errMsg: error message |

For example: 1 192.168.30.81:1029 "4 arguments not enough."

F. MDM Agent memory information notification.

| Type | ID | Data | Description |
|---|---|---|---|
| Reply | 3 | devNick total free | total: total size of memory<br>free: free space of memory |

For example: 3 192.168.30.81:1029 62452 47968

**2. Get DLL Name of MDM Agent**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 33 | devNick [name] | name: DLL name<br>layer: device directory name under dll directory |
| Reply | 34 | devNick name [layer] | |
| Reply | 35 | devNick [name] | |

Note: If an optional name argument is given in the request, then it will check if the corresponding DLL exists. If it exists, only one message ID 34 with the given name is replied; otherwise message ID 34 is not replied. For example:

34 192.168.30.81:1029 net
34 192.168.30.81:1029 time
34 192.168.30.81:1029 launch
34 192.168.30.81:1029 fmngr
34 192.168.30.81:1029 shell
34 192.168.30.81:1029 proc
34 192.168.30.81:1029 system
35 192.168.30.81:1029

**3. System Execution**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 4 | devNick prog args | prog: executable program name or path<br>args: program arguments |
| Reply | 5 | devNick | |

Note: If no program arguments, use "" to denote no argument. If there are multiple arguments with spaces between them, also use double quote to surround them.

**4. Two-tier File Transfer**

Two-tier file transfer is used when MDM Client and MDM Gateway are on the same computer. The files on this computer can directly upload to MDM Agent or the files can directly download from MDM Agent to this computer.

A. File Download

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 61 | devNick devDir fName cliDir [dstName] | devDir: a device directory<br>fName: file name<br>cliDir: client directory<br>dstName: destination file name<br>reportKey: internal use<br>state: internal use<br>foffs: file offset<br>fsize: file size<br>cliPath: client file path<br>devPath: device file path |
| Reply | 53 | devNick reportKey state foffs fsize cliPath devPath | |
| Reply | 51 | devNick reportKey cliPath devPath | |

Note: The message ID 53 is used for file download status report. The message ID 51 denotes file download complete. For example:

53 192.168.30.82:1065 1 4 29 29 c:\temp\test.txt C:\test\test.txt
51 192.168.30.82:1065 1 c:\temp\test.txt C:\test\test.txt

B. File Upload

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 62 | devNick devDir fName cliDir [dstName] | devDir: a device directory<br>fName: file name<br>cliDir: client directory<br>dstName: destination file name<br>reportKey: internal use<br>state: internal use<br>foffs: file offset<br>fsize: file size<br>cliPath: client file path<br>devPath: device file path |
| Reply | 54 | devNick reportKey state foffs fsize cliPath devPath | |
| Reply | 52 | devNick reportKey cliPath devPath | |

Note: The message ID 54 is used for file upload status report. The message ID 52 denotes file upload complete. For example:

54 192.168.30.82:1065 1 5 0 29 c:\temp\test.txt C:\test\test.txt
54 192.168.30.82:1065 1 5 29 29 c:\temp\test.txt C:\test\test.txt
52 192.168.30.82:1065 1 c:\temp\tes.txt C:\test\test.txt

C. Notification of file transfer error

| Type | ID | Data | Description |
|---|---|---|---|
| Reply | 50 | devNick errMsg | errMsg: error message while file transfer |

For example:

50 192.168.30.82:1065 "fail to open file for read: C:\test\nofile.txt"

**5. Three-tier File Transfer**

A. File Download

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 73 | devNick devDir fName cliDir [dstName] | devDir: a device directory<br>fName: file name<br>cliDir: client directory<br>dstName: destination file name<br>reportKey: internal use<br>state: internal use<br>foffs: file offset<br>fsize: file size<br>cliPath: client file path<br>devPath: device file path |
| Reply | 53 | devNick reportKey state foffs fsize cliPath devPath | |
| Reply | 51 | devNick reportKey cliPath devPath | |

Note: The message ID 53 is used for file download status report. The message ID 51 denotes file download complete. For example:

53 192.168.30.82:1065 1 5 0 29 c:\temp\test.txt C:\test\test.txt
53 192.168.30.82:1065 1 5 29 29 c:\temp\test.txt C:\test\test.txt
51 192.168.30.82:1065 1 c:\temp\test.txt C:\test\test.txt

