MAGTEK®
SECURITY FROM THE INSIDE

oDynamo

MagTek Common Message Structure (MTCMS)
Programmer's Reference Manual (C++)

May 2018

Manual Part Number:
D998200159-10

REGISTERED TO ISO 9001:2015

MagTek I 1710 Apollo Court I Seal Beach, CA 90740 I Phone: (562) 546-6400 I Technical Support: (888) 624-8350
www.magtek.com

Table 0.1 Revisions

Rev Number	Date	Notes
10	05/25/2018	Initial Release

SOFTWARE LICENSE AGREEMENT

IMPORTANT: YOU SHOULD CAREFULLY READ ALL THE TERMS, CONDITIONS AND RESTRICTIONS OF THIS LICENSE AGREEMENT BEFORE INSTALLING THE SOFTWARE PACKAGE. YOUR INSTALLATION OF THE SOFTWARE PACKAGE PRESUMES YOUR ACCEPTANCE OF THE TERMS, CONDITIONS, AND RESTRICTIONS CONTAINED IN THIS AGREEMENT. IF YOU DO NOT AGREE WITH THESE TERMS, CONDITIONS, AND RESTRICTIONS, PROMPTLY RETURN THE SOFTWARE PACKAGE AND ASSOCIATED DOCUMENTATION TO THE ADDRESS ON THE FRONT PAGE OF THIS DOCUMENT, ATTENTION: CUSTOMER SUPPORT.

TERMS, CONDITIONS, AND RESTRICTIONS

MagTek, Incorporated (the "Licensor") owns and has the right to distribute the described software and documentation, collectively referred to as the "Software."

LICENSE: Licensor grants you (the "Licensee") the right to use the Software in conjunction with MagTek products. LICENSEE MAY NOT COPY, MODIFY, OR TRANSFER THE SOFTWARE IN WHOLE OR IN PART EXCEPT AS EXPRESSLY PROVIDED IN THIS AGREEMENT. Licensee may not decompile, disassemble, or in any other manner attempt to reverse engineer the Software. Licensee shall not tamper with, bypass, or alter any security features of the software or attempt to do so.

TRANSFER: Licensee may not transfer the Software or license to the Software to another party without the prior written authorization of the Licensor. If Licensee transfers the Software without authorization, all rights granted under this Agreement are automatically terminated.

COPYRIGHT: The Software is copyrighted. Licensee may not copy the Software except for archival purposes or to load for execution purposes. All other copies of the Software are in violation of this Agreement.

TERM: This Agreement is in effect as long as Licensee continues the use of the Software. The Licensor also reserves the right to terminate this Agreement if Licensee fails to comply with any of the terms, conditions, or restrictions contained herein. Should Licensor terminate this Agreement due to Licensee's failure to comply, Licensee agrees to return the Software to Licensor. Receipt of returned Software by the Licensor shall mark the termination.

LIMITED WARRANTY: Licensor warrants to the Licensee that the disk(s) or other media on which the Software is recorded are free from defects in material or workmanship under normal use.

THE SOFTWARE IS PROVIDED AS IS. LICENSOR MAKES NO OTHER WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Because of the diversity of conditions and PC hardware under which the Software may be used, Licensor does not warrant that the Software will meet Licensee specifications or that the operation of the Software will be uninterrupted or free of errors.

IN NO EVENT WILL LICENSOR BE LIABLE FOR ANY DAMAGES, INCLUDING ANY LOST PROFITS, LOST SAVINGS, OR OTHER INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE, OR INABILITY TO USE, THE SOFTWARE. Licensee's sole remedy in the event of a defect in material or workmanship is expressly limited to replacement of the Software disk(s) if applicable.

GOVERNING LAW: If any provision of this Agreement is found to be unlawful, void, or unenforceable, that provision shall be removed from consideration under this Agreement and will not affect the enforceability of any of the remaining provisions. This Agreement shall be governed by the laws of the State of California and shall inure to the benefit of MagTek, Incorporated, its successors or assigns.

ACKNOWLEDGMENT: LICENSEE ACKNOWLEDGES THAT HE HAS READ THIS AGREEMENT, UNDERSTANDS ALL OF ITS TERMS, CONDITIONS, AND RESTRICTIONS, AND AGREES TO BE BOUND BY THEM. LICENSEE ALSO AGREES THAT THIS AGREEMENT SUPERSEDES ANY AND ALL VERBAL AND WRITTEN COMMUNICATIONS BETWEEN LICENSOR AND LICENSEE OR THEIR ASSIGNS RELATING TO THE SUBJECT MATTER OF THIS AGREEMENT.

