ICOtronic Documentation

MyTooliT

Contents

1 Documentation **1**

2 Download **2**

3 Build **2**

4 Akronyms **2**

5 Terms **2**

6 MyTooliT Communication Protocol **3**

6.1 Introduction 3

6.2 Protocol Specification 4

7 Commands **8**

7.1 Blocks 8

7.2 Block `System` 9

7.3 Block `Streaming` 18

7.4 Block `Statistical Data and Quantity` 21

7.5 Block `Configuration` 22

7.6 Block `EEPROM` 29

7.7 Block `ProductData and RFID` 29

7.8 Block `Test` 32

7.9 Errors 33

1 Documentation

This repository collects various documentation for the ICOtronic system.

2 Download

You can access this repository and its submodules using the following command:

```
git clone --recursive git@github.com:MyTooliT/Documentation.git
```

3 Build

You can use bookdown to generate

- HTML,
- EPUB, and
- PDF

versions of this documentation. To do that please use the following make commands:

```
# Generate HTML documentation
make html

# Generate PDF
make pdf

# Generate EPUB document
make epub
```

4 Akronyms

- **AEM**: Advanced Energy Monitoring
- **BP**: Byte Position
- **CAN**: Controller Area Network
- **CAN-FD**: CAN Flexible Data Rate
- **CSMA/CD**: Carrier Sense Multiple Access/Collision Detection
- **CSMA/CR**: Carrier Sense Multiple Access/Collision Resolution
- **DLC**: Data Length Code
- **ECU**: Electronic Control Units
- **ESD**: Electro Statical Discharge
- **GD1**: Graceful Degradation Level 1
- **GD2**: Graceful Degradation Level 2
- **MSB**: Most Significant Byte
- **SHA**: Sensory Holder Assembly
- **STH**: Sensory Tool Holder
- **STU**: Stationary Transceiver Unit

5 Terms

- **Event (Message)**: Even messages transport information about signals and events/states
- **Header**: Supplemental data placed at the beginning of a block
- **Jitter**: Difference between best-case time and worst-case time

- **Node**: Self-contained unit that interacts with other nodes via the MyTooliT communication protocol
- **Payload**: Transmitted user data
- **Trailer**: Terminating part of a message; May support check functionality

6 MyTooliT Communication Protocol

This document defines the MyTooliT network protocol. The MyTooliT network protocol exchanges information over data link layers like Bluetooth or Controller Area Network (CAN).

CAN (2.0) logically splits a message into

- a payload, and
- an identifier.

The identifier contains

- a sender field to define the node of origin of each message,
- a receiver field to define a message receiver, and
- the command number to
  - specify actions,
  - answers to actions,
  - or specify errors.

Each command, defined by its number, will be acknowledged via the same command number. A

- request bit defines request (acknowledgement) commands, and
- an error bit defines errors.

Please note that errors must not requested.

The MyTooliT communication protocol may also exchange information via Bluetooth. For that purpose CAN messages will be stored into the payload of a data link layer like Bluetooth. The identifier field is handled via a 4 byte header and the payload by an additional payload that follows each message header. Note that a message may have a larger payload than 8 bytes (up to 64 Bytes per message as defined by the CAN-FD specification) but the length is limited to 8 bytes, if CAN 2.0 is used in the transport chain.

The MyTooliT protocol can also use other data link layer formats like CAN-FD. For example, you can use the protocol for IP application because it is an end-to-end based network protocol.

6.1 Introduction

CAN was introduced by BOSCH in the 1980s in the automotive industry to exchange short real time messages between Electronic Control Units (ECU). Each ECU may act as a master i.e. send frames and thus each ECU may control the system

- by inserting error frames,
- acknowledging frames,
- sending information or
- processing information.

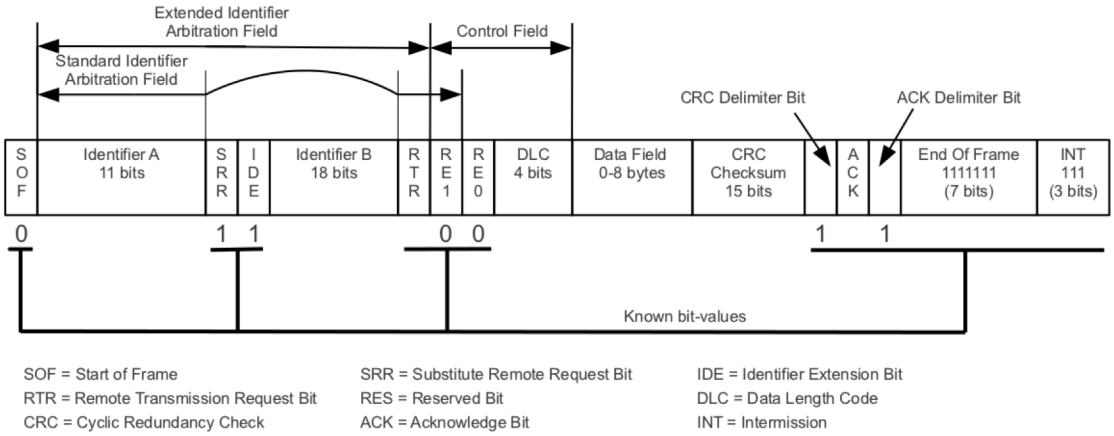

Figure 1: CAN Frame

A standard base format (11 bit identifier) and an extended format (29 bit identifier) exist. The MyTooliT communication protocol is based on the extended format. The following figure describes the extended format:

For more information, please take a look at the Wikipedia article about CAN or other available literature (e.g. Experimental Framework for Controller Area Network based on a Multi-Processor-System-on-a- Chip).