B. File Upload

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 74 | devNick devDir fName cliDir [dstName] | devDir: a device directory<br>fName: file name<br>cliDir: client directory<br>dstName: destination file name<br>reportKey: internal use<br>state: internal use<br>foffs: file offset<br>fsize: file size<br>cliPath: client file path<br>devPath: device file path |
| Reply | 54 | devNick reportKey state foffs fsize devPath cliPath | |
| Reply | 52 | devNick reportKey devPath cliPath | |

Note: The message ID 54 is used for file upload status report. The message ID 52 denotes file upload complete. For example:

54 192.168.30.82:1065 1 5 0 29 C:\test\test.txt c:\temp\test.txt
54 192.168.30.82:1065 1 5 29 29 C:\test\test.txt c:\temp\test.txt
52 192.168.30.82:1065 1 C:\test\test.txt c:\temp\test.txt

C. Notification of file transfer error

| Type | ID | Data | Description |
|---|---|---|---|
| Reply | 50 | devNick errMsg | errMsg: error message while file transfer |

For example: 50 192.168.30.82:1065 "fail to open file for read: c:\temp\nofile"

**6. Configuration File**

A. Reload configuration file

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 91 | devNick | |
| Reply | 92 | devNick | |

B. Set a configuration entry

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 93 | devNick key value | key: configuration entry name<br>value: configuration entry value |
| Reply | 94 | devNick key | |

For example: 94 192.168.30.82:1065 tcp_server_port

C. Save all configurations into configuration file

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 95 | devNick | |
| Reply | 96 | devNick | |

D. Get all configurations or a configuration entry

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 97 | devNick [key] | key: configuration entry name<br>value: configuration entry value |
| Reply | 98 | devNick key = value | |
| Reply | 99 | devNick [key] | |

Note: If optional key argument is given in the request, then it will check if the corresponding configuration entry exists. If it exists, only one message ID 98 with the given key is replied; otherwise message ID 98 is not replied. For example: 98 192.168.30.81:1029 tcp_server_port = 16888

E. Notification of configuration error

| Type | ID | Data | Description |
|---|---|---|---|
| Reply | 90 | devNick errMsg | errMsg: error message |

For example: 90 192.168.30.82:1065 "93 error setting variable wrong_key"

# Messages Supported by MDM File Manager Library

**1. Browse a device directory**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 140 | devNick devDir | devDir: a device directory<br>isDir: 1 for directory and 0 for not<br>fSize: file size; 0 for directory<br>fName: file or directory name<br>fTime: file modification time<br>fMode: Linux file mode; 0 for Windows |
| Reply | 141 | devNick isDir fSize fName fTime fMode | |
| Reply | 142 | devNick devDir | |

For example:

141 192.168.30.82:1065 0 57344 AutoLaunch.exe 2009/01/15 02:10:20 0
141 192.168.30.82:1065 1 0 dll 2010/06/30 12:50:39 0
141 192.168.30.82:1065 0 73728 mdm-main.exe 2010/06/30 08:36:37 0
141 192.168.30.82:1065 0 143360 mdm.dll 2010/06/30 06:43:34 0
141 192.168.30.82:1065 0 44544 ShutDownC.exe 2005/03/20 10:21:50 0
142 192.168.30.82:1065 C:\mdm

**2. Check if a device file or directory exists**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 143 | devNick devPath | devPath: a device path<br>flag: 1 for existing and 0 for not |
| Reply | 144 | devNick devPath flag | |

For example:

144 192.168.30.82:1065 C:\mdm 1
144 192.168.30.82:1065 C:\mdm\mdm-main.exe 1
144 192.168.30.82:1065 C:\fakeFolder 0

**3. Rename a device file or directory**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 145 | devNick srcPath dstPath | srcPath: source device path<br>dstPath: destination device path |
| Reply | 146 | devNick srcPath dstPath | |

For example:

146 192.168.30.82:1065 C:\test\command.txt C:\test\cmd.txt
146 192.168.30.82:1065 C:\test C:\tst

**4. Create a device directory**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 147 | devNick devDir | devDir: a device directory |
| Reply | 148 | devNick devDir | |

For example: 148 192.168.30.82:1065 C:\test

**5. Delete a device directory**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 149 | devNick devDir | devDir: a device directory |
| Reply | 150 | devNick devDir | |

Note: All files and directories under the target device directory are also deleted.