QUESTIONS REGARDING THIS AGREEMENT SHOULD BE ADDRESSED IN WRITING TO MAGTEK, INCORPORATED, ATTENTION: CUSTOMER SUPPORT, AT THE ADDRESS LISTED IN THIS DOCUMENT, OR E-MAILED TO SUPPORT@MAGTEK.COM.

Table of Contents

1 Introduction

This document provides instructions for software developers who want to create Windows C++ software solutions that include a MagTek Common Message Structure (MTCMS) device connected to a Windows PC.

2 About MTCMS Library

Custom Windows software installed on a host PC can communicate with MagTek Common Message Structure (MTCMS) devices via USB, network interface, or serial interface using the MTCMS library.

The supported platforms for Windows C++ projects include Windows 7, Windows 8/8.1, and Windows 10. The Windows C++ project should contain references to the main library file: **`MTCMS.dll`**.

3 How to Set Up the MagTek CMS SDK for Windows C++ Projects

To add the MagTek CMS libraries to a Windows C++ project in Microsoft Visual Studio, follow these steps:

1) Create or open your project in Visual Studio.
2) Copy the following DLL file from the **MTCMSDemo** folders to the library folder of your software project:
   - `MTCMS.dll`
3) In the Visual Studio Solution Explorer, right-click the project and select **Add Reference** to show the **Add Reference** window.
4) Select the **Browse** tab and press the **Browse...** button.
5) Navigate to your library folder, select **`MTCMS.dll`**, then press the **Add** button.
6) In your custom software, create an instance of **`MTDevice`**. For examples, see the source code included with the **MTCMSDemo** project and/or **Appendix A** Code Examples.
7) Begin using the features provided by the MTCMS library.

4 MTCMS Class Methods

After creating an instance of the MTCMS class in your software project, use the methods described in this section to communicate with MagTek CMS device.

4.1 requestDeviceList

This method initiates request to discover devices that are visible to the host using the specified connection interface. The DeviceListReceived event will provide information regarding the available devices once the discovery process is completed.

```
MTCMS_API void requestDeviceList(MTConnectionType connectionType);
```

Parameters:

Parameter	Description

connectionType	MTConnectionType value:
	`MTConnectionType.USB,`
	`MTConnectionType.IP,`
	`MTConnectionType.Serial`

Return Value: None

4.2 setConnectionType

This method sets the connection type of the device..

```
MTCMS_API void setConnectionType(MTConnectionType connectionType);
```

Parameters:

Parameter	Description

connectionType	MTConnectionType value:
	`MTConnectionType.USB,`
	`MTConnectionType.IP,`
	`MTConnectionType.Serial`

Return Value: None

4.3 setAddress

This method sets the address of the device.

```
MTCMS_API void setAddress(const char* deviceAddress);
```

Parameters:

Parameter	Description
deviceAddress	String value of the address.

The following table shows the address formats supported by the different connection types:

<table>
<tr><th>Connection Type</th><th>Address Format</th></tr>
<tr><td>USB</td><td>[PATH]

<table><tr><th>Parameter</th><th>Description</th></tr><tr><td>[PATH]</td><td>The OS specific device path to the USB device. The path is normally retrieved from the Address property of MTDeviceInformation.</td></tr></table></td></tr>
<tr><td>IP</td><td>[IPA]
or
[IPA]:[PORT]

<table><tr><th>Parameter</th><th>Description</th></tr><tr><td>[IPA]</td><td>The IP address of the device in dotted-quad notation (i.e. 192.178.1.123).</td></tr><tr><td>[PORT]</td><td>The TCP port of the device.
(Default: 5000)</td></tr></table></td></tr>
</table>

Serial

PORT=**[PORT]**,
BAUDRATE=**[BAUDRATE]**,
DATABITS=**[DATABITS]**,
PARITY=**[PARITY]**,
STOPBITS=**[STOPBITS]**,
HANDSHAKE=**[HANDSHAKE]**,
STARTINGBYTE=**[STARTINGBYTE]**,
ENDINGBYTE=**[ENDINGBYTE]**,
CRCMODE=**[CRCMODE]**

Parameter	Description
[PORT]	The OS specific device path to the serial port (i.e. COM4).
[BAUDRATE]	The data baud rate . (Default: 9600)
[DATABITS]	The data bits per byte. (Default: 8)
[PARITY]	The parity checking protocol. (Default: NONE). Supported Values: NONE,EVEN,ODD,SPACE,MARK
[STOPBITS]	The number of stop bits per byte. (Default: 1) Supported Values: 1,1.5,2
[HANDSHAKE]	The handshaking protocol for serial port transmission of data. (Default: NONE) Supported Values: NONE,RTS,XONXOFF,RTSXONSOFF
[STARTINGBYTE]	The special character used as the starting byte for each message. (Default is empty string) An empty string indicates no special character is used as the starting byte for each message.
[ENDINGBYTE]	The special character used as the ending byte for each message. (Default is 0x0A)

Connection Type	Address Format	
		An empty string indicates no special character is used as the ending byte for each message.
	[CRCMODE]	A value of 0 indicates CRC is disabled, otherwise CRC is enabled. (Default: 0)

Return Value: None

4.4 setDeviceID

This method sets the device ID.

```
MTCMS_API void setDeviceID(const char* deviceID);
```

Parameters:

Parameter	Description
deviceID	String value of the device ID.

