ICOtronic Documentation

MyTooliT

Contents

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 Overview ICOtronic System

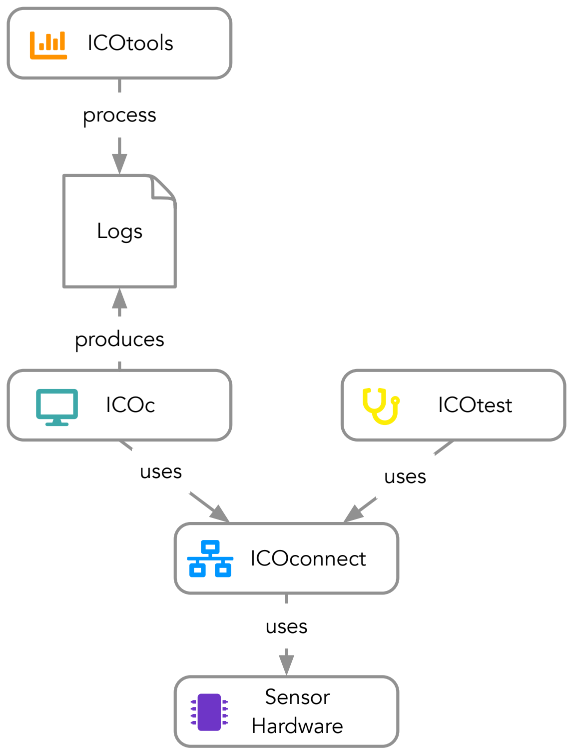
The text below describes how the (lower levels) of the ICOtronic system *should work*. Currently the system works similarly, but

- the communication interface (ICOconnect),
- the tests (ICOtest), and
- the user interface (ICOc)

are all part of a single monolithic code base.

- **ICOconnect**
  - Python package for CAN access to sensor hardware
  - Based on MyTooliT Communication protocol
  - Ideally available online (via PyPi) (requires opening up the code)
- **ICOc**
  - Uses ICOconnect to communicate with sensor hardware
  - User interface for sensor hardware
  - Configures STH/STU attributes (e.g. name, sampling frequency)
  - Records data (e.g. acceleration values) as log files

- **ICOtest**
  - Uses ICOconnect to communicate with sensor hardware
  - Test environment for sensor hardware (STH, STU)
  - Tests if the hardware works correctly
- **ICOtools**
  - Scripts that use data stored by ICOc to analyze captured data



7 Data Analysis

7.1 Tools

The list below describes some of the tools you can use to work with the `.hdf` files created by ICOc.

7.1.1 General

- HDFView: Tool to open and view HDF files. For more information on how to work with the files created by ICO in HDFView please take a look at the documentation of ICOc.
- `h5dump`: CLI tool to view HDF5 data; Part of HDF5 library download

7.1.2 Specific

- ICOlyzer: Tools to analyze measurement data from the ICOtronic system

7.2 Timestamps

The **embedded devices** of the ICOtronic system (STH, STU, SMH) only have knowledge of **relative time**, i.e. they can tell you about the time difference between two timepoints, but they do not know "which time it is". They simply do not have access to a time source for absolute time, such as a time server or an atomic clock.

In the case of ICOc the **measurement computer** adds the information about the **absolute time** a measurement takes place. This means that the timestamps in the measurement files are dependent on the time settings of the computer. Since the timing information is taken from the **timestamp of a CAN message** multiple data points have the same timestamp. The 8 data bytes of a CAN streaming data message will contain **three 16 bit measurement values**, which all **share the same timestamp**.

8 ADC

8.1 Guesstimates

- $\frac{3.9kS}{s} \cdot 256$ oversamples $= \frac{1MS}{s} \rightarrow$ 500kHz max. unambiguous signal detection (Nyquist)
- $\frac{15.6kS}{s} \cdot 64$ oversamples $= \frac{15.6kS}{s} \rightarrow$ 8kHz max. unambiguous signal detection (Nyquist)

8.2 Sources

- ADC stuff in BGM data sheets: search for SFDR - Spurious-free dynamic range (same as max. frequency, but good search query keyword)
- 1MS/s as ADC info (max. ADC)
- oversample rates taken from Walther's documentation, since result gets to 8kHz (what we use) this seems plausible

8.3 Further Reading

- somewhere some old ADC tests by NL (which combinations work, which don't)
- shared ADC pin with SPI and other stuff

8.4 Shared ADC SPI Pin

known since 2019 Spring, documented here 2021 Spring

- due to low pin count on BGM113 the PIN register is shared between ADC and SPI, as well as other functions
- on BGM12x other pins could be moved away, probably would alleviate ADC config issues
- Walther however dropped out 2020 before move to BGM12x

9 EEPROM

9.1 Terms

- Little Endian: Store the least significant byte (LSB) at the first (smallest) memory address and the most significant byte (MSB) at the last (highest) memory address

9.2 Layout

- Every page consists of 256 bytes
- The address of a page is the page number multiplied by 256

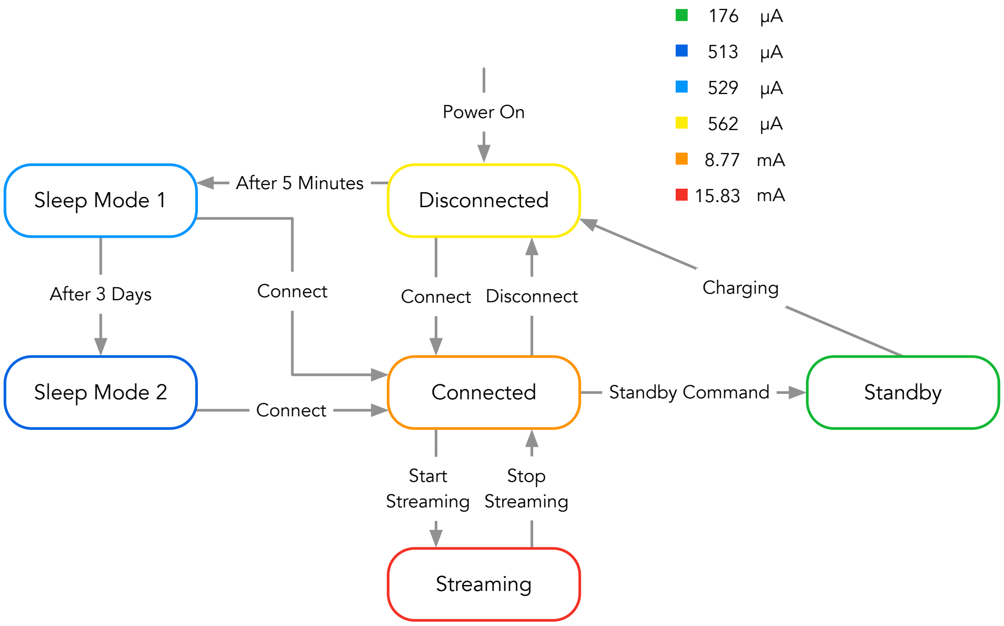
9.2.1 Pages:

Page Number	Name	Description
0	System Configuration	Store system specific data e.g. Bluetooth name and advertisement time
4	Product Data	Store product data e.g. serial number
5	Statistics	Store statistic data e.g. power on/off cycles
8	Calibration	Store configuration data like slope (`k`) and offset (`d`) values to derive SI value from ADC value

9.2.1.1 Page `System Configuration`

Byte	Length	Name	Comment	Format	Unit
0	1	Init	• `0xac`: Initialized • `0xca`: Locked • Other Value: Uninitialized)	-	-
1	8	Radio Name	Bluetooth advertisement name	ASCII	-
9	4	Sleep Time 1	Little Endian	Unsigned	ms
13	2	Advertisement Time 1	Little Endian	Unsigned	$0.625 \cdot$ ms
15	4	Sleep Time 2	Little Endian	Unsigned	ms
19	2	Advertisement Time 2	Little Endian	Unsigned	$0.625 \cdot$ ms

9.2.1.1.1 Sleep & Advertisement Times



Source for power consumption values (Firmware `2.1.10`): Bitrix24

- `Sleep Time 1`: Time to switch from the `Disconnected` state to `Sleep Mode 1` (low power usage)
- `Sleep Time 2`: Time switch from the `Sleep Mode 1` state to `Sleep Mode 2` (very low power usage)
- `Advertisement Time 1`: Advertisement time in `Sleep Mode 1`
- `Advertisement Time 2`: Advertisement time in `Sleep Mode 2`

9.2.1.2 Page `Product Data`

Byte	Length	Name	Comment	Format
0	8	Global Trade Identification Number (GTIN)	Little Endian	Unsigned
8	5	Hardware Version: Reserved		–
13	1	Hardware Version: Major		Unsigned
14	1	Hardware Version: Minor		Unsigned
15	1	Hardware Version: Patch		Unsigned
16	5	Firmware Version: Reserved		–
21	1	Firmware Version: Major		Unsigned
22	1	Firmware Version: Minor		Unsigned
23	1	Firmware Version: Patch		Unsigned
24	8	Release Name		UTF-8
32	32	Serial Number		UTF-8
64	128	Product Name		UTF-8
192	64	OEM Free Use		-

9.2.1.2.1 Version Numbers

- Version numbers will look like this `Major.Minor.Patch` (e.g. `1.2.3`)
- Major specifies the first digit of the version number (usually only increased for “breaking” changes)
- Minor specifies the second digit of the version number (usually only increased for “minor” changes)
- Patch specifies the third digit of the version number (usually increased for “bug fixes”)

9.2.1.2.2 Release Name

This text specifies the code name of the STH/STU software release

9.2.1.2.3 Serial Number

- Place for manufacture serial number (derived from ISBN)
- Possible Layout:
  - Product Group
  - Subgroup
  - Manufacture ID
  - Product Number
  - Check Digit
- Currently unused

9.2.1.2.4 Product Name

- This text might be used to extend the serial Number
- Possible Use: URL that point to additional information
- Currently unused

9.2.1.2.5 OEM Free Use

- Manufacture specific information
- Format is free to choose

9.2.1.3 Page `Statistics`

Byte	Length	Name	Comment	Format	Unit
0	4	Power On Cycles	Little Endian	Unsigned	-
4	4	Power Off Cycles	Little Endian	Unsigned	-
8	4	Operating Time	Little Endian	Unsigned	s
12	4	Under Voltage Counter	Little Endian	Unsigned	-
16	4	Watchdog Reset Counter	Little Endian	Unsigned	-
20	4	Production Date: Year		ASCII	-
24	2	Production Date: Month		ASCII	-
26	2	Production Date: Day		ASCII	-
28	4	Batch Number	Consecutive number for manufactured devices	ASCII	-

9.2.1.3.1 Power On Cycles

Please note, that a reset also counts as power on cycle

9.2.1.3.2 Under Voltage Counter

Counts of under voltages that cause turn off state (brownout)

Byte	Length	Name	Comment	Format

9.2.1.4 Page `Calibration`

Byte	Length	Name	Comment	Format
0	4	Acceleration X: Slope	Little Endian	Float
4	4	Acceleration X: Offset	Little Endian	Float
8	4	Acceleration Y: Slope	Little Endian	Float
12	4	Acceleration Y: Offset	Little Endian	Float
16	4	Acceleration Z: Slope	Little Endian	Float
20	4	Acceleration Z: Offset	Little Endian	Float
24	4	Voltage Battery: Slope	Little Endian	Float
28	4	Voltage Battery: Offset	Little Endian	Float
32	4	Voltage 2: Slope	Little Endian	Float
36	4	Voltage 2: Offset	Little Endian	Float
40	4	Voltage 3: Slope	Little Endian	Float
44	4	Voltage 3: Offset	Little Endian	Float
48	4	Internal Temperature: Slope	Little Endian	Float
52	4	Internal Temperature: Offset	Little Endian	Float
56	4	Temperature 2: Slope	Little Endian	Float
60	4	Temperature 2: Offset	Little Endian	Float
64	4	Temperature 3: Slope	Little Endian	Float
68	4	Temperature 3: Offset	Little Endian	Float

9.2.1.4.1 Slope & Offset The values slope (`k`) and offset (`d`) specify the values in the equation for the linear function:

$$y = f(x) = k \cdot x + d$$

9.3 STH EEPROM

This file contains the default values of the STH EEPROM. For a more detailed description of the values, please take a look at the description of the EEPROM layout.

9.3.1 Used Pages

Page Number	Page Name
`0x0`	System Configuration
`0x4`	Product Data
`0x5`	Statistics
`0x8`	Calibration

9.3.1.1 Page `System Configuration`

Name	Address	Length	Read Only	Value	Comment	Unit	Format
EEPROM Status	0	1	True	`0xac`	Value for initialized EEPROM	-	
STH Name	1	8	False	Base64 encoded Bluetooth MAC address or firmware name	e.g. `CGvXAd6B`, `Tanja`	-	UTF-8
Sleep Time 1	9	4	False	300000	5 minutes	ms	Unsigned
Advertisement Time 1	13	2	False	2000	1.25 seconds	0.625 · ms	Unsigned
Sleep Time 2	15	4	False	259200000	3 days	ms	Unsigned
Advertisement Time 2	19	2	False	4000	2.5 seconds	0.625 · ms	Unsigned

9.3.1.1.1 Initialization All of the values of the system configuration are set to default values on reset of the STH, if the EEPROM status (byte) is **not** set to `Initialized` (`0xac`) or `Locked` (`0xca`). The default values are described above. The name is set to the firmware version name (`Tanja`) and does not use the Base64 encoded Bluetooth MAC address.

9.3.1.2 Page `Product Data`

Name	Address	Length	Read Only	Value	Format
GTIN	0	8	True	0	Unsigned
Hardware Version: Major	13	1	False	-	Unsigned
Hardware Version: Minor	14	1	False	-	Unsigned
Hardware Version: Patch	15	1	False	-	Unsigned
Firmware Version: Major	21	1	False	-	Unsigned
Firmware Version: Minor	22	1	False	-	Unsigned
Firmware Version: Patch	23	1	False	-	Unsigned
Release Name	24	8	False	Tanja	UTF-8
Serial Number	32	32	True	0	UTF-8
Product Name	64	128	True	0	UTF-8
OEM Free Use	192	64	True	0	-

9.3.1.2.1 Version Numbers

- **Hardware Version**: This number depends on the hardware version (printed on the PCB). The value itself can be changed in the main configuration file of ICOc
- **Firmware Version**: This number depends on the current STH software version
- **Release Name**: This text can be changed in the configuration of ICOc
- **Serial Number**: This text can be changed in the configuration of ICOc
- **Product Name**: This text can be changed in the configuration of ICOc
- **OEM Free Use**: This value can be changed in the configuration of ICOc.

9.3.1.3 Page `Statistics`

Name	Address	Length	Read Only	Value	Unit	Format
Power On Cycles	0	4	True	0	-	Unsigned
Power Off Cycles	4	4	True	0	-	Unsigned
Operating Time	8	4	True	0	s	Unsigned
Under Voltage Counter	12	4	True	0	-	Unsigned
Watchdog Reset Counter	16	4	True	0	-	Unsigned
Production Date: Year	20	4	False	-	-	ASCII
Production Date: Month	24	2	False	-	-	ASCII
Production Date: Day	26	2	False	-	-	ASCII
Batch Number	28	4	True	-	-	Unsigned

- **Production Date**: This date depends on the production date of the STH (printed on the PCB). It can be changed in the configuration of ICOc.
- **Batch Number**: This value can be changed in the configuration of ICOc.

9.3.1.4 Page `Calibration`

Name	Address	Length	Read Only	Value	Format
Acceleration X: Slope	0	4	False	-	Float
Acceleration X: Offset	4	4	False	-	Float
Acceleration Y: Slope	8	4	False	-	Float
Acceleration Y: Offset	12	4	False	-	Float
Acceleration Z: Slope	16	4	False	-	Float
Acceleration Z: Offset	20	4	False	-	Float

9.3.1.4.1 Acceleration

- **Acceleration: Slope**: Acceleration increase for a single step in a certain direction (x, y, z) according to the following formula:

  $$\frac{a_{max}}{ADC_{max}}$$

  Here

  - a_{max} is the maximum acceleration difference (e.g. `200` for a ±100 g sensor)
  - ADC_{max} is the maximum value of the ADC (e.g. `65553` (= `2`^{1}) for a 16-bit analog-digital converter)

- **Acceleration: Offset**: The negative offset of the acceleration value in a certain direction (x, y, z) according to the following formula:

  $$-\frac{a_{max}}{2}$$

  Here a_{max} is the maximum acceleration difference (e.g. `100` for a ±50 g sensor)

Note: Since the maximum acceleration difference (a_{max}) should be the same for each axis (for all acceleration sensor we use), the slope and offset values should be the same for each axis as well.

9.4 STU EEPROM

This file contains the default values for the STU EEPROM. For a more detailed description of the values, please take a look at the description of the EEPROM layout.

9.4.1 Used Pages

Page Number	Page Name
`0x0`	System Configuration
`0x4`	Product Data
`0x5`	Statistics

9.4.1.1 Page `System Configuration`

Name	Address	Length	Read Only	Value	Comment	Unit	Format
EEPROM Status	0	1	True	`0xac`	Value for initialized EEPROM	-	
STU Name	1	8	False	Firmware version name	e.g. `Valerie`	-	UTF-8

9.4.1.1.1 Initialization All of the values of the system configuration are set to the default values above on reset of the STU, if the EEPROM status (byte) is **not** set to `Initialized` (`0xac`) or `Locked` (`0xca`). The values for the sleep times and advertisement times are set (to the same values the STH uses) on initialization too. However, since the STU is not battery-powered these timing values are probably not relevant.

9.4.1.2 Page `Product Data`

Name	Address	Length	Read Only	Value	Format
GTIN	0	8	True	0	Unsigned
Hardware Version: Major	13	1	True	-	Unsigned
Hardware Version: Minor	14	1	True	-	Unsigned
Hardware Version: Patch	15	1	True	-	Unsigned
Firmware Version: Major	21	1	True	-	Unsigned
Firmware Version: Minor	22	1	True	-	Unsigned
Firmware Version: Patch	23	1	True	-	Unsigned
Release Name	24	8	True	Valerie	UTF-8
Serial Number	32	32	True	0	UTF-8
Product Name	64	128	True	0	UTF-8
OEM Free Use	192	64	True	0	-

9.4.1.2.1 Version Numbers

- **Hardware Version**: This number depends on the hardware version (printed on the PCB). The value itself can be changed in the main configuration file of ICOc
- **Firmware Version**: This number depends on the current STU software version

- **Release Name**: This text can be changed in the configuration of ICOc
- **Serial Number**: This text can be changed in the configuration of ICOc
- **Product Name**: This text can be changed in the configuration of ICOc
- **OEM Free Use**: This value can be changed in the configuration of ICOc.

9.4.1.3 Page `Statistics`

Name	Address	Length	Read Only	Value	Unit	Format
Power On Cycles	0	4	True	0	-	Unsigned
Power Off Cycles	4	4	True	0	-	Unsigned
Operating Time	8	4	True	0	s	Unsigned
Under Voltage Counter	12	4	True	0	-	Unsigned
Watchdog Reset Counter	16	4	True	0	-	Unsigned
Production Date: Year	20	4	True	-	-	ASCII
Production Date: Month	24	2	True	-	-	ASCII
Production Date: Day	26	2	True	-	-	ASCII
Batch Number	28	4	True	-	-	Unsigned

- **Production Date**: This date depends on the production date of the STU. It can be changed in the configuration of ICOc.
- **Batch Number**: This value can be changed in the configuration of ICOc.

10 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.

10.1 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.

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:

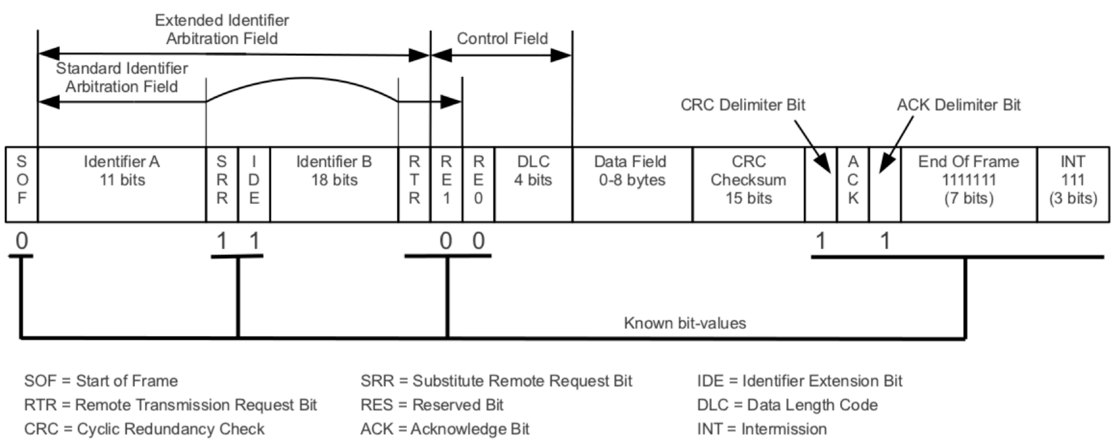


Figure 1: CAN Frame

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.

10.1.1 Reserved Bits

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

10.2 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.

10.2.1 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

10.2.2 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).
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.

10.2.3 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.

10.2.4 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.

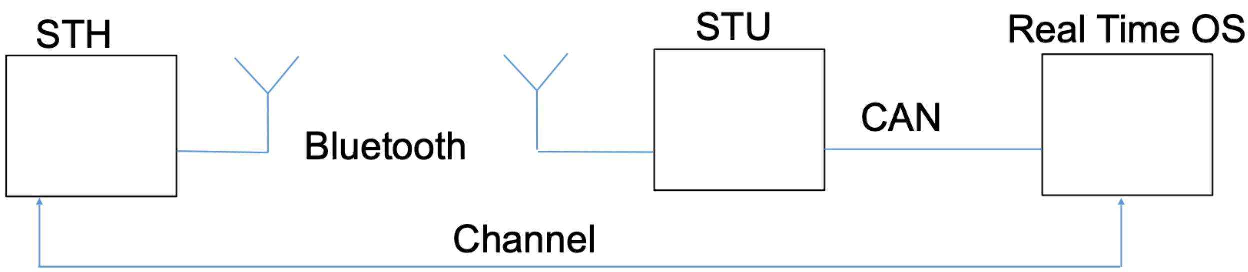


Figure 2: End To End Communication

10.2.5 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.

10.2.6 Payload

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

10.2.7 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.

10.2.8 Transmission Speed

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

10.2.9 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.

10.2.10 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.

11 Commands

11.1 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

11.2 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	–

11.2.1 Command `Verboten`

This command is mainly used for initialization purposes

11.2.2 Command `Reset`

Reset the specified receiver. This command has no payload.

11.2.3 Command `Get/Set State`

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

11.2.3.1 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)

11.2.3.2 Payload

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

11.2.3.3 Acknowledgment Payload

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

11.2.3.4 Error Payload

Byte 2
`Error Reason`

11.2.4 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 status 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

Value	Meaning
`1`	Error

- `Network State`:

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

11.2.4.1 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

11.2.4.1.1 STH

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

11.2.4.1.2 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`

11.2.4.1.3 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

11.2.4.2 Acknowledgement Payload

- Same structure as payload

11.2.4.3 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

11.2.5 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:

```c
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

11.2.5.1 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

11.2.5.1.1 STH

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

11.2.5.1.2 STU

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

11.2.5.1.3 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

11.2.5.2 Acknowledgement Payload

- Same structure as payload

11.2.5.3 Error Payload

- Same structure as error payload for node status command

11.2.6 Command `Bluetooth`

- In general you need at least the following commands to connect to an STH

  1. `Activate`: Activate Bluetooth on the STU
  2. `Get number of available devices`: Check which STHs are available at the STU
  3. `Connect to device (with Bluetooth MAC address)` or `Connect to device (with device number)`: Connect to the STH at the specified STU

  Connecting to the STH will not work, if you do not **check for available devices first**

- `Bluetooth Subcommand`

Value	Meaning
`0`	Reserved

Value	Meaning
`1`	Activate
`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`	Deactivate
`10`	Get send counter
`11`	Received RX frames
`12`	Get RSSI (Received Signal Strength Indication)
`13`	Read energy mode reduced
`14`	Write energy mode reduced
`15`	Read energy mode lowest
`16`	Write energy mode lowest
`17`	Get Bluetooth MAC address
`18`	Connect to device (with Bluetooth MAC address)

- The Bluetooth `Activate` command (on the STU) is required
  - to **enable the advertisement** (and hence the OTA update functionality) of the STU and
  - before you search for sensor devices.

  This dual functionality is (probably) also the reason why a second STU might show up in the list of available (sensor) devices.

- `Device Number`: Sequential positive number assigned by STU to available STH nodes
  - For a single STH this number will be `0`
  - The number `255` (`0xff`) is reserved for "self addressing" (used for example when we ask a connected STH for its own MAC address). **Note**: A connected STH also returns its own name, if you use the read name subcommands (`5` and `6`) and a device number other than `0xff`.

- `Bluetooth Value`

`Bluetooth Subcommand`	Value
`0`	–
`1`	–
`2`	–
`3`	ASCII string
`4`	ASCII string (`NULL`)
`5`	–
`6`	–
`7`	–
`8`	–
`9`	–
`10`	–
`11`	–
`12`	–
`13`	–

`Bluetooth Subcommand`	Value
14	Byte 3 – 6: Time from normal to reduced energy mode in ms (Little Endian) Byte 7 – 8: Advertisement time for reduced energy mode in 0.625 · ms (Little Endian)
15	–
16	Byte 3 – 6: Time from reduced to lowest energy mode in ms Byte 7 – 8: Advertisement time for lowest energy mode in 0.625 · ms Little endian 0 = read
17	–
18	Bytes of Bluetooth MAC address in reversed order (from right to left)

- `Bluetooth Return Value`

`Bluetooth Subcommand`	Value
0	`NULL`
1	6 Bytes containing `NULL` (`0`)
2	ASCII string containing the number of available devices
3	ASCII string
4	ASCII string
5	ASCII string containing the first 6 characters of the Bluetooth advertisement name
6	• ASCII string containing the last 2 characters of the Bluetooth advertisement name • `NULL` if not connected
7	First byte is: • **`true`** (`1`) if in search mode, at least single device was found, no legacy mode and scanning mode active • **`false`** (`0`) otherwise
8	First byte is: • **`true`** (`1`) if connected • **`false`** (`0`) otherwise Followed by 5 bytes containing `NULL` (`0`)
9	6 Bytes containing `NULL` (`0`)
10	6 Byte **`unsigned int`** (Big Endian)
11	6 Byte **`unsigned int`**
12	• First byte contains RSSI as signed number • All other bytes are `NULL` (`0`)
13	Byte 3 – 6: Time form normal to reduced energy mode in ms Byte 7 – 8: Advertisement time for reduced energy mode in 0.625 · ms Big Endian
14	Byte 3 – 6: Time form normal to reduced energy mode in ms (Little Endian) Byte 7 – 8: Advertisement time for reduced energy mode in 0.625 · ms (Little Endian)
15	Byte 3 – 6: Time form reduced to lowest energy mode in ms Byte 7 – 8: Advertisement time for lowest energy mode in 0.625 · 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 0.625 · ms Little Endian
17	Bytes of Bluetooth MAC address in reversed order (from right to left)
18	Bytes of Bluetooth MAC address in reversed order (from right to left)

11.2.6.1 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 `Activate` command)

11.2.6.2 Acknowledgement Payload

Byte 1
Same as Payload

Byte 2
Same as Payload

Byte 3 – 8
`Bluetooth Return Value`

11.3 Block `Streaming`

Number	Block Command	Access	Permanently Stored
`0x00`	Data	Event	–
`0x20`	Voltage	Event	–

11.3.1 Values

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

Value	Data Amount
0	Stop (stream)
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

The streaming data itself can have the following structure:

  - value 1
  - value 2
  - value 3
  - value 1 / value 2 / value 3
  - value 1 / value 2
  - value 1 / value 3
  - value 2 / value 3

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

- `Request`:

Value	Meaning
`0`	Single Request
`1`	Stream

- `Bytes`:

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

11.3.2 Command `Data`

- This command is usually used to access **acceleration streaming data** (for certain axes)
- On **newer firmware**/hardware the streaming command might return **other data as well** (temperature, force, …)
- We refer to measurement channel 1 (2 and 3) in the text below, while we previously used the term x-axis (y-axis and z-axis)
- Requesting while streaming is possible
- Only single stream allowed
- Requesting stream in different format stops last stream
- Tuple format (depending on active channel, see payload):
  - 1/2/3
  - 1/2
  - 1/3
  - 2/3

11.3.2.1 Payload

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

11.3.2.2 Acknowledgment Payload

Byte 1
Same as Payload

Byte 2
Sequence Counter

11.3.2.2.1 Streaming Data Bytes

- Data is sent in little endian order (at least for 2 byte format)
- Older streaming data is stored in first bytes, newer data in later bytes
- Values are stored in first available bytes,
  - first measurement channel 1 (`x`) (if requested),
  - then measurement channel 2 (`y`) (if requested),
  - then measurement channel 3 (`z`) (if requested)
- Data length depends on requested values and number of sets

Examples

- Request first measurement channel
- Single data set
- 2 Byte format

Byte 3
Value Channel 1 (LSB)

Byte 4
Value Channel 1 (MSB)

- Request second and third measurement channel
- Single data set
- 2 Byte format

Byte 5
Value Channel 2 (LSB)

Byte 6
Value Channel 2 (MSB)

Byte 7
Value Channel 3 (LSB)

Byte 8
Value Channel 3 (MSB)

11.3.3 Command `Voltage`

11.3.3.1 Notes

- Highest voltage sampling rate determines bit stream rate
- Requesting while streaming is possible
- To determine the supply/battery voltage (voltage 1) value you need to multiply the returned values with the number `5.7`. This is the result of the voltage divider circuit we use (which contains a 470 kΩ and 100 kΩ resistor).

11.3.3.2 Payload

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

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

11.4 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

11.4.1 Command `Power On Cycles, Power Off Cycles`

11.4.1.1 Notes

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

11.4.1.2 ACK Payload

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

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

11.4.2 Command `Operating time`

11.4.2.1 Notes

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

11.4.2.2 ACK Payload

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

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

11.4.3 Command `Under Voltage Counter`

11.4.3.1 ACK Payload

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

11.4.4 Command `Watchdog Reset Counter`

11.4.4.1 ACK Payload

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

11.4.5 Command `Production Date`

11.4.5.1 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

11.5 Block `Configuration`

Number	Block Command	Access	Permanently Stored
`0x00`	Get/Set ADC Configuration	Read/Write	x
`0x01`	Get/Set Sensors	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

11.5.1 Command `Get/Set ADC Configuration`

11.5.1.1 Values

- `Get/Set Config`:

Value	Meaning
`0`	Get Config
`1`	Set Config

- `Prescaler`:

  Possible Values: 1 – 127

- `Acquisition Time`:
  - Sample and hold time i.e. time to charge capacitor that is cut off and measured at digital quantisation in cycles
  - $value + 1$ iff $value \leq 3$ (e.g. 4 cycles for a value of 3)
  - $2^{value-1}$ iff $value > 3$ (e.g. 8 cycles for a value of 4)
  - Possible acquisition times: 1, 2, 3, 4, 8, 16, 32, … , 256

- `Oversampling Rate`:
  - Oversampling Rate: 2^{value}
  - No oversampling if $value = 0$
  - Possible oversampling rates: 1, 2, 4, 8, 16, … , 4096

- `Reference Voltage`:
  - $value = V_{reference} \cdot 20$ (e.g. 25 for 1.25 V)
  - Possible Voltages:
    * 1V25
    * 1V65
    * 1V8
    * 2V1
    * 2V2
    * 2V5
    * 2V7
    * 3V3 (VDD)
    * 5V
    * 6V6

11.5.1.2 Payload

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

Byte 2
`Prescaler`

Byte 3
`Acquisition Time`

Byte 4
`Oversampling Rate`

Byte 5
`Reference Voltage`

Byte 6 - Byte 8
Reserved

11.5.1.3 Acknowledgment Payload

- Same structure as payload

11.5.1.4 Notes

11.5.1.4.1 Sampling Rate

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

$$f_{clock} = 38400000 Hz$$

11.5.1.4.2 Setting at Reset

- `Prescaler`: 2
- `Acquisition Time`: 8 (byte value = 4)
- `Oversampling Rate`: 64 (byte value = 6)
- `Reference Voltage`: 3.3 (V_{DD}) (byte value = 66)

11.5.1.4.3 Recommended Values The table below list some ADC configuration values we recommend, if you want to change the sample rate from the **default of 9524 Hz**.

Sample Rate [Hz]	Prescaler	Acquisition Time	Oversampling Rate
9524	**2**	**8**	**64**
9375	3	3	64
8889	2	32	32
6897	2	16	64
4762	2	8	128
3448	2	16	128
2381	2	8	256
1724	2	16	256
1190	2	8	512
862	2	16	512
595	2	8	1024
431	2	16	1024
298	2	8	2048
216	2	16	2048
149	2	8	4096
108	2	16	4096

11.5.2 Command `Get/Set Sensors`

11.5.2.1 Notes

- If a sensor number sent with a "Set" command is greater than the number of sensors defined by the sensor device then the sensor device will react with an error message.
- The sensor number `0` represents a special value that tells the sensor device to keep the current sensor configuration value for the specified measurement channel.

11.5.2.2 Values

- `Get/Set State`:

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

11.5.2.3 Payload

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

Byte 2
Sensor (number) for first measurement channel

Byte 3
Sensor (number) for second measurement channel

Byte 4
Sensor (number) for third measurement channel

Byte 5 - Byte 8
Reserved

11.5.2.4 Acknowledgment Payload

- Same structure as payload

11.5.3 Command `Get/Set Calibration Factor k`

11.5.3.1 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
3	z-Axis / Third measure point

- `Get/Set Value`:

Value	Meaning
0	Get Value
1	Set Value

11.5.3.2 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)

11.5.3.3 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)

11.5.4 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`.

11.5.5 Command `Calibration Measurement`

11.5.5.1 Notes

- Activating the self test of an accelerometer:
  - `Calibration Get/Set`: Get
  - `Calibration Measurement Element`: Acceleration
  - `Calibration Method`: Activate

  seems to only work for the next measurement. This means the STH will return the “normal” acceleration value after you read the acceleration a second time, even if you do not deactivate the self test before the second measurement.

11.5.5.2 Values

- `Calibration Get/Set`:

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

- `Calibration Method`:

Value	Meaning
0	Reserved
1	Activate
2	Deactivate
3	Measure

- `Calibration Measurement Element`:

Value	Meaning
0	Acceleration
1	Temperature (for $V_{REF} = 1.25\ V$ the temperature is returned in 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)

- `Reference Voltage`: This value specifies the reference voltage in fractures of $\frac{1}{20}$ of a volt. A common value would be 66 ($\frac{66}{20} = \frac{33}{10} = 3.3$) for the supply voltage (V_{DD} Voltage Drain Drain) of 3.3 V.

11.5.5.3 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
`Reference Voltage`

Byte 5 - Byte 8
Reserved

11.5.5.4 Acknowledgment Payload

Byte 1 - Byte 4
Same as Payload

Byte 5 - Byte 8
Result

11.5.6 Command `HMI Configuration`

11.5.6.1 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

11.5.6.2 Payload

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

Byte 2
Number (0-255)

Byte 3
`ON/OFF`

Byte 4
Reserved

Byte 5 - Byte 8
Reserved

11.5.6.3 Acknowledgment Payload

- Same structure as payload

11.6 Block `EEPROM`

Number	Block Command	Access	Permanently Stored
`0x00`	EEPROM Read	Read	x
`0x01`	EEPROM Write	Write	x
`0x20`	Read Write Request Counter	Read	x

11.6.1 Command `EEPROM Read`

11.6.1.1 Notes

- Used to read data from EEPROM directly

11.6.1.2 Payload

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

11.6.1.3 Acknowledgment Payload

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

11.6.2 Command `EEPROM Write`

11.6.2.1 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`)

11.6.2.2 Payload

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

11.6.2.3 Acknowledgment Payload

- Same structure as payload

11.6.3 Command `Read Write Request Counter`

11.6.3.1 Notes

- The current documentation of this command is based on the (old) code of ICOc

11.6.3.2 Payload

Byte 1 - Byte 8
0

11.6.3.3 Acknowledgment Payload

Byte 1 - Byte 4	Byte 5 - Byte 8
Undefined	EEPROM Write Requests

11.7 Block `Product Data and RFID`

Number	Block Command	Access	Permanently Stored
`0x00`	Global Trade Identification Number (GTIN)	Read	x
`0x01`	Hardware Version	Read	x
`0x02`	Firmware Version	Read	x
`0x03`	Release Name	Read	x
`0x04` - `0x07`	Serial Number 1-4	Read	x
`0x08` - `0x17`	Product Name 1-16	Read	x
`0x18` - `0x1F`	OEM Free Use 0-7	Read	x
`0x80`	Tool RFID product information	Read	-

11.7.1 Command `Global Trade Identification Number (GTIN)`

11.7.1.1 Acknowledgment Payload

Byte 1 (MSB) – Byte 8 (LSB)
GTIN (`unsigned int`)

11.7.2 Command `Hardware Version`

11.7.2.1 Acknowledgment Payload

Byte 1 – Byte 5	Byte 6	Byte 7	Byte 8
Reserved	Major Version	Minor Version	Patch Version

11.7.3 Command `Firmware Version`

11.7.3.1 Acknowledgment Payload

Byte 1 – Byte 5	Byte 6	Byte 7	Byte 8
Reserved	Major Version	Minor Version	Patch Version

11.7.4 Command `Release Name`

11.7.4.1 Acknowledgment Payload

Byte 1 – Byte 8
Firmware Release Name (ASCII)

- `NULL` terminated or 8 bytes long

11.7.5 Command `Serial Number`

11.7.5.1 Notes

Number	Purpose
`0x04`	Get first part of serial number
`0x05`	Get second part of serial number
`0x06`	Get third part of serial number
`0x07`	Get last part of serial number

11.7.5.2 Acknowledgment Payload

- UTF-8 string (8 bytes for each part)
- The whole serial number is a concatenation of its parts starting with the first part of the serial number

11.7.6 Command `Product Name`

11.7.6.1 Notes

- Multiple Strings in different languages possible

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

11.7.6.2 Acknowledgment Payload

- UTF-8 string (8 bytes)

11.7.7 Command `OEM Free Use`

11.7.7.1 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
`0x1D`	OEM Free Use 5
`0x1E`	OEM Free Use 6
`0x1F`	OEM Free Use 7

11.7.7.2 Acknowledgment Payload

- 8 bytes for each block command

11.7.8 Command `Tool RFID product information`

11.7.8.1 Acknowledgment Payload

- to be determined

11.8 Block `Test`

Number	Block Command	Access	Permanently Stored
`0x00`	Reserved	-	-
`0x01`	Test signal	-	-
`0x69`	Test Pfeifferl	-	-

11.8.1 Command `Test signal`

11.8.1.1 Payload

11.8.1.1.1 Byte 1:

Value	Meaning
0	Reserved
1	Line
2	Ramp

11.8.1.1.2 Byte 2: Module (Module specific)

11.8.1.1.3 Byte 3-8: Module specific

11.8.1.2 Acknowledgment Payload

11.8.1.2.1 Byte 1:

Value	Meaning
0	Reserved
1	Line
2	Ramp

11.8.1.2.2 Byte 2-3: Module (Module specific)

11.8.1.2.3 Byte 4-8: Module specific

11.9 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

Value	Description	Example
5	Wrong key/magic number	
6	No SuperFrame inside SuperFrame	
7	EEPROM defect	

11.9.1 Command `Test Pfeifferl`

11.9.1.1 Payload

11.9.1.1.1 Byte 1: Ch:

Silabs Channel number, 0-39

11.9.1.1.2 Byte 2: Type:

- 254: CW
- 253: pn9

11.9.1.1.3 Byte 3: Wait_Before:

Time till the RF is turned on in seconds

11.9.1.1.4 Byte 4-5: On_Duration:

Duration the RF is turned on in seconds

11.9.1.1.5 Byte 6: Pwr:

RF Power (32 = max)

11.9.1.1.6 Byte 7: Phy:

Physical antenna chosen (1 = default)

11.9.1.1.7 Byte 8: Len:

Length of an RF packet (37 = default)

11.9.1.2 Acknowledgment Payload

- Same structure as payload
- ACK is sent before starting the process