|  |  |  |  |
| --- | --- | --- | --- |
| Customer: | Project: | Date: | Confidentiality |
|  |  |  |  |

mSSP MULTI SLAVE Serial Protocol SPECIFICATION

Sisällys

[1 Introduction 2](#_Toc198645933)

[1.1 Purpose 2](#_Toc198645934)

[1.2 Glossary 2](#_Toc198645935)

[2 System Description 3](#_Toc198645936)

[3 MULTI SLAVE Serial Protocol 4](#_Toc198645937)

[3.1 MSSP Frames 4](#_Toc198645938)

[3.1.1 COBS 4](#_Toc198645939)

[3.1.2 MSSP MESSAGE STRUCTURE 6](#_Toc198645940)

[3.1.3 Control byte 7](#_Toc198645941)

[3.1.4 CRC-16 7](#_Toc198645942)

[3.1.5 Endianess 7](#_Toc198645943)

[3.2 Master – Slave communication 7](#_Toc198645944)

[3.2.1 Slave / slave group address 8](#_Toc198645945)

[3.2.2 Request, Indication and response message 8](#_Toc198645946)

[3.2.3 Messages to a single slave 9](#_Toc198645947)

[3.2.4 Messages to slave group 9](#_Toc198645948)

[3.2.5 Messages to all slaves 9](#_Toc198645949)

[3.2.6 Repeated message 9](#_Toc198645950)

[3.2.7 Corrupted message 9](#_Toc198645951)

[4 Messages 10](#_Toc198645952)

[4.1 MSG\_DEVICE\_INFO\_REQ 10](#_Toc198645953)

[4.2 MSG\_DEVICE\_INFO\_RESP 10](#_Toc198645954)

[4.2.1 Device type 10](#_Toc198645955)

[4.3 MSG\_GET\_SINGLE\_PARAM\_REQ 11](#_Toc198645956)

[4.4 MSG\_GET\_SINGLE\_PARAM\_RESP 12](#_Toc198645957)

[4.5 MSG\_GET\_LIGHT\_VALUE\_REQ 12](#_Toc198645958)

[4.6 MSG\_GET\_LIGHT\_VALUE\_RESP 12](#_Toc198645959)

[5 Commands 14](#_Toc198645960)

[6 Results 15](#_Toc198645961)

[7 DOCUMENT INFORMATION 16](#_Toc198645962)

1. Introduction

This document provides protocol specification for the Multi Slave Serial Protocol (MSSP), which is used to control Teknoware Smart Lighting accessories.

* 1. Purpose

The aim of the document is to describe the MSSP system, the required hardware and functionality of the protocol.

* 1. Glossary

Table 1. Definitions, acronyms and abbreviations

| Term | Definition |
| --- | --- |
| MSSP | Multi Slave Serial Protocol |
| CRC | Cyclic Redundancy Check |
| RS-485 | Serial communication interface |

1. System Description

MSSP system consists of one master device and n slave devices. The slave devices are connected to RS-485 linear bus network. Only two wires are used for data communication, so communication is half duplex.

Communication is always initiated by the master.

1. MULTI SLAVE Serial Protocol
   1. MSSP Frames

All MSSP communication is done in frames, where each frame has a start of frame and end of frame bytes. This makes it simple for receiver to detect, when full message has been received. In MSSP 0x00 byte is used for both start of frame and end of frame bytes.

Since 0x00 can’t be used in real data part of the frame, the data must be encoded to remove all the 0x00 from real data. Constant overhead byte stuffing (COBS) encoding algorithm is used to remove all 0x00 bytes from the data.

Receiver uses COBS decoding algorithm to get the real data in the received frame.

MSSP frame structure is shown in Figure 2.

0x00

COBS ENCODED MESSAGE

0x00

Figure 2. MSSP Frame Structure

* + 1. COBS

Consistent Overhead Byte Stuffing (COBS) is an algorithm for encoding data streams. The algorithm replaces each zero data byte with a non-zero value so that no zero data bytes will appear in the encoded data.