Return Value: None

4.5 openDevice

This method opens the connection to the device.

```
MTCMS_API void openDevice();
```

Parameters: None

Return Value: None

4.6 closeDevice

This method closes the connection to the device.

```
MTCMS_API void closeDevice();
```

Parameters: None

Return Value: None

4.7 isDeviceConnected

This method returns whether the device is connected or not.

```
MTCMS_API bool isDeviceConnected();
```

Parameters: None

Return Value:
Return true if the device is connected. Otherwise, return false.

4.8 sendDataString

This method sends a command string to the device.

```
MTCMS_API int sendCommandString(const char* dataString);
```

Parameters:

Parameter	Description
dataString	Command to be sent in hexadecimal string format.

Return Value:

- 0 = Success
- 9 = Error
- 15 = Busy

4.9 sendDataBytes

This method sends a command to the device.

```
MTCMS_API int sendDataBytes(
const unsigned char* dataBytes,
int dataBytesLength);
```

Parameters:

Parameter	Description
dataBytes	Command to be sent in byte array format.
dataBytesLength	Length of the command bytes.

Return Value:

- 0 = Success
- 9 = Error
- 15 = Busy

4.10 sendMTCMSMessage

This method sends a command to the device.

```
MTCMS_API int sendMTCMSMessage(MTCMSMessage* message);
```

Parameters:

Parameter	Description
message	MTCMSMessage to be sent to the device.

Return Value:

- 0 = Success

- 9 = Error
- 15 = Busy

5 MTCMSMessage Structure

These methods allows building CMS messages to be used in communications with MagTek CMS devices.

5.1 CreateMTCMSMessage

This constructor method builds an MTCMSMessage instance with the provided values.

```
MTCMS_API MTCMSMessage* CreateMTCMSMessage(
int messageType,
int applicationID,
int commandID,
int dataTag,
const unsigned char* data,
int dataLength);
```

Parameters:

Parameter	Description
messageType	MessageType value
applicationID	ApplicationID value
commandID	CommandID value
dataTag	Data tag value
data	Data value
dataLength	Length of data value

Return Value: MTCMSMessage structure

5.2 CreateMTCMSMessageFromBytes

This constructor method builds an MTCMSMessage instance with the provided values.

```
MTCMS_API MTCMSMessage* CreateMTCMSMessageFromBytes(
const unsigned char* messageBytes,
int messageBytesLength);
```

Parameters:

Parameter	Description
messageBytes	Message in byte array value
messageBytesLength	Length of message bytes value

Return Value: MTCMSMessage structure

5.3 CreateMTCMSRequestMessage

This constructor method builds an MTCMSMessage instance with the provided values.

```
MTCMS_API MTCMSMessage* MTCMSMessage(
int applicationID,
int commandID,
```

```
int dataTag,
const unsigned char* data,
int dataLength);
```

Parameters:

Parameter	Description
applicationID	ApplicationID value
commandID	CommandID value
dataTag	Data tag value
data	Data value
dataLength	Length of data value

Return Value: MTCMSMessage structure

5.4 ReleaseMTCMSRequestMessage

This method releases the resource allocated by calls to CreateMTCMSMessage and CreateMTCMSRequestMessage methods.

```
MTCMS_API void ReleaseMTCMSMessage(MTCMSMessage* message);
```

Parameters:

Parameter	Description
message	MTCMSMessage to deallocate.

Return Value: None

6 MTCMS Library Enumerations and Structures

The MTCMS Library uses the following constants and data structures.