A main feature of CAN are prioritized messages i.e. if two or more senders try to send messages simultaneously, the message with the highest priority (lowest identifier) will be sent instantly and the remaining ones afterwards (CSMA/CR).

This design requires that each message identifier must be unique (each sender has a set of messages) and subscribers must queue messages according to their priority.

The priority-based concept of messages is a key feature of the MyTooliT network protocol. The protocol uses CAN 2.0, Bluetooth and other data link layer protocols to transport messages between end nodes. Thus, MyTooliT transport messages between end nodes over diverse data link protocols. The flow control is managed by the prioritization of messages, the end-to-end-communication and by limiting the overall traffic to 40%/60% of the total bandwidth.

6.1.1 Reserved Bits

Reserved Bits must be transmitted as `0`. This is required for compatibility.

6.2 Protocol Specification

Each CAN 2.0 frame consists of

- an identifier,
- a payload,
- a data length code (DLC), and
- physical transport bits.

The following figure shows the essential parts of an extended CAN 2.0 frame:

Identifier	DLC	Payload
29 Bits	4 Bits	0 – 8 Bytes

The

- identifier describes the message,
- the data length code stores the length of the payload (CAN 2.0: 0 – 8 Byte, CAN-FD 0 – 64 bytes), and
- the payload stores message data.

6.2.1 Identifier

	V	Command	R1	Sender	R2	Receiver
Bit	0	1 – 16	17	18 – 22	23	24 – 28

The following table describes the identifier field.

Field	Purpose
V	Version number • Must be `0` or the frame will be discarded
Command	Command to be executed or acknowledged
R1/R2	Reserved
Sender	Number of the original sender (frames may hop) • `0` Not allowed
Receiver	Number of the target receiver (frames may hop) • `0` broadcasts at field bus (local network) with ACK • `0x1F` broadcasts at field bus(local network) without ACK

6.2.2 Command

	Command Number	A	E
Bit	0 – 13	14	15

The command number contains the command block and the block command:

Block	Block Command
0 – 5	6 – 13

The following table describes the whole command field.

Field	Purpose
Command Number	64 command blocks (6 Bit) • A command block supports up to 256 (8 Bit) block commands • Values: 1 – 16383 (14 bit), `0` is not valid • Commands are described here
A	Acknowledge field • `1` for a request • `0` for an acknowledgement Note that a single command may trigger multiple acknowledges (streaming).

Field	Purpose
E	Error Bit • Indicates an error • `1` if it is an error • `0` if it is not an error • An error code is supported via the payload • The error format is 8 bytes long. The first byte describes the error number and the following 7 bytes are used for an error description. Furthermore, there are general errors (1 – 255) that are followed by `0` and specific errors that are followed by variable bits.

6.2.3 Abstracted CAN Messages

As mentioned in the introduction the MyTooliT protocol derives the priorities message concept from CAN 2.0. Therefore, the CAN header (identifier and DLC) are abstracted by a 4 byte header as described in the table below.

Note: The `DLC0` bit is at position 0 and the command resides in the 2 bytes at the highest addresses.

Bit	Name	Description
0 – 3	Data Length Code (DLC)	Length of message as described by the CAN-FD standard
4 – 8	Receiver	End subscriber to be addressed as described in the Section “Identifier”
9	Reserved	Reserved
10 – 14	Sender	End subscriber that sends message as described in the Section “Identifier”
15	Reserved	Reserved
16 – 31	Command	Command as described in Section “Command Field”

The transport of messages over a data link layer (except CAN 2.0) are fulfilled by putting messages consisting of header and payload in a row up to the length of the data link layer payload. Each node manage the prioritization of messages in each send queue by a prioritized message queue.

6.2.4 Addressing

A network consists of two or more subscribers and each subscriber use a unique number (1 – 30; 0 = Broadcast with ACK; 31 = Broadcast without ACK) called address. The address targets a specific subscriber (or all subscribers). Note that the send number is important for the acknowledgement.

This addressing scheme yields an end-to-end management of the communication state i.e. the internal states of elements inside the end-to-end subscribers do not influence the logical communication state. Thus, only a single channel must be supported for a MyTooliT information exchange i.e. an incoming message that does not address the subscriber is discarded or forwarded. This means the MyTooliT commands can be used over other communication protocols like Bluetooth. Note that the simultaneous transport via CAN 2.0 may not possible due to the replication of the sender and receiver (and the command) at the data link layer.

In the MyTooliT protocol the subscribers manage the error handling e.g. re-request something after a timeout. If that is not the case, then other counter measurements must be fulfilled.

