CANedge1 Docs, 01.08.05

Release 01.08.05

CSS Electronics

Jan 08, 2025

CONTENTS

0.1 About this manual

0.1.1 Purpose

This manual describes the functionality of the CANedge1 (firmware 01.08.05) with focus on:

1. Hardware & installation
2. Configuration
3. Firmware upgrade

This manual does not provide details on available software/API tools.

Note

Most of the information contained in this manual is found in the *configuration* sections.

0.1.2 Notation used

The following notation is used throughout this documentation:

0.1.2.1 Admonitions

Note

Used to highlight supplementary information

Warning

Used if incorrect use may result in unexpected behaviour

Danger

Used if incorrect use may result in damage to the device or personal injury

0.1.2.2 Number bases

When relevant, the base of a number is written explicitly as x_y, with y as the base.

The following number bases are used throughout this documentation:

- Binary ($y = 2$). Example: The binary number 10101010 is written as 10101010_2
- Decimal ($y = 10$). Example: The decimal number 170 is written as 170_{10}
- Hexadecimal ($y = 16$). Example: The hexadecimal number AA is written as AA_{16}

The value of a number is the same regardless of the base (e.g. the values in above examples are equal $10101010_2 = 170_{10} = AA_{16}$). However, it is sometimes more convenient to represent the number using a specific base.

0.2 Specification

0.2.1 Logging

- Storage
  - Extractable industry grade micro SD-card (8-32GB)
  - Standard FAT file system (can be read directly by a PC)
  - Logging to industry standard `.MF4` (ASAM MDF4) file format
- Organization
  - Log files grouped by session (power cycle)
  - Log files split based on file configurable size or time
  - Optional cyclic-logging mode (oldest log file is deleted when memory is full)
- Performance
  - Simultaneous logging from 2 x CAN-bus + 2 x LIN-bus
  - Message time stamping with 50 us resolution
  - High message rate[1]
  - Optional data compression (LZSS)
- Security
  - Globally unique device ID with customizable device name
  - Power safe (device can be disconnected during operation without risk of data corruption)
  - Optional end-2-end data encryption (AES128-GCM)

0.2.2 Real-time clock (RTC)

- High precision real-time clock retains date and time when device is off
- The real-time clock can be automatically synced from various sources[2]

0.2.3 CAN-bus (x2)

- Physical
  - Two physical CAN-bus interfaces
  - Industry standard DB9 (D-sub9) connectors
- Transceiver
  - Compliant with CAN Protocol Version 2.0 Part A, B and ISO 11898-1
  - Compliant with ISO CAN FD and Bosch CAN FD
  - Ideal passive behavior when unpowered (high impedance / no load)
  - Short circuit protection
  - Transient protection
  - TXD dominant timeout (prevents network blocking in the event of a failure)
  - Data rates up to 5Mbps[3]
- Controller

[1] See the performance tests
[2] Synchronization sources depend on device variant. See configuration section for more information
[3] Supported FD bit-rates: 1M, 2M, 4M

  - Based on MCAN IP from Bosch
  - Bit-rate: Auto-detect (from list[4]), manual simple (from list[5]) or advanced (bit-timing)
  - 128 standard CAN ID + 64 extended CAN ID filters (per interface)
  - Advanced filter configuration: Range, mask, acceptance, rejection
  - Configurable transmit messages, single shot or periodic (up to 128/64 regular/extended)
  - Message down-sampling based on:
    * Count
    * Time
    * Change in data
  - Support for Remote-Transmission-Request (RTR) frames
  - Silent modes: Restricted (acknowledge only) or monitoring (transmission disabled)
  - Supports all CAN based protocols (J1939, CANopen, OBD2, NMEA 2000, ...)[6]
  - Application
    * Cross-channel *control-message* for start/stop of reception/transmission
    * Heartbeat-message to broadcast device time, space left on SD-card and reception/transmission state

0.2.4 LIN-bus (x2)

- Physical
  - Two physical LIN-bus interfaces
  - Industry standard DB9 (D-sub9) connectors
  - No internal diode and resistor for publishing mode
- Transceiver
  - Protection: ±8kV HBM ESD, ±1.5kV CDM, ±58V bus fault
  - Supports 4V to 24V applications
  - TXD dominant timeout (prevents network blocking in the event of a failure)
  - Data rates up to 20kbps
- Controller
  - Support for both publisher and subscriber modes
  - Automatic[7] and custom frame lengths
  - Classic and Extended checksum formats
  - Configurable transmit messages, single shot or periodic

0.2.5 Electrical

- Device supply
  - Channel 1 (`CH1`) voltage supply range: +7.0 V to +32 V DC[8]
  - Reverse voltage protection[9]

[4] Bit-rate list: 5k, 10k, 20k, 33.333k, 47.619k, 50k, 83.333k, 95.238k, 100k, 125k, 250k, 500k, 800k, 1M
[5] Bit-rate list: 5k, 10k, 20k, 33.333k, 47.619k, 50k, 83.333k, 95.238k, 100k, 125k, 250k, 500k, 800k, 1M, 2M, 4M
[6] The device logs raw data frames
[7] Data lengths are defined by bits 4 and 5 of the LIN identifier
[8] The device is supplied trough connector 1 (`CH1`)
[9] Up to 24V

  - Transient voltage event protection on supply lines[10]
  - Consumption: 0.8 W[11]
- Secondary port output supply[12]
  - Channel 2 (`CH2`) fixed 5 V output supply (up to 1 A)[13]
  - Supports power out scheduling to control the output state based on time of day

0.2.6 Mechanical

- Status indicated using external LEDs
- Robust and compact aluminum enclosure
- Operating temperature: -25 °C to +70 °C
- Hardware version 00.03:
  - Dimensions: 44.2 x 75.0 x 20.0 mm (L x W x H)[14]
  - Weight: ~ 70 g[15]
- Hardware version $\leq$ 00.02:
  - Dimensions: 50.2 x 75.4 x 24.5 mm (L x W x H)[15]
  - Weight: ~ 100 g[15]

[10] The transient voltage protection is designed to clamp low energy voltage events. High energy voltage events may overheat and destroy the input protection
[11] Peak consumption during logging and active network connectivity (if supported)
[12] Can be used to supply external devices
[13] The 5V output can be used to power WiFi hotspots, sensors, small actuators, external LEDs, etc.
[14] Excluding any external antennas and flanges
[15] Excluding any external antennas

0.3 Hardware

This section contains information regarding the CANedge hardware, including installation requirements, connector pinouts, enclosure, SD card, LEDs and label.

0.3.1 Installation

This section outlines the installation requirements that shall be satisfied.

Table of Contents

0.3.1.1 Supply quality

The nominal voltage shall be kept within specifications at all times. The device is internally protected against low energy voltage events which can be expected as a result of supply wire noise, ESD and stub-wire inductance.

If the supply line is shared with inductive loads, care should be taken to ensure high energy voltage events do not reach the device. Automotive environments often include several sources of electrical hazards, such as load dumps (disconnection of battery while charging), relay contacts, solenoids, alternator, fuel injectors etc. The internal protection circuitry of the device is not capable of handling high energy voltage events directly from such sources.

0.3.1.2 Grounding

ISO 11898-2 tolerates some level of ground offset between nodes. To ensure the offset remains within range, it is recommended to use a single point ground reference for all nodes connected to the CAN-bus. This may require the ground wire to be