Encoded data length is always longer than original data, since COBS algorithm needs at least one overhead byte for encoding. The overhead of the COBS algorithm is very low compared to other byte stuffing algorithms, since it needs maximum of one overhead byte for each 254 bytes of data.

Consequently, the time to transmit the encoded byte sequence is highly predictable, which makes COBS useful for real-time applications in which jitter may be problematic.

COBS does, however, require up to 254 bytes of look ahead buffer, before transmitting its first byte, since it needs to know the position of the first zero byte (if any) in the following 254 bytes.

COBS algorithm works similarly as linked list. One extra byte is added to the beginning of data and this byte value indicates the location of the first zero in the data, then all zeroes in the data are replaced by value indicating location of the next zero data. If there are none zero values in following 254 bytes, an extra byte must be added to data after these 254 non-zero bytes and encoder replaces the current zero byte with value 255.

Decoding is pretty simple, since only the 0x00 bytes in the original data has been replaced by another value. Decoder uses the first encoded byte to know, which is the first byte that needs to be replaced with zero. Then it uses the byte that it replaced with zero to know the next location of the byte to replace with zero, etc. The value 255 has a special meaning, since it means that decoder does not replace the value in next location (after 255 bytes) with zero, but discards it from decoded data and only uses its value as location of the next zero value.

Encoding examples

These examples show how various data sequences would be encoded by the COBS algorithm. In the examples, all bytes are expressed as hexadecimal values, and encoded data is shown with text formatting to illustrate various features:

* **Bold** indicates a data byte which has not been altered by encoding. All non-zero data bytes remain unaltered.
* Green indicates a zero data byte that was altered by encoding. All zero data bytes are replaced during encoding by the offset to the following zero byte (i.e. one plus the number of non-zero bytes that follow). It is effectively a pointer to the next packet byte that requires interpretation: if the addressed byte is non-zero then it is the following group header byte zero data byte that points to the next byte requiring interpretation; if the addressed byte is zero then it is the end of packet.
* Red is an overhead byte which is also a group header byte containing an offset to a following group, but does not correspond to a data byte. These appear in two places: at the beginning of every encoded packet, and after every group of 254 non-zero bytes.
* A blue zero byte appears at the end of every packet to indicate end-of-packet to the data receiver. This packet delimiter byte is not part of COBS proper; it is an additional framing byte that is appended to the encoded output.

Table 4. Encoding examples.

|  |  |  |
| --- | --- | --- |
| Example | Unencoded data (hex) | Encoded with COBS (hex) |
| 1 | 00 | 01 01 00 |
| 2 | 00 00 | 01 01 01 00 |
| 3 | 11 22 00 33 | 03 **11 22** 02 **33** 00 |
| 4 | 11 22 33 44 | 05 **11 22 33 44** 00 |
| 5 | 11 00 00 00 | 02 **11** 01 01 01 00 |
| 6 | 01 02 03 ... FD FE | FF **01 02 03 ... FD FE** 00 |
| 7 | 00 01 02 ... FC FD FE | 01 FF **01 02 ... FC FD FE** 00 |
| 8 | 01 02 03 ... FD FE FF | FF **01 02 03 ... FD FE** 02 **FF** 00 |
| 9 | 02 03 04 ... FE FF 00 | FF **02 03 04 ... FE FF** 01 01 00 |
| 10 | 03 04 05 ... FF 00 01 | FE **03 04 05 ... FF** 02 **01** 00 |

For more details of COBS algorithm see: <https://en.wikipedia.org/wiki/Consistent_Overhead_Byte_Stuffing>

* + 1. MSSP MESSAGE STRUCTURE

MSSP message structure is shown in Figure 3. Each message has a header, which contains version, control, address and command parameters. Message can also have data part, but that is optional, so message can be sent without any data content. The last part of the message is 16-bit CRC, which is calculated for the whole message starting from control parameter and ending to the last byte of data.