The following figure shows the overall idea of network addressing.

6.2.5 DLC

The MyTooliT protocol uses the DLC as described the CAN-FD standard. The DLC must transfer over other protocols in the same format. Thus the DLC is limited by the data link layer i.e. requesting a command via CAN 2.0 and Bluetooth yields a limit of 8 bytes.

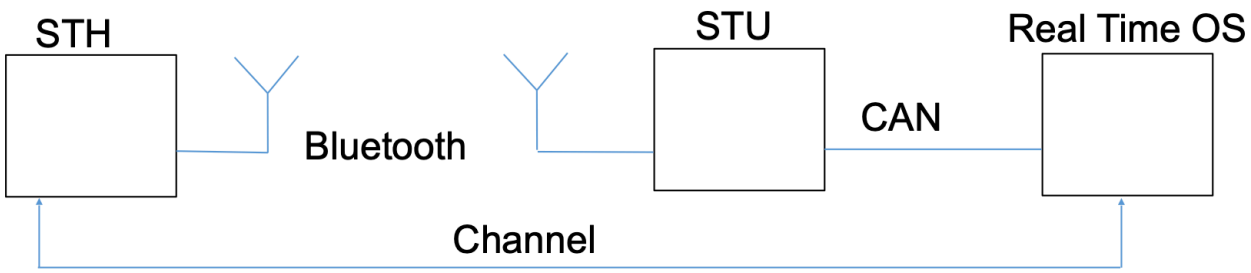

Figure 2: End To End Communication

6.2.6 Payload

The payload transports user data and/or sub-payloads.

6.2.7 Startup an Backup Strategy

This is currently not implemented. CAN transmits at 1 MBit gross (gross bitrate: bitrate including physical protocol overhead) and Bluetooth transmits the payload at 1Mbit net (net bitrate: bitrate excluding physical protocol overhead). Note, that Bluetooth is a CSMA/CD protocol that will cause jitter without taking any other actions. Collisions also reduce the total bandwidth.

6.2.8 Transmission Speed

The transmission speed depends of the supported data link layer formats.

6.2.9 Bluetooth

The actual Bluetooth transmissions speed is 1 MBit gross. However, a message transmission might be delayed due to CSMA/CD. CSMA/CD prevents transmission, if an ongoing transport is in process in the corresponding transport frequency interval. Each collision delays the transport time exponentially. Note that simultaneously sending or any radio interference may destroy any radio frame and the actual Bluetooth configuration avoids re-requests at the protocol stack level (application must do this).

Bluetooth supports a net bandwidth of about 700 kBit if each frame is 255 bytes long. However, Bluetooth applications supports a maximum net bandwidth of about 420 kbit/s.

6.2.10 CAN 2.0

The transmission speed should be aligned to a maximum of 40% of the total bandwidth. However, in any case there must not be any higher utilization than 60% of the overall bandwidth. In the case of fair message distribution with many nodes and many sporadic messages, the limit should be a utilization of 40%. In cases with many permanent messages the limit may be set to 60%.

The 40% utilization for CAN2.0 with bit stuffing is calculated as follows:

$$U = \frac{m \cdot 79 + \sum_{m=0}^{m} \left(8 \cdot p_m + \lfloor p_m \cdot \frac{8}{5} \rfloor\right)}{B}$$

Here

- B is the gross bandwidth per second (e.g. 1Mbit/s),
- m is the overall number of send messages per second,
- p_m the payload length in bytes for each message and
- U is the overall utilization.

The 60% utilization without bit stuffing is calculated as follows:

$$U = \frac{m \cdot 67 + \sum_{m=0}^{m} (8 \cdot p_m)}{B}$$

The 40% utilization for CAN-FD with bit stuffing is calculated as follows:

$$U = \frac{m \cdot 79}{B_{ID}} + \frac{\sum_{m=0}^{m} \left(8 \cdot p_m + \lfloor p_m \cdot \frac{8}{5} \rfloor\right)}{B_p}$$

Here

- B_{ID} is the gross identifier bandwidth per second (e.g. 1Mbit/s), and
- B_p is the gross payload bandwidth per second (e.g. 8 Mbit/s).

The 60% utilization for CAN-FD without bit stuffing is calculated as follows:

$$U = \frac{m \cdot 67}{B_{ID}} + \frac{\sum_{m=0}^{m} (8 \cdot p_m)}{B_p}$$

Thus the bandwidth consumption for a streaming message (64 bytes payload each ms) calculates as follows at 1Mbit/8Mbit:

$$U_{Stuff} = \frac{1000 \cdot 79}{1000000} + \frac{1000 \cdot (512 + 102)}{8000000} = 0.079 + 0.07675 = 0.15575(15.6\%)$$

and

$$U = \frac{1000 \cdot 67}{1000000} + \frac{1000 \cdot 512}{8000000} = 0.067 + 0.064 = 0.131(13.1\%)$$

Alarm messages – they will be periodically repeated until muted or alarm off event occurs e.g. temperature drops under a certain limit after reaching certain alarm limit – and streaming messages are periodic messages.

Sporadic messages trigger on demand e.g. setting a program status word requires a request and an acknowledgement. The acknowledgement and the request are sporadic messages.