For example: 150 192.168.30.82:1065 C:\tst

**6. Delete a device file**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 151 | devNick devPath | devPath: a device path |
| Reply | 152 | devNick devPath | |

For example: 152 192.168.30.82:1065 C:\test\cmd.txt

**7. Delete all device files in a directory**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 153 | devNick devDir | devDir: a device directory |
| Reply | 154 | devNick devDir | |

For example: 154 192.168.30.82:1065 C:\tst

**8. Change file mode of a Linux device file or directory**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 155 | devNick devPath fMode | devPath: a device path |
| Reply | 156 | devNick devPath fMode | fMode: Linux file mode in 3 digits |

For example: 156 192.168.30.81:1029 /tmp/test.txt 664

# Messages Supported by MDM System Information Library

**1. Reboot device**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 170 | devNick | |
| Reply | 171 | devNick | |

**2. Get device system information**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 172 | devNick | |
| Reply | 173 | devNick agentVer model frmVer osVer CPUtype hostname | agentVer: MDM Agent version<br>model: product model name<br>frmVer: firmware version<br>osVer: OS version<br>CPUtype: CPU type<br>hostname: host name |

For example:

173 192.168.30.81:1029 "1.0.0 (Build 10062817)" IA3341-LX 1.0 "Standard Linux 2.6.9-uc0" MOXA-ART Moxa

173 192.168.30.82:1065 "1.0.0 Build 10063014" V2100-XPE 1.0 "Windows XPe 5.1" "Intel(R) Atom(TM)" OEM-NP05862JW93

**3. Get device storage information**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 174 | devNick | |
| Reply | 175 | devNick total free | total: total size of device storage<br>free: free space of device storage |

For example: 175 192.168.30.81:1029 3960 6144

# Messages Supported by MDM Network Management Library

**1. Obtain all device interface information**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 180 | devNick | |
| Reply | 181 | devNick name flag [IP] [mask] [gw] [MAC] | name: interface name<br>flag: 1 for fixed IP , 0 for DHCP<br>IP: IP address<br>mask: network mask<br>gw: default gateway<br>MAC: MAC address |
| Reply | 182 | devNick | |

Note: When an interface for DHCP is obtained, the last four arguments are not replied. For example:

181 192.168.30.81:1029 eth0 1 192.168.30.81 255.255.255.0 192.168.30.254 00:90:e8:e7:c3:d1
181 192.168.30.82:1065 "Local Area Connection" 1 192.168.30.82 255.255.255.0 192.168.30.254 00:90:E8:00:D6:88

**2. Set device interface information**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 183 | devNick name flag [IP] [mask] [gw] | name: interface name<br>flag: 1 for fixed IP , 0 for DHCP<br>IP: IP address<br>mask: network mask<br>gw: default gateway |
| Reply | 184 | devNick name | |

Note: When an interface for DHCP is set, the last four arguments are not sent. For example: 184 192.168.30.82:1065 "Local Area Connection 2"

**3. Get device DNS list of an interface**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 185 | devNick name | name: interface name<br>DNSlist: DNS server list |
| Reply | 186 | devNick name [DNSlist] | |

Note: If there are multiple DNS servers for the DNSlist, then each DNS server is treated as an argument and a double quote is not needed. For example:

186 192.168.30.81:1029 eth0 192.168.1.97 192.168.1.91
186 192.168.30.82:1065 "Local Area Connection" 192.168.1.91 192.168.1.97

**4. Set device DNS list for an interface**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 187 | devNick name [DNSlist] | name: interface name<br>DNSlist: DNS server list |
| Reply | 188 | devNick name | |

Note: To clear DNS setting, don't give values to NDSlist to denote no DNS server list. For example: 188 192.168.30.82:1065 "Local Area Connection"

# Messages Supported by MDM Time Management Library

**1. Get device time**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 190 | devNick | YYYY-MM-DD: date |
| Reply | 191 | devNick YYYY-MM-DD hh:mm:ss | hh:mm:ss: time |

For example: 191 192.168.30.81:1029 2010-06-28 18:52:39

**2. Set device time**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 192 | devNick YYYY-MM-DD hh:mm:ss | YYYY-MM-DD: date |
| Reply | 193 | devNick | hh:mm:ss: time |

# Messages Supported by MDM Process Management Library

**1. Get device process information**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 200 | devNick | pid: process ID |
| Reply | 201 | devNick pid progCmd | progCmd: program command |
| Reply | 202 | devNick | |

For example:

201 192.168.30.82:1065 3052 rdpclip.exe
201 192.168.30.82:1065 3192 LogonUI.exe
201 192.168.30.82:1065 3428 NotifyWindow.exe
201 192.168.30.82:1065 3460 RTHDCPL.EXE
201 192.168.30.82:1065 116 mdm-main.exe
201 192.168.30.82:1065 2316 scrnsave.scr