Address parameter contains always the slave address or slave group address. Master address is never used, so all messages that master sends contain the address of the receiver. All messages that slaves send contain the address of the sender slave address.

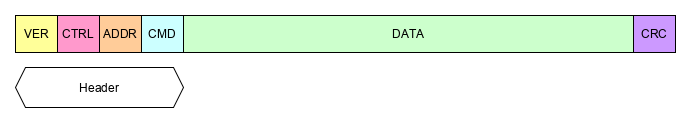


Figure 3. Message Structure

Table 5. Explanation of the frame structure.

| Abbreviation | Type | Explanation |
| --- | --- | --- |
| VER | uint8 | Bits 7-4: Reserved  Bits 3-0: Protocol version  0x0 = NOT USED, so we are able to read version without COBS decoding.  0x1 = version 1 (currently used) |
| CTRL | uint8 | Bit 7: Reserved  Bit 6: Master/Slave message  0 = slave sent this message  1 = master sent this message  Bit 5-4: Reserved  Bits 3-0: Message counter  counter from 1 to 15 Counter starts again from 1 after 15.  Note! 0 is never used as counter value to ensure that at least one bit is   always set, so control byte can be read without COBS decoding. |
| ADDR | uint8 | Slave address or Slave group address.  Address 0 is not allowed, so address byte can be read without COBS decoding. |
| CMD | uint8 | Command |
| DATA | 0-N bytes | Data |
| CRC | uint16 | CRC-16 Checksum |

* + 1. Control byte

Master/Slave message bit indicates, who sent the message. The bit is set, when master sends the message. The bit is not set, when slave sends the message.

Message counter is counter from 1 to 15, which rolls over to 1, when 15 is reached. Master increases this counter every time, when it sends a message. The counter is not increased in repeated message. Slave does not increase the counter, so slave sends the message with the same counter value as it received the request message. 0 is never used as counter value to ensure that at least on bit is always set, so control byte can be read without COBS decoding.

* + 1. CRC-16

Checksum is calculated for the whole message structure. 16-bit CRC algorithm CRC16-CCITT is used for checksum calculation. If the CRC doesn’t match the data packet it causes packet to be discarded. The CRC follows the specification from <http://srecord.sourceforge.net/crc16-ccitt.html>, and contains the parameters:

* Width = 16 bits
* Truncated polynomial = 0x1021
* Initial value = 0xFFFF
* Input data is NOT reflected
* Output CRC is NOT reflected
* No XOR is performed on the output CRC
  + 1. Endianess

Little endian format is used for MSSP protocol.

* 1. Master – Slave communication

All communication between Master and Slave devices are initiated by master. Master sends request message to slave device and slave device sends response message back to the master.

* + 1. Slave / slave group address

All slaves have individual address, which is set by master, when system is started first time. Several slaves can be controlled with one message using slave group addresses. Each slave is part of one slave group address, which is also set, when system is started first time. Slave and slave group addresses are defined in Table 7.

Table 7. Slave / slave group address definitions.

|  |  |
| --- | --- |
| Address | Number |
| Slave | 0x01 – 0x64 |
| Spare | 0x65 – 0x7B |
| Access through service port | 0x7C |
| Light sensor without force pins | 0x7D |
| Production test address | 0x7E |
| Slave in forced state | 0x7F |
| Slave group 1 | 0x80 |
| Slave group 2 | 0x81 |
| … | … |
| Slave group 64 | 0xBF |
| FW update group 1 | 0xC0 |
| FW update group 2 | 0xC1 |
| … |  |
| FW update group 16 | 0xCF |
| Spare | 0xD0 – 0xFD |
| Unset address | 0xFE |
| Broadcast to all slaves | 0xFF |

* + 1. Request, Indication and response message

There are two types of messages: request and response messages.

Only master can send request messages and only slaves can send response messages.

Request message is used, when master needs to query something from the slave. Slave must always send response message back to master, when it receives a request message.

