

1 MeCom Protocol Specification

1.1 Introduction

This document describes the Meerstetter Engineering GmbH (meCom) communication standard. The objective of this standard is to ensure link-layer and application-layer compatibility between Devices and Host Systems and Service stations. This document is intended to assist designers of Meerstetter Engineering GmbH equipment by providing a high-level common reference document.

The Protocol uses a client-server architecture. This type of system has several client nodes (the user control devices) that issue explicit commands to the server node and process responses. The Server node will not typically transmit data without a request from the client node and does not communicate with other servers.

1.2 Frame Structure

Link layer transmissions are sent in small blocks of data, called frames. Each frame is made up of several smaller groups, called FrameField. The table below illustrates the basic construction of frames.

| Field | FrameField | Data Type | Length | Description |
|-------|--------------|------------|------------|------------------------------------|
| 1 | Control | ASCII char | 8 Bits | See Control FrameField |
| 2 | Address | UINT8 | 16 Bits | See Device Address |
| 3 | Sequence Nr. | UINT16 | 32 Bits | See Sequence Number |
| 4 | Payload | N * 8 Bits | N * 8 Bits | See Application Protocol Structure |
| 5 | Frame CRC | UINT16 | 32 Bits | See Frame CRC Algorithm |
| 6 | End-of-Frame | ASCII char | 8 Bits | ASCII <CR> |

1.2.1 Control FrameField

The identifiers have been selected such that the receiving device, to detect the start of a frame, can synchronize to a character with a value of the following table:

| Control Type | ASCII | HEX |
|------------------|-------|------|
| Device | ! | 0x21 |
| Host Frame Start | # | 0x23 |

The Server does always use Control Type "Device". The user may use one of the 2 interfaces. Usually the interface to be used is specified in the detailed communication protocol of the corresponding document.

1.2.2 Address

This address field represents always the device address. The device response does also contain the own device address.

1.2.3 Sequence Number

The sequence number is always replayed by the device to help the client system to ensure that this is the right answer on a specific question. It is recommended to initialize the sequence number on the client machine to a random value and increase it on every message. If the device does not send an answer (maybe because the request or answer got lost), it is recommended to use the same sequence number again, to resend the query or set command.

1.2.4 Frame CRC

The Frame Checksum is a CRC-CCITT (CRC-16) calculated by both the sender and the receiver of a frame. It ensures that the frame was not corrupted by the transmission medium. The Frame Checksum is represented by four HEX digits. See at the C Code Example in the Appendix.

1.3 Application Protocol Structure

It is used to control the Device from a Master Software (Host Software). In this context, the Master Software acts as a client and the device acts as a server.

1.3.1 Set Command

These operations are used to set a specific parameter or trigger an action on the Device. The server responds in either of the following ways:

- a. Send acknowledge message
- b. Send error message

The content of the Payload FrameField depends on the command and is specified in the device detailed communication protocol.

The following example sends a reset command to the device.

| FrameField | Control | Address | Sequence Nr | Payload | Frame CRC | End of Frame |
|------------------------|---------|---------|-------------|--------------------|-----------|--------------|
| DataFieldType | 1 ASCII | UINT8 | UINT16 | Depends on command | UINT16 | 1 ASCII |
| Tx Data Example | # | 00 | BDE2 | RS | 9780 | <CR> |

The length and the data of the Payload FrameField depends on the used command.

If the Device accepts the command, it will answer with an Acknowledge. The Acknowledge is always returning the CRC of the prior set command.

| FrameField | Control | Address | Sequence Nr | CRC from Set Command | End of Frame |
|------------------------|---------|---------|-------------|----------------------|--------------|
| DataFieldType | 1 ASCII | UINT8 | UINT16 | UINT16 | 1 ASCII |
| Rx Data Example | ! | 00 | BDE2 | 9780 | <CR> |

1.3.2 Query Commands

These operations are used to query a specific parameter or status of the device. The server responds in either of the following ways:

- a) Send response message with the requested content.
- b) Send error message

The content of the Payload FrameField depends on the command and is specified in the device detailed communication protocol.