Sporadic messages should have a reserved bandwidth of at least 10% (in an alarm shower case, the alarm messages will be prioritized). An overload case must be handled at the application level e.g. turn off all streaming messages and go to a graceful degradation state or a fail-save state. Note that time triggered communication eliminates such cases because each message transmission is pre-scheduled.

7 Commands

7.1 Blocks

Block	Short Description	Extended Description
0x00	System	System commands are used to modify/request the state of each unit (e.g. reset) or an the overall system state (e.g. transmission speed)
0x04	Streaming	Streaming commands are used to transmit data streams, but may be also used for single requests. The super frame is also located in this block.
0x08	Statistical Data and Quantity	This command group is used to store statistical data that can be used for histograms such as operating time and the number of power on/off cycles
0x28	Configuration	This command block is used to set configuration data (e.g. you can set the sampling rate of acceleration data here).
0x3D	EEPROM	Used for writing and reading EEPROM data directly
0x3E	ProductData and RFID	Used to store product data like a serial number. Furthermore, this block provides access to RFID information that is supported via connected tools.
0x3F	Test	Test Config Page

7.2 Block System

Number	Block Command	Access	Permanently Stored
0x00	Verboten	–	–
0x01	Reset	Event	–
0x02	Get/Set State	Read/Write	–
0x05	Get Node Status	Read/Write	–
0x06	Get Error Status	Read/Write	–
0x0B	Bluetooth	Read	–

7.2.1 Command Verboten

This command is mainly used for initialization purposes

7.2.2 Command Reset

Reset the specified receiver. This command has no payload.

7.2.3 Command Get/Set State

- Not fully implemented
- Startup state determines operating state
- Standby state works

7.2.3.1 Values

- Get/Set State:

Value	Meaning
0	Get State
1	Set State

- Location:

Value	Meaning
0	No Change
1	Bootloader
2	Application
3	Reserved

- State:

Value	Meaning
0	Failure (No acknowledgement will be sent; Only power on resets this state)
1	Error (No active communication)
2	Turn Off/Standby
3	Graceful degradation level 2
4	Graceful degradation level 1
5	Operating
6	Startup
7	No change

- Error Reason:

Value	Meaning
1	Set state not available
2	Wrong subscriber (e.g. accessing application as bootloader)

7.2.3.2 Payload

Byte 1				
Bit 7	Bit 6	Bit 5 – 4	Bit 3	Bit 2 – 0
Get/Set State	Reserved	Location	Reserved	State

7.2.3.3 Acknowledgment Payload

Byte 1
0x08

7.2.3.4 Error Payload

Byte 2
Error Reason

7.2.4 Command Get Node Status

- Note that the state may not be set instantly.
- The node status word is defined differently for STH and STU
- STH node status word:

```
typedef union
{
 struct
 {
  uint32_t bError :1; /**< Error or healthy */
  uint32_t u3NetworkState :3; /**< Which state has node in the network */
  uint32_t Reserved :28; /**< Reserved */
 };
 uint32_t u32Word;
 uint8_t au8Bytes[4U];
} NodeStatusWord_t;
```

- STU node staus word:

```
  struct
  {
    uint32_t bError :1; /**< Indicates an overall Error */
    uint32_t u3NetworkState :3; /**< Which state has node in the network */
    uint32_t bEnabledRadio :1; /**< Radio port enabled(1) or disabled(0) */
    uint32_t bEnabledCan :1; /**< CAN port enabled(1) or disabled(0) */
    uint32_t bRadioActive :1; /**< Radio Active(Connected to Bluetooth) or not */
    uint32_t Reserved :25; /**< Reserved */
  };
  uint32_t u32Word;
  uint8_t au8Bytes[4U];
} NodeStatusWord_t;
```

- Error Bit:

Value	Meaning
0	No Error
1	Error

- Network State:

Value	Meaning
0	Failure
1	Error
2	Standby
3	Graceful Degradation 2

Value	Meaning
4	Graceful Degradation 1
5	Operating
6	Startup
7	No Change

- `Radio Port`:

Value	Meaning
0	Radio Port Disabled
1	Radio Port Enabled

- `CAN Port`:

Value	Meaning
0	CAN Port Disabled
1	CAN Port Enabled

- `Radio Activity`:

Value	Meaning
0	Disconnected from Bluetooth
1	Connected to Bluetooth

7.2.4.1 Payload

- Setting the value `0` for the node status word mask means that we request the status word
- Currently the only supported payload should be 8 null (`0x00`) bytes

7.2.4.1.1 STH

Byte 1		
Bit 7 – 4	Bit 3 – 1	Bit 0
Reserved	`Network State`	`Error Bit`

7.2.4.1.2 STU

Byte 1					
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3 – 1	Bit 0
Reserved	`Radio Activity`	`CAN Port` Enabled	`Radio Port` Enabled	`Network State`	`Error Bit`

7.2.4.1.3 STH & STU

Byte 2
Reserved

Byte 3
Reserved

Byte 4
Reserved

Byte 5
Status Word Mask

Byte 6
Status Word Mask

Byte 7
Status Word Mask

Byte 8
Status Word Mask

7.2.4.2 Acknowledgement Payload

- Same structure as payload

7.2.4.3 Error Payload

- The (possibly incorrect) length of the status word (5 instead of 4 bytes) was taken from the original documentation.

Byte 1
Mask Used Not Allowed

Byte 4
Status Word

Byte 5
Status Word

Byte 6
Status Word

Byte 7
Status Word

Byte 8
Status Word

7.2.5 Command `Get Error Status`

- STH definition:

```
typedef union
{
 struct
 {
  uint32_t bTxBluetoothFail :1; /**< Tx Fail Counter for Bluetooth (non single set) */
  uint32_t bAdcOverRun :1;  /**< Determines ADC over run (not able to shuffle data in time) */
  uint32_t Reserved :30;
 };
 uint32_t u32Word;
 uint8_t au8Bytes[4U];
} ErrorStatusWord_t;
```

- STU definition:

```
typedef union
{
  struct
  {
      uint32_t bTxCanFail :1; /**< Tx Fail Counter for CAN (non single set) */
      uint32_t Reserved :31; /**< DAC was not fed */
  };
  uint32_t u32Word;
  uint8_t au8Bytes[4U];
} ErrorStatusWord_t;
```

- `Transmission Failure` (Bluetooth for STH, CAN for STU):

Value	Meaning
`0`	No Transmission Failure
`1`	Transmission Failure

- `ADC Overrun`:

Value	Meaning
`0`	No ADC Overrun Error
`1`	ADC Overrun Error

7.2.5.1 Payload

- Setting the value `0` for the status word mask means that we request the error status word
- Currently the only supported payload should be 8 null (`0x00`) bytes

7.2.5.1.1 STH

Byte 1		
Bit 7 – 2	Bit 1	Bit 0
Reserved	`ADC Overrun`	Bluetooth `Transmission Failure`

7.2.5.1.2 STU

Byte 1	
Bit 7 – 2	Bit 0
Reserved	CAN `Transmission Failure`

7.2.5.1.3 STH & STU

Byte 2
Reserved

Byte 3
Reserved

Byte 4
Reserved

Byte 5
Status Word Mask

Byte 6
Status Word Mask

Byte 7
Status Word Mask

Byte 8
Status Word Mask

7.2.5.2 Acknowledgement Payload

- Same structure as payload

7.2.5.3 Error Payload

- Same structure as error payload for node status command

7.2.6 Command `Bluetooth`

- `Bluetooth Subcommand`

Value	Meaning
`0`	Reserved
`1`	Connect
`2`	Get number of available devices
`3`	Write device name #1 and set device name #2 to `NULL`
`4`	Write device name #2 and push it to STH (read will be equivalent in the future)
`5`	Read first part (6 bytes) of device name
`6`	Read second part (2 bytes) of device name
`7`	Connect to device (with device number)
`8`	Check if connected
`9`	Disconnect
`10`	Get send counter
`11`	Received RX frames
`12`	Absolute RSSI indicator (number must be interpreted as negative)
`13`	Read energy mode reduced
`14`	Write energy mode reduced
`15`	Read energy mode lowest
`16`	Write energy mode lowest
`17`	Bluetooth MAC address
`18`	Connect to device (with Bluetooth MAC address)

- `Device Number`: Sequential positive number assigned by STU to available STH nodes; For a single STH this number will be `0`.
- `Bluetooth Value`

Bluetooth Subcommand	Value
0	–
1	–
2	–
3	ASCII string
4	ASCII string (NULL)
5	–
6	–
7	–
8	–
9	–
10	–
11	–
12	–
13	–
14	Byte 3 – 6: Time form normal to reduced energy mode in ms Byte 7 – 8: Advertisement time for reduced energy mode in ms Big endian
15	–
16	Byte 3 – 6: Time form reduced to lowest energy mode in ms Byte 7 – 8: Advertisement time for lowest energy mode in ms Little endian 0 = read
17	Byte 3: Value 255 (self addressing)

- Bluetooth Return Value

Bluetooth Subcommand	Value
0	NULL
1	NULL
2	ASCII string containing the number of available devices
3	ASCII string containing the first 6 characters of the Bluetooth advertisement name
4	ASCII string containing the last 2 characters of the Bluetooth advertisement name
5	ASCII string
6	ASCI string of connected device or NULL if not connected
7	NULL (Byte 3 is true if in search mode, at least single device was found, no legacy mode and scanning mode active; false otherwise)
8	NULL if not connected Not NULL otherwise
9	NULL
10	6 Byte unsigned int
11	6 Byte unsigned int
12	Byte 3: uint8_t deviceType Byte 4: int8_t Byte 5 – 8: NULL
13	Byte 3 – 6: Time form normal to reduced energy mode in ms Byte 7 – 8: Advertisement time for reduced energy mode in ms Big Endian
14	Byte 3 – 6: Time form normal to reduced energy mode in ms Byte 7 – 8: Advertisement time for reduced energy mode in ms Big Endian
15	Byte 3 – 6: Time form reduced to lowest energy mode in ms Byte 7 – 8: Advertisement time for lowest energy mode in ms Little Endian
16	Byte 3 – 6: Time form reduced to lowest energy mode in ms Byte 7 – 8: Advertisement time for lowest energy mode in ms Little Endian
17	Byte 3 – 6: Bluetooth MAC address in little endian format

7.2.6.1 Payload

Byte 1
`Bluetooth Subcommand`

Byte 2
`Device Number`

Byte 3 – 8
`Bluetooth Value`

Note: Use `0` bytes if Device Number or `Bluetooth Value` are not applicable (e.g. when you use the `Connect` command)

7.2.6.2 Acknowledgement Payload

Byte 1
Same as Payload

Byte 2
Same as Payload

Byte 3 – 8
`Bluetooth Return Value`

7.3 Block `Streaming`

Number	Block Command	Access	Permanently Stored
`0x01`	Acceleration	Event	–
`0x20`	Voltage	Event	–

7.3.1 Values

- The `Data Sets` bits used in the sections below can have the following values:

Value	Data Amount	Possible Data
0	Stop (stream)	• value 1 • value 2 • value 3 • value 1 / value 2 / value 3 • value 1 / value 2 • value 1 / value 3 • value 2 / value 3

Value	Data Amount	Possible Data
1	1 data set	
2	3 data sets	
3	6 data sets	
4	10 data sets	
5	15 data sets	
6	20 data sets	
7	30 data sets	• value 1 • value 2 • value 3

The chronological order starts with the oldest set (BP) and continues with newer values (BP + t), where t is the time point.

- `Request`:

Value	Meaning
`0`	Stream
`1`	Single Request

- `Bytes`:

Value	Meaning
`0`	2 Bytes for each data point
`1`	3 Byte for data point

- `Active`

Value	Meaning
`0`	Data for specified data point will not be measured/sent
`1`	Data for specified data point will be measured/sent

7.3.2 Command `Acceleration`

- Requesting while streaming is possible
- Only single stream allowed
- Requesting stream in different format stops last stream
- Tuple format (depending on active axis, see payload):
  - x/y/z
  - x/y
  - x/z
  - y/z

7.3.2.1 Payload

Byte 1					
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2 – 0
`Request`	`Bytes`	X-Axis `Active`	Y-Axis `Active`	Z-Axis `Active`	`Data Sets`

7.3.2.2 Acknowledgment Payload

Byte 1
Same as Payload

Byte 2
Sequence Counter

7.3.2.2.1 2 Byte Format

Byte 3
MSB (BP)

Byte 4
LSB (BP + 1)

7.3.2.2.2 3 Byte Format

Byte 3
MSB (BP)

Byte 4
(BP + 1)

Byte 5
LSB (BP + 2)

7.3.3 Command `Voltage`

7.3.3.1 Notes

- Highest voltage sampling rate determines bit stream rate
- Requesting while streaming is possible

7.3.3.2 Payload

Byte 1					
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2 – 0
`Request`	`Bytes`	Voltage 1 `Active`	Voltage 2 `Active`	Voltage 3 `Active`	`Data Sets`

7.3.3.3 Acknowledgment Payload The command uses the same format as the "Acknowledgment Payload" of the `Acceleration` command.

7.4 Block `Statistical Data and Quantity`

Number	Block Command	Access	Permanently Stored
`0x00`	Power On Cycles, Power Off Cycles	Read	x
`0x01`	Operating time	Read	x
`0x02`	Under Voltage Counter	Read	x
`0x03`	Watchdog Reset Counter	Read	x
`0x04`	Production Date	Read	x

7.4.1 Command `Power On Cycles, Power Off Cycles`

7.4.1.1 Notes

- Power off means power away e.g. Accumulator out of energy
- Power On includes resets

7.4.1.2 ACK Payload

Byte 1 (MSB) - Byte 4 (LSB)
Power On Cycles

Byte 5 (MSB) - Byte 8 (LSB)
Power Off Cycles

7.4.2 Command `Operating time`

7.4.2.1 Notes

- Seconds since first power are stored each half an hour
- The STH also stored seconds since reset in disconnect case.

7.4.2.2 ACK Payload

Byte 1 (MSB) - Byte 4 (LSB)
Seconds since reset

Byte 5 (MSB) - Byte 8 (LSB)
Seconds since first power on

7.4.3 Command `Under Voltage Counter`

7.4.3.1 ACK Payload

Byte 1 (MSB) - Byte 4 (LSB)
Under voltage counter since first power on

7.4.4 Command `Watchdog Reset Counter`

7.4.4.1 ACK Payload

Byte 1 (MSB) - Byte 4 (LSB)
Watchdog Resets since first power on

7.4.5 Command `Production Date`

7.4.5.1 ACK Payload

Byte 1 (MSB) - Byte 4 (LSB)
ASCII String of the Production Date in the format: yyyymmdd where y=year, m=month, d=day

7.5 Block `Configuration`

Number	Block Command	Access	Permanently Stored
`0x00`	Get/Set Acceleration Configuration	Read/Write	x
`0x60`	Get/Set Calibration Factor k	Read/Write	x
`0x61`	Get/Set Calibration Factor d	Read/Write	x
`0x62`	Calibration Measurement	Read/Write	x
`0xC0`	HMI Configuration	Read/Write	x

7.5.1 Command `Get/Set Acceleration Configuration`

7.5.1.1 Notes

7.5.1.1.1 Sampling Rate

$$\frac{f_{CLOCK}}{(Prescaler+1)\cdot(AcquisitionTime+12+1)\cdot OverSamplingRate}$$

$$f_{clock} = 38400000Hz$$

7.5.1.1.2 Prescaler

- Byte2

7.5.1.1.3 Acquisition Time

- Sample and Hold Time i.e. Time to charge capacitor that is cut off and measured at digital quantisation
- 2^(Byte3-1) iff Byte3 > AdcAcquisitionTime4
- (Byte3+2) iff Byte3 < =AdcAcquisitionTime4

7.5.1.1.4 Over Sampling Rate

- 2^Byte4; No Over Sampling if Byte4=0

7.5.1.1.5 ADC Reference Voltage

- 1V25
- 1V65
- 1V8
- 2V1
- 2V2
- 2V5
- 2V7
- 3V3(VDD)
- 5V
- 6V6

7.5.1.1.6 Setting at Reset

- 2/Aqu8(4)/OverSampling64(6)/VDD

7.5.1.2 Values

- `Get/Set State`:

Value	Meaning
`0`	Get State
`1`	Set State

7.5.1.3 Payload

Byte 1	
Bit 7	Bit 6 – 0
`Get/Set State`	Reserved

Byte 2
ADC Prescaler

Byte 3
Acquisition time (See Note)

Byte 4
Power of over sampling rate e.g. 10->1024 OverSampling Rate, 0=no Over Sampling

Byte 5
Reference: Voltage*20 e.g. 3.3V->66

Byte 6 - Byte 8
Reserved

7.5.1.4 Acknowledgment Payload

- Same structure as payload

7.5.2 Command `Get/Set Calibration Factor k`

7.5.2.1 Values

- `Calibration Element`:

Value	Meaning
`0`	Acceleration
`1`	Temperature
`32`	Voltage

- `Number or axis`:

Value	Meaning
`0`	Reserved
`1`	x-Axis / First measure point
`2`	y-Axis / Second measure point

Value	Meaning
3	z-Axis / Third measure point

- Get/Set Value:

Value	Meaning
0	Get Value
1	Set Value

7.5.2.2 Payload

Byte 1
Calibration Element

Byte 2
Number or axis

Byte 3	
Bit 7	Bit 6 – 0
Get/Set Value	Reserved

Byte 4
Reserved

Byte 5 (MSB) - Byte 8 (LSB)
k (Slope) according to IEEE 754 single precision (float)Calibration=kx+d (Also calculation to SI value or any other value)

7.5.2.3 Acknowledgment Payload

Byte 1
Calibration Element

Byte 2
Number or axis

Byte 3
Reserved

Byte 4
Reserved

Byte 5 (MSB) - Byte 8 (LSB)
k (Slope) according to IEEE 754 single precision (float)Calibration=kx+d (Also calculation to SI value or any other value)

7.5.3 Command `Get/Set Calibration Factor d`

Payload and Acknowledgment Payload have the same Structure as `Get/Set Calibration Factor k` but with `d (Offset)` instead of `k (Slope)` from `kx+d`.

7.5.4 Command `Calibration Measurement`

7.5.4.1 Values

- `Calibration Get/Set`:

Value	Meaning
`0`	Get (Ignores the remaining bits of this byte)
`1`	Set

- `Calibration Method`:

Value	Meaning
`0`	Reserved
`1`	Inject
`2`	Eject
`3`	Measure

- `Calibration Measurement Element`:

Value	Meaning
`0`	Acceleration
`1`	Temperature (for VREF=1250 temperature gets calculated to m°C)
`32`	Voltage
`96`	VSS(Ground)
`97`	VDD (Supply)
`98`	Regulated Internal Power
`99`	Operation Amplifier Output

- `Dimension`:

Value	Meaning
`0`	Reserved
`1`	1. Dimension(x)
`2`	2. Dimension(y)
`3`	3. Dimension(z)

7.5.4.2 Payload

Byte 1			
Bit 7	Bit 6 - Bit 5	Bit 4	Bit 3 - Bit 0
`Calibration Get/Set`	`Calibration Method`	Reset	Reserved

Byte 2
`Calibration Measurement Element`

Byte 3
`Dimension`

Byte 4
Reserved

Byte 5 - Byte 8
Reserved

7.5.4.3 Acknowledgment Payload

Byte 1 - Byte 4
Same as Payload

Byte 5 - Byte 8
Result

7.5.5 Command `HMI Configuration`

7.5.5.1 Values

- `Get/Set Sampling Rate`:

Value	Meaning
0	Get Sampling Rate
1	Set Sampling Rate

- LED:

Value	Meaning
0	Reserved
1	LED

- ON/OFF:

Value	Meaning
0	Reserved
1	On (Reset value)
2	Off

7.5.5.2 Payload

Byte 1	
Bit 7	Bit 6 - Bit 0
Get/Set Sampling Rate	LED

Byte 2
Number (0-255)

Byte 3
ON/OFF

Byte 4
Reserved

Byte 5 - Byte 8
Reserved

7.5.5.3 Acknowledgment Payload

- Same structure as payload

7.6 Block EEPROM

Number	Block Command	Access	Permanently Stored
0x00	EEPROM Read	Read	x
0x01	EEPROM Write	Write	x

7.6.1 Command EEPROM Read

7.6.1.1 Notes

- Used to read data from EEPROM directly

7.6.1.2 Payload

Byte 1	Byte 2	Byte 3	Byte 4 - Byte 8
Page	Offset	Length	Reserved

7.6.1.3 Acknowledgment Payload

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5 - Byte 8
Page	Offset	Length	Reserved	Data

7.6.2 Command EEPROM Write

7.6.2.1 Notes

- Used to write data to EEPROM directly
- You are not allowed to change all values, if the EEPROM is locked (byte 0 is set to value 0xca)

7.6.2.2 Payload

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5 - Byte 8
Page	Offset	Length	Reserved	Data

7.6.2.3 Acknowledgment Payload

- Same structure as payload

7.7 Block ProductData and RFID

Number	Block Command	Access	Permanently Stored
0x00	Global Trade Identification Number (GTIN)	Read	x
0x01	Hardware Revision	Read	x

Number	Block Command	Access	Permanently Stored
0x02	Firmware Version	Read	x
0x03	Release Name	Read	x
0x04 - 0x07	Serial Number 1-4	Read	x
0x08 - 0x17	Name 1-16	Read	x
0x18 - 0x1F	OEM Free Use 0-7	Read	x
0x80	Tool RFID product information	Read	-

7.7.1 Command Global Trade Identification Number (GTIN)

7.7.1.1 Acknowledgment Payload

- 8 Byte GTIN
- Format: uint64_t (unsigned int)
- MSB: Byte 0
- LSB: Byte 7

7.7.2 Command Hardware Revision

7.7.2.1 Acknowledgment Payload

- 8 Bytes totally
  - 5 Bytes Reserved
  - 1 Byte Major Revision
  - 1 Byte Minor Revision
  - 1 Byte Build Revision

7.7.3 Command Firmware Version

7.7.3.1 Acknowledgment Payload

- 8 Bytes totally
  - 5 Bytes Reserved
  - 1 Byte Major Revision
  - 1 Byte Minor Revision
  - 1 Byte Build Revision

7.7.4 Command Release Name

7.7.4.1 Acknowledgment Payload

- 8 Byte ASCII Code
- NULL terminated or 8 Byte long

7.7.5 Command Serial Number

7.7.5.1 Notes

Command	Purpose
0x04	Serial Number 1
0x05	Serial Number 2
0x06	Serial Number 3
0x07	Serial Number 4

7.7.5.2 Acknowledgment Payload

- UTF-8 String (8 Byte)

7.7.6 Command Name

7.7.6.1 Notes

- Multiple Strings in different languages possible

Command	Purpose
0x08	Name 1
0x09	Name 2
0x0A	Name 3
0x0B	Name 4
0x0C	Name 5
0x0D	Name 6
0x0E	Name 7
0x0F	Name 8
0x10	Name 9
0x11	Name 10
0x12	Name 11
0x13	Name 12
0x14	Name 13
0x15	Name 14
0x16	Name 15
0x17	Name 16

7.7.6.2 Acknowledgment Payload

- UTF-8 String (8 Byte)

7.7.7 Command OEM Free Use

7.7.7.1 Notes

Command	Purpose
0x18	OEM Free Use 0
0x19	OEM Free Use 1
0x1A	OEM Free Use 2
0x1B	OEM Free Use 3
0x1C	OEM Free Use 4

Command	Purpose
0x1D	OEM Free Use 5
0x1E	OEM Free Use 6
0x1F	OEM Free Use 7

7.7.7.2 Acknowledgment Payload

- 8 Byte

7.7.8 Command Tool RFID product information

7.7.8.1 Acknowledgment Payload

- to be determined

7.8 Block Test

Number	Block Command	Access	Permanently Stored
0x00	Reserved	-	-
0x01	Test signal	-	-

7.8.1 Command Test signal

7.8.1.1 Payload

7.8.1.1.1 Byte 1:

Value	Meaning
0	Reserved
1	Line
2	Ramp

7.8.1.1.2 Byte 2: Module (Module specific)

7.8.1.1.3 Byte 3-8: Module specific

7.8.1.2 Acknowledgment Payload

7.8.1.2.1 Byte 1:

Value	Meaning
0	Reserved
1	Line
2	Ramp

7.8.1.2.2 Byte 2-3: Module (Module specific)

7.8.1.2.3 Byte 4-8: Module specific

7.9 Errors

Value	Description	Example
0	Specific Error	
1	Not available	
2	General Error	
3	Write not allowed	Setting of memory area in word not allowed
4	Unsupported format	64 Byte Data via CAN2.0 is not possible
5	Wrong key/magic number	
6	No SuperFrame inside SuperFrame	
7	EEPROM defect	