* + 1. Messages to a single slave

Master can control single slave by using slave’s address as address in the message. Master can send request messages to a single slave.

* + 1. Messages to slave group

Master can send messages to a group of slave devices by using group address as address in the sent message.

* + 1. Messages to all slaves

Master can send messages to all slaves in the bus by using broadcast address 0xFF as address in the sent message.

* + 1. Repeated message

Master can re-send same message to the slave(s). Message counter is not added in repeated message.

Slave recognised repeat message by using

* Message counter value
* Message command
* Message CRC

Slave first checks the value of the message counter. If the message counter value is not incremented, the slave executes checks for the received command and the CRC value checks. They should be the same as in the previous received message.

* + 1. Corrupted message

If slave receives a message, which has non-matching CRC, slave just discards the message. No response message will be sent.

1. Messages
   1. MSG\_DEVICE\_INFO\_REQ

This message requests slave to send its device id, group address and firmware info to master.

Data: None

* 1. MSG\_DEVICE\_INFO\_RESP

Slave device sends its unique 32-bit device ID, device type and firmware version with this message.

Data:

|  |  |  |
| --- | --- | --- |
| Parameter | Size | Description |
| device\_type | uint32 | Device type |
| device\_id | uint32 | Device id |
| firmware\_version | uint32 | Firmware version X.Y.Z:  xxxx 0000 0000 0000 = X (0 – 255)  0000 xxxx 0000 0000 = Y (0 – 255)  0000 0000 xxxx xxxx = Z (0 – 65535) |
| addr | uint8 | Slave address |
| group | uint8 | Group address |

* + 1. Device type

Device type is product specific ID, which can be used for example to verify that the SW to be updated is for correct device. Device type can be added to firmware update package header, so programming SW can verify that device type in device and device type in SW package are identical.

Table 8. Currently decided addresses

| Device type | Device description |
| --- | --- |
| 0x001xxxxx  xxxxx = yyyy in hex | Light sensor – TSAyyyy |
| 0x002xxxxx  xxxxx = yyyy in hex | LCU – TSByyyy |
| 0x003xxxxx  xxxxx = yyyy in hex | Emergency light unit - TWTyyyy |
|  |  |
| Legacy values | These were defined, before the current numbering scheme was taken into use. |
| 0x00001000 | Light sensor – TSA0002x |
| 0x00006001 | LCU – TSB0711 |
| 0x3555434C | LCU – TSB0701 |
| 0x32425354 | LCU – TSB0702 |
| 0x35425354 | LCU – TSB0710 |
| 0x30474643 | LCU – TSB0701 (configuration file) |
| 0x31474643 | LCU – TSB0702 (configuration file: scenarios or active light control) |
| 0x00006002 | LCU – TSB0902 |
| 0x00006003 | LCU – TSB0903 |

* 1. MSG\_GET\_SINGLE\_PARAM\_REQ

Requests slave to send single parameter value.

Data:

| Parameter | Type | Description |
| --- | --- | --- |
| param\_type | uint8 | Parameter type |
| param\_number | uint8 | Parameter number |

* 1. MSG\_GET\_SINGLE\_PARAM\_RESP

Slave device sends value to requested parameter.

Data:

|  |  |  |
| --- | --- | --- |
| Parameter | Size | Description |
| param\_type | uint8 | Type of the parameter |
| param\_number | uint8 | Number of parameter type. For example, if param\_type is PWM, using 1 in param\_number, means that parameter is PWM1. Using 2 means PWM2, etc.  Note that param\_number starts from 1, i.e. typically value 1 is used here. |
| param\_value | 2 bytes | parameter value. |

**NOTE!** If the parameter type is not supported by the requested device, it will respond with MSG\_GET\_SINGLE\_PARAM\_RESP message where:

* param\_type = 0xFF (= Unknown parameter)
* param\_number = 0
* param\_value = 0
  1. MSG\_GET\_LIGHT\_VALUE\_REQ

Sent by LCU device to light sensor to get light reading.