6.1 MTConnectionType Values

Device connection types:

```
enum MTConnectionType
{
      USB,
      IP,
      Serial
};
```

6.2 MTConnectionState Values

Device connection states:

```
enum MTConnectionState
{
      Disconnected,
      Connecting,
```

```
      Error,
      Connected,
      Disconnecting
};
```

6.3 MTDeviceInformation

Device information structure:

```
struct MTDeviceInformation
{
      const char* Id;
      const char* Name;
      const char* Address;
      const char* ProductId;
};
```

6.4 MTCMSMessage

MTCMSMessage structure:

```
struct MTCMSMessage
{
      int messageType;
      int applicationID;
      int commandID;
      int resultCode;
      int dataTag;
      const unsigned char* data;
      int dataLength;
      const unsigned char* messageBytes;
      int messageBytesLength;
};
```

7 MTDevice Events

7.1 OnDeviceList

The library will call this function when device information is available.

```
typedef void (__stdcall * OnDeviceListEvent)(
void* sender,
MTConnectionType connectionType,
int deviceCount,
MTDeviceInformation* deviceList);
```

Parameter	Description
sender	Object representing the publisher of the event.

connectionType	MTConnectionType value:
	`MTConnectionType.USB,`
	`MTConnectionType.IP,`
	`MTConnectionType.Serial`

deviceCount	Number of devices in deviceList.
deviceList	A list of MTDeviceInformation objects.

Return Value: None

7.2 OnDeviceConnectionStateChanged

This event occurs when the connection state of the device is changed.

```
typedef void (__stdcall * OnDeviceConnectionStateChangedEvent)(
void* sender,
MTConnectionState state);
```

Parameter	Description
sender	Object representing the publisher of the event.

state	MTConnectionState value indicating the state of the device:
	`MTConnectionState.Disconnected`
	`MTConnectionState.Connecting`
	`MTConnectionState.Error`
	`MTConnectionState.Connected`
	`MTConnectionState.Disconnecting`

Return Value: None

7.3 OnDeviceDataString

This event occurs when a response is received from the device.

```
typedef void (__stdcall * OnDeviceDataStringEvent)(
void* sender,
```

```
const char* dataString);
```

Parameter	Description
sender	Object representing the publisher of the event.
dataString	String representing data received.

Return Value: None

7.4 OnDeviceDataBytes

This event occurs when a response is received from the device.

```
typedef void (__stdcall * OnDeviceDataBytesEvent)(
void* sender,
const unsigned char* dataBytes
int dataLength);
```

Parameter	Description
sender	Object representing the publisher of the event.
dataBytes	Bytes representing data received.
dataLength	Length of dataBytes.

Return Value: None

7.5 OnDeviceResponseMessage

This event occurs when a response is received from the device.

```
typedef void (__stdcall * OnDeviceResponseMessageEvent)(
void* sender,
const MTCMSMessage* response);
```

Parameter	Description
sender	Object representing the publisher of the event.
response	MTCMSMessage representing data received.

7.6 OnDeviceNotificationMessage

This event occurs when a response is received from the device.

```
typedef void (__stdcall * OnDeviceNotificationMessageEvent)(
void* sender,
const MTCMSMessage* notification);
```

Parameter	Description
sender	Object representing the publisher of the event.
notification	MTCMSMessage representing data received.

Appendix A Code Examples

A.1 Connect to Device

```
openDevice();
```

A.2 Send Command String to Device

```
const char* dataString = "C00101C10100C20114";
sendDataString(dataString);
```

A.3 Send Command Bytes to Device

```
unsigned char* dataBytes =
{0xC0,0x01,0x01,0xC1,0x01,0x00,0xC2,0x01,0x14};

sendDataBytes(dataBytes,9);
```

A.4 Send MTCMSMessage to Device

```
MTCMSMessage* message = CreateMTCMSMessage(0x01,0x00,0x14, NULL,0,
false);
sendMTCMSMessage(message);
ReleaseMTCMSMessage(message);
```

A.5 Send MTCMSRequestMessage to Device

```
MTCMSMessage* requestMessage = CreateMTCMSRequestMessage
                                        (0x00,0x14, NULL,0, false);
sendMTCMSMessage(requestMessage);
ReleaseMTCMSMessage(message);
```

A.6 Receiving Connection State Updates from Device

```
void  _stdcall OnDeviceConnectionStateChanged(void* sender,
MTConnectionState state)
{
      switch (state)
      {
      case MTConnectionState::Connected:
            Log(_T("[Connected]"));
            break;
      case MTConnectionState::Connecting:
            Log(_T("[Connecting...]"));
            break;
      case MTConnectionState::Disconnecting:
            Log(_T("[Disconnecting...]"));
            break;
      case MTConnectionState::Disconnected:
            Log(_T("[Disconnected]"));
            break;
      }
```

```
}
```

A.7 Receiving Response Message from Device

```
void __stdcall OnDeviceResponseMessage(void* sender, const
MTCMSMessage* response)
{
      processResponseMessage(response);
}
```

A.8 Receiving Notification Message from Device

```
void __stdcall OnDeviceNotificationMessage(void* sender, const
MTCMSMessage* notification)
{
      processNotificationMessage(notification);
}
```

A.9 Close Device

```
closeDevice();
```