**2. Stop a device process**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 203 | devNick pid [progCmd] | pid: process ID |
| Reply | 204 | devNick | progCmd: optional program command |

# Messages Supported by MDM Auto-launch Library

**1. Get device auto-launch information**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 210 | devNick | prog: executable program name or path |
| Reply | 211 | devNick prog args | args: program arguments |
| Reply | 212 | devNick | |

Note: If no program arguments, use "" to denote no argument. If there are multiple arguments with spaces between them, also use double quote to surround them. For example: 211 192.168.30.82:1065 C:\mdm\mdm-main.exe

**2. Set device auto-launch information**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 213 | devNick prog args | prog: executable program name or path<br>args: program arguments |
| Request | 214 | devNick | |
| Reply | 215 | devNick | |

Note: If no program arguments, use "" to denote no argument. If there are multiple arguments with spaces between them, also use double quote to surround them.

# Messages Supported by MDM Shell Execution Library

**1. Execute a device program to get its output and execution result**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 280 | devNick devPath [args] | devPath: executable program path<br>args: optional program arguments<br>id: ID from 0 to 15<br>data: program output data<br>returnCode: 0 for success and others for failure |
| Reply | 281 | devNick id | |
| Reply | 286 | devNick id data | |
| Reply | 287 | devNick id returnCode | |

Note: The double quote cannot be used in message ID 286. For example:

281 192.168.30.82:1065 0
287 192.168.30.82:1065 0 0

**2. Stop execution of a device program**

| Type | ID | Data | Description |
|---|---|---|---|
| Request | 282 | devNick id | id: ID from 0 to 15 |
| Reply | 283 | devNick id | |

For example: 283 192.168.30.82:1065 0

# B

# MDM API Functions

The following lists all MDM API functions for reference purposes.

| Start up an MDM Core in a program or thread with specified type(s) |
|---|
| **int mdm_start(unsigned int type, unsigned int thread);** |
| Input:<br><type> the type of program to start up<br><thread> 1 for thread and 0 for program<br>Output: none |
| Return:<br>0 on success and negative value for failure. |

| Check if the MDM Core is alive |
|---|
| **int mdm_alive(void);** |
| Input: none<br>Output: none |
| Return:<br>1 for alive and 0 for dead |

| Stop the MDM Core |
|---|
| **void mdm_stop(void);** |
| Input: none<br>Output: none |
| Return: none |

| Set an user-defined function to handle the messages delivered from gateways |
|---|
| **void mdm_set_mngr_function(mdm_message_t msg);** |
| Input:<br><msg> an user-defined function for MDM UI/Client<br>Output: none |
| Return: none |

| Split a message packet in char string format with space delimiters into an array of string elements |
|---|
| **int split_line(char **av, int count, char *pkt);** |
| Input:<br><pkt> the message packet<br><count> the size of the array av<br>Output:<br><av> point to the array of the string elements |
| Return:<br>the number of string elements |

| Parse a message packet in char string format with space delimiters |
|---|
| **char* next_arg(char **pkt);** |
| Input:<br><pkt> point to the address of the message packet<br>Output:<br><pkt> move the address to the next element |
| Return:<br>point to the first element |

| Send a formatted message over a one-hop connection. |
|---|
| **void send_cmd(MDMCONN *con, unsigned int tag, const char *fmt, ...);** |
| Input:<br><con> the connection<br><tag> message id<br><fmt> format the content of the message<br>Output: none |
| Return: none |

| Send a formatted message over two-hop connections. |
|---|
| **void send_peer(MDMCONN *con, char *nick, unsigned int tag, const char *fmt, ...);** |
| Input:<br><con> the connection<br><nick> the name/id of the device<br><tag> message id<br><fmt> format the content of the message<br>Output: none |
| Return: none |

| Send a non-formatted message over a one-hop connection |
|---|
| **void send_cmd_data(MDMCONN *con, unsigned int tag, const char *pkt, unsigned int len);** |
| Input:<br><con> the connection<br><tag> message id<br><pkt> the message data<br><len> the length of the message<br>Output: none |
| Return: none |

| Send a non-formatted message over two-hop connections. |
|---|
| **void send_peer_data(MDMCONN *con, char *nick, unsigned int tag, const char *pkt, unsigned int len);** |
| Input:<br><con> the connection<br><nick> the name/id of the device<br><tag> message id<br><pkt> the message data<br><len> the length of the message<br>Output: none |
| Return: none |