Data:

|  |  |  |
| --- | --- | --- |
| Parameter | Size | Description |
| number of samples | uint8 | Number of samples to be included into calculation of average light value. |
| raw flag | uint8 | 0 = use calibration factor to calculate average light value  1 = return raw light value, without calibration factor. This functionality is for sensor calibration system only! |

* 1. MSG\_GET\_LIGHT\_VALUE\_RESP

Light sensor responds by this message to light request message.

Data:

|  |  |  |
| --- | --- | --- |
| Parameter | Size | Description |
| sensor value | uint16 | calculated average sensor (raw) value of n-samples. (n = number of samples in request message) |
| average value | uint16 | normalized (0..1000) value, calculated of calibrated sensor\_values |
| last value | uint16 | last measure value 0..1000, calculated of calibrated sensor\_value |

1. Commands

0x00 MSG\_DEVICE\_INFO\_REQ/RESP

0x0A MSG\_GET\_LIGHT\_VALUE\_REQ/RESP

0x0C MSG\_GET\_SINGLE\_PARAM\_REQ/RESP

1. Results

0x00 = TW\_MSSP\_OK

0x01 = TW\_MSSP\_INVALID\_CHANNEL

0x02 = TW\_MSSP\_INVALID\_COMMAND

0x03 = TW\_MSSP\_INVALID\_LENGTH

0x04 = TW\_MSSP\_INVALID\_CRC

0x05 = TW\_MSSP\_NO\_MSG

0x06 = TW\_MSSP\_MSG\_NOT\_FOR\_ME

0x07 = TW\_MSSP\_BUSY

0x08 = TW\_MSSP\_NOT\_IMPLEMENTED

0x09 = TW\_MSSP\_ERASE\_FAILED

0x0A = TW\_MSSP\_ALIGNMENT\_ERROR

0x0B = TW\_MSSP\_INVALID\_ACTIVE\_FW

0x0C = TW\_MSSP\_MESSAGE\_LOST

0x0D= TW\_MSSP\_FW\_UPDATE\_FAILED

0x0E = TW\_MSSP\_LAST\_FRAME\_NOT\_RECEIVED

0x0F = TW\_MMSP\_DATA\_NOT\_FOR\_ERASED\_PARTITION

0x10 = TW\_MSSP\_PARAMETER\_SIZE\_TOO\_LARGE

0x11 = TW\_MMSP\_FLASH\_WRITE\_FAILED

0x12 = TW\_MSSP\_WRONG\_DEVICE\_TYPE,

0x13 = TW\_MSSP\_NOT\_IN\_CORRECT\_STATE,

0x14 = TW\_MSSP\_FW\_CRC\_FAILED,

0x15 = TW\_MSSP\_TIMEOUT\_TIMER\_ON,

0x16 = TW\_MSSP\_RESPONSE\_TIMEOUT,

0x17 = TW\_MSSP\_SEND\_FAILED

0x18 = TW\_MSSP\_INVALID\_PROTOCOL\_VERSION

0x19 = TW\_MSSP\_REPEAT\_MSG\_SEND

0x1A = TW\_MSSP\_MSG\_SEND

1. DOCUMENT INFORMATION

|  |  |  |
| --- | --- | --- |
| Project: | Teknoware document reference: | |
| MSSP | MSSP Specification | |
| Customer: | Customer document reference: | Confidentiality: |
| TEKNOWARE OY | XXXXXXXXXXX | TEKNOWARE CONFIDENTIAL |
| Prepared by: | Checked by: | Approved by: |
|  |  |  |
| Revision: | Pages: | Date: |
| 12 | 28 | 2.9.2021 |

Revision follow-up

|  |  |  |  |
| --- | --- | --- | --- |
| Revision: | Purpose: | Date: | Author: |
| 0 | Document created | 28.5.2019 | Markku Vorne |



**Teknoware Oy**

Ilmarisentie 8, FI-15200 Lahti, Finland

Tel: +358 883 020

teknoware.com