The following example queries the Identification String of the device:

| FrameField | Control | Address | Sequence Nr | Payload | Frame CRC | End of Frame |
|------------------------|---------|---------|-------------|--------------------|-----------|--------------|
| DataFieldType | 1 ASCII | UINT8 | UINT16 | Depends on command | UINT16 | 1 ASCII |
| Tx Data Example | # | 01 | 15AA | ?IF | 257D | <CR> |

If the Device accepts the command, it will answer with the following structure

| FrameField | Control | Address | Sequence Nr | Payload | Frame CRC | End of Frame |
|------------------------|---------|---------|-------------|----------------------|-----------|--------------|
| DataFieldType | 1 ASCII | UINT8 | UINT16 | Depends on command | UINT16 | 1 ASCII |
| Rx Data Example | ! | 01 | 15AA | 8065-TEC SW G01..... | 342D | <CR> |

1.3.3 DataFieldType Definition

For all operations requiring an argument, this directly follows the command mnemonic. Since mnemonics have fixed length, no special delimiter is required.
Numerical values are represented as follows:

| DataFieldType | Length on Data string | Representation | Range |
|---------------|-----------------------|----------------|----------------------------|
| UINT4 | 1 ASCII character | 0...F | 0 ... 15 |
| UINT8 | 2 ASCII characters | 2 HEX digits | 0 ... 255 |
| UINT16 | 4 ASCII characters | 4 HEX digits | 0 ... 65535 |
| UINT32 | 8 ASCII characters | 8 HEX digits | 0 ... 4294967295 |
| INT8 | 2 ASCII characters | 2 HEX digits | -128 ... 127 |
| INT16 | 4 ASCII characters | 4 HEX digits | -32768 ... 32767 |
| INT32 | 8 ASCII characters | 8 HEX digits | -2147483648 ... 2147483647 |
| FLOAT32 | 8 ASCII characters | 8 HEX digits | According to IEEE754 |
| DOUBLE64 | 16 ASCII characters | 16 HEX digits | According to IEEE754 |

Example: A UINT16 value of decimal 23456 is transmitted as 4 ASCII chars: 5BA0

The memory area of the FLOAT32 value is copied to a UINT32 memory area and transferred as it would be a UINT32.

1.3.4 Server Error Codes

Any Set or Query command may result in an error condition on the server side. For the client application to recover from as many error conditions as possible, the server indicates the error codes listed in below.

The Error Message has this format:

The following example responds with the Server Error Code "EER_PAR_NOT_AVAILABLE".

| FrameField | Control | Address | Sequence Nr | Payload | Frame CRC | End of Frame |
|-----------------|---------|---------|-------------|-------------|-----------|--------------|
| DataFieldType | 1 ASCII | UINT8 | UINT16 | + and UINT8 | UINT16 | 1 ASCII |
| Rx Data Example | ! | 01 | 15AC | +05 | 7509 | <CR> |

Errors 0 ... 99 → Common errors

Errors 100 ... 255 → Device specific errors

| Error Code | Symbol | Description |
|------------|----------------------------|--|
| 1 | EER_CMD_NOT_AVAILABLE | Command not available |
| 2 | EER_DEVICE_BUSY | Device is busy |
| 3 | ERR_GENERAL_COM | General communication error |
| 4 | EER_FORMAT | Format error |
| 5 | EER_PAR_NOT_AVAILABLE | Parameter is not available |
| 6 | EER_PAR_NOT_WRITABLE | Parameter is read only |
| 7 | EER_PAR_OUT_OF_RANGE | Value is out of range |
| 8 | EER_PAR_INST_NOT_AVAILABLE | Instance is not available |
| 9 | ERR_PAR_GENERAL_FAILURE | Parameter general Error. Device internal failure on this parameter. |

2 General Application commands

- The following listed commands are usually available in all devices from Meerstetter Engineering.
- This capture only shows the content of the Payload FrameField.

2.1 Reset Device “RS”

The following command specifies the **Payload** to reset the Device.

After receiving the command, the device will reset after a delay of 200ms.

| | |
|------------------------|---------------|
| PayloadField | Command |
| DataFieldType | 2 ASCII Chars |
| Tx Data Example | RS |

Response:

- ACK if everything is ok.
- Various server error codes.

2.2 Read Firmware Identification String “?IF”

The following command specifies the **Payload** for querying the firmware Identification String.

The Firmware Identification String for this example is: 9017-LDD QCL SW G01

The Firmware Identification String is always 20 chars long. Not used chars are filled up with spaces.

| | |
|----------------------|---------------|
| PayloadField | Command |
| DataFieldType | 3 ASCII Chars |
| Example | ?IF |

Response:

| | |
|------------------------|---------------------|
| PayloadField | Value |
| DataFieldType | 20x ASCII Chars |
| Rx Data Example | 9017-LDD QCL SW G01 |

3 Application Commands to Interact with the Parameter System

- Most of the following commands are available on our devices. Please refer to the device specific communication protocol for details.
- The examples do always specify the exact **Payload** data, which must be sent, or will be received in the specified command.
- The Payload must be placed into the MeCom Frame, which is specified in this document.

3.1 General Explanations

- The Device has a parameter system, which manages all parameters that are accessible from outside.
- The **MeParID** identifies each parameter group.
- Each Parameter group may have several Instances, identified over the Instance number. Usually only instance 1 or 2 is used.
- Each parameter has a specified **MeParType** format for the value.
- Each parameter has specified the **MeParFlags**.
- The newer graphical Configuration Software publishes all the information about MeParID, Instance and MeParType in a tooltip field assigned to every graphical parameter field.
- The older Service Software does not have this feature. The information about the parameters are contained in the specific communication protocol documents.

3.1.1 MeParType Definition

The value of the parameters value may have different types for the data representation. The communication protocol itself know much more datatypes then the parameter system is effectively using. The following table shows all the used MeParType and the corresponding enumeration.

| MeParType | MeParType Enumeration (UINT8) (Type field in Meta Data Command) |
|------------------|---|
| FLOAT32 | 0 |
| INT32 | 1 |
| DOUBLE64 * | 2 |
| LATIN1 | 3 |
| BYTE | 4 |

*DOUBLE64 is not used in the current firmware.

3.1.2 MeParFlags Definition

This UINT8 variable is used as Bit Field to define the following states:

| Bit Number in UINT8 | Meaning |
|---------------------|-------------------------------------|
| 0 | Reading this parameter is permitted |
| 1 | Writing this parameter is permitted |
| 2 | Not used, do ignore. |
| 3 | Not used, do ignore. |
| 4 | Not used, do ignore. |
| 5 | Not used, do ignore. |
| 6 | Not used, do ignore. |
| 7 | Not used, do ignore. |

3.2 Commands to interact with the Parameter System

3.2.1 Read Value “?VR”

The following command specifies the Payload for querying a specific parameter value.

The example queries the MeParID 1000 and the Instance 1 and gets returned a FLOAT32 value of 22.34.

Query Payload:

| | | | |
|------------------------|---------------|---------|----------|
| PayloadField | Command | MeParID | Instance |
| DataFieldType | 3 ASCII Chars | UINT16 | UINT8 |
| Tx Data Example | ?VR | 03E8 | 01 |
| Decoded Example | ?VR | 1000 | CH1 |

Response Payload:

| | |
|------------------------|-----------|
| PayloadField | Value |
| DataFieldType | MeParType |
| Rx Data Example | 41B2B852 |
| Decoded Example | 22.34 |

3.2.2 Set Value “VS”

The following command specifies the Payload for setting a specific parameter value.

The Example sets the MeParID 3000, Instance 1 to a FLOAT32 value of 25.00.

Query Payload:

| | | | | |
|------------------------|---------------|---------|----------|--------------|
| PayloadField | Command | MeParID | Instance | Value to set |
| DataFieldType | 2 ASCII Chars | UINT16 | UINT8 | MeParType |
| Tx Data Example | VS | 0BB8 | 01 | 41C80000 |
| Decoded Example | VS | 3000 | CH1 | 25.00 |

Response Payload:

- ACK if everything is ok.
- Various server error codes.

3.2.3 Read Meta Data for the parameter “?VM”

This command specifies the **Payload** to query all Meta data information about a parameter. It is optional to use this command. You may also read out all Meta information about the parameter from the Service Software tool tip and program it static to your application.

The Example queries the Meta data for the MeParID 1000, Instance 1

Query Payload:

| | | | |
|------------------------|---------------|---------|----------|
| PayloadField | Command | MeParID | Instance |
| DataFieldType | 3 ASCII Chars | UINT16 | UINT8 |
| Tx Data Example | ?VM | 03E8 | 01 |
| Decoded Example | ?VM | 1000 | CH1 |

Response Payload:

| PayloadField | MeParType Enumeration | MeParFlags | Max nr of Instances | Max nr of Elements | Minimum Value | Maximum Value | Actual Value |
|------------------------|-----------------------|------------|---------------------|--------------------|-----------------------|---------------|--------------|
| DataFieldType | UINT8 | UINT8 | UINT8 | UINT32 | Depends on MeParType. | | |
| Rx Data Example | 00 | 01 | 01 | 1 | FF800000 | 7F800000 | 420BF648 |
| Decoded Example | FLOAT32 | Read OK | 1 | 1 | -Infinity | +Infinity | 34.99051 |

Details to the Example:

- MeParType is FLOAT32, this means we must interpret the Min, Max, and Actual value as FLOAT32
- Flags says only reading is allowed for this parameter.
- Max Nr of Instances says that only one instance available for this parameter.
- Max Nr of Elements says that this parameter has only one element. There are parameters that have arrays of elements.
- Minimum Value interpreted as FLOAT32 is: -Infinity (negative infinity)
- Maximum Value interpreted as FLOAT32 is: +Infinity (positive infinity)
- Actual Value interpreted as FLOAT32 is: 34.99051 (in this case 34.99051°C for TEC)

3.2.4 Get Big Data “?VB”

This command is used to query larger data elements like temperature log data arrays or some text elements. The example of this command queries the Display Format Argument String Default Text (TEC Controller Family). Parameter 6024 Instance 1. The read start position is specified with 0 and the maximum answer length is 256 elements.

Query Payload:

| | | | | | |
|------------------------|---------------|---------|----------|---------------------|--------------------------------|
| PayloadField | Command | MeParID | Instance | Read start position | Max number of elements to read |
| DataFieldType | 3 ASCII Chars | UINT16 | UINT8 | UINT32 | UINT16 |
| Tx Data Example | ?VB | 1788 | 01 | 00000000 | 0100 |
| Decoded Example | ?VB | 6024 | 1 | 0 | 256 |

Response Payload:

| | | | |
|------------------------|-------------------------|--------------------|--|
| PayloadField | Received nr of elements | Has more data flag | Data |
| DataFieldType | UINT16 | UINT8 | Nr of elements of FLOAT32, INT32, DOUBLE64, LATIN1 or BYTE |
| Rx Data Example | 0015 | 00 | 54 65 6D 70 3A 20 7B 31 30 30 30 3B 31 3B 33 3B 38 7D B0 43 00 |
| Decoded Example | 21 | No | Temp: {1000;1;3;8}°C |

Definition of “Element”:

The fields “Read start position”, “Max number of elements to read” and “Received nr of elements” specify the number of FLOAT32, INT32, DOUBLE64, LATIN1 or BYTE elements and not the number of bytes.

The type LATIN1 or BYTE use two ASCII Chars from 0 to F to describe the content.

Length use cases:

- The host might send a low value for “Max number of elements to read” from the device. For example, 10 elements. In this case the device sends only 10 elements per package and sets the “Has more data flag” as long as the end has not been reached. In this case the host should set “Read start position” in the first package to 0 and in the next package to 10...
- The host might set “Max number of elements to read” to a high value. For example, the meComAPI for .Net sends FFFF for this field. In this case, the device will only send around 256 elements per package, because of the limited buffer size. The behavior to get all data is the same as above.
- It is most efficient if the whole data can be transferred in one package, therefore it is recommended to set “Max number of elements to read” to a high value and let the device decide how many elements can be sent in one package.

3.2.5 Set big Data “VB”

This command is used to set bigger data elements like a “Display Format Argument String” or a Lookup Table.

The following example sets a “Display Format Argument String” to the device. This is a feature, which is available for our TEC Controller Family. Currently not yet available for LTC-1141.

Set Payload:

| PayloadField | Command | MeParID | Instance | Write start position | Nr of elements in this package | Is last package | Data |
|-----------------|---------------|---------|----------|----------------------|--------------------------------|-----------------|--|
| DataFieldType | 2 ASCII Chars | UINT16 | UINT8 | UINT32 | UINT16 | UINT8 | Nr of elements of FLOAT32, INT32, DOUBLE64, LATIN1 or BYTE |
| Tx Data Example | VB | 1788 | 1 | 00000000 | 000C | 01 | 48 65 6C 6C 6F 20 57 6F 72 6C 64 00 |
| Decoded Example | VB | 6024 | Line 1 | 0 | 12 | Yes | Hello World |

Response:

- ACK if everything is ok.
- Various server error codes. In case of BUSY, just send it again.

Length use cases:

- In case of LTC-1141, similar as the Get Big Data Command
- In case of the TEC Controller Family devices, there are some restrictions:
 - The maximum allowed data length is only 232 bytes.
 - These maximum 232 data bytes must be sent in one package, therefore “Write start position” must always be 0 and “Is last package” must always be set.

Nr of elements in this package

- If the data Type is LATIN1, the zero terminator is also counted.

3.2.6 Read Bulk Value “?VX”

There is a command that can query up to TBD parameter values in one command. This massively reduces the protocol overhead. Please ask us to give you the specs if you need it.

3.2.7 Read Limits for the parameter “?VL”

This is an old command and is only implemented for compatibility. It is recommended to use ?VM.

This command reads the limits of a parameter. In this example the limits of the parameter 3000 instance 1 is read.

Query Payload:

| | | | |
|------------------------|---------------|---------|----------|
| PayloadField | Command | MeParID | Instance |
| DataFieldType | 3 ASCII Chars | UINT16 | UINT8 |
| Tx Data Example | ?VL | BB8 | 01 |
| Decoded Example | ?VL | 3000 | CH1 |

Response Payload:

| | | | |
|------------------------|-----------------------|-----------------------|---------------|
| PayloadField | MeParType Enumeration | Minimum Value | Maximum Value |
| DataFieldType | UINT8 | Depends on MeParType. | |
| Rx Data Example | 0 | C3888000 | 447A0000 |
| Decoded Example | FLOAT32 | -273 | 1000 |

4 C code examples

4.1 Send Frame function

This Function is used to send a Frame to the Device.

- ucAddress is the Device Address
- usSequenceNr is the random sequence Number which will be returned by the Device
- ucLength is the length of the payload field
- ucData is the pointer to the payload data

```
void UART1_SendFrame(unsigned char ucAddress, unsigned short usSequenceNr, unsigned
char ucLength, unsigned char *ucData)
{
    const unsigned char ucHex[16] =
    {'0','1','2','3','4','5','6','7','8','9','A','B','C','D','E','F'};
    unsigned char uc; unsigned short usCRC = 0;

    uc = 0x21;                UART1_Send(uc); CRC16Algorithm(&usCRC, uc);
    uc = ucHex[ucAddress / 16];    UART1_Send(uc); CRC16Algorithm(&usCRC, uc);
    uc = ucHex[ucAddress % 16];    UART1_Send(uc); CRC16Algorithm(&usCRC, uc);

    uc = ucHex[usSequenceNr/4096];    UART1_Send(uc); CRC16Algorithm(&usCRC, uc);
    uc = ucHex[(usSequenceNr/256)%16];    UART1_Send(uc); CRC16Algorithm(&usCRC, uc);
    uc = ucHex[(usSequenceNr%256)/16];    UART1_Send(uc); CRC16Algorithm(&usCRC, uc);
    uc = ucHex[(usSequenceNr%256)%16];    UART1_Send(uc); CRC16Algorithm(&usCRC, uc);

    for(uc = 0; uc < ucLength; uc++)
    {
        UART1_Send(*ucData);
        CRC16Algorithm(&usCRC, *ucData);
        ucData++;
    }
    UART1_Send(ucHex[usCRC/4096]);
    UART1_Send(ucHex[(usCRC/256)%16]);
    UART1_Send(ucHex[(usCRC%256)/16]);
    UART1_Send(ucHex[(usCRC%256)%16]);
    UART1_Send(0x0D);
}
```

4.2 Frame CRC Algorithm

The used standard is: CRC-CCITT (CRC-16)

```
void CRC16Algorithm(unsigned short *CRC, unsigned char Ch)
{
    unsigned int genPoly = 0x1021; //CCITT CRC-16 Polynomial
    unsigned int uiCharShifted = ((unsigned int)Ch & 0x00FF) << 8;
    *CRC = *CRC ^ uiCharShifted;
    for (int i = 0; i < 8; i++)
        if ( *CRC & 0x8000 ) *CRC = (*CRC << 1) ^ genPoly;
        else *CRC = *CRC << 1;
    *CRC &= 0xFFFF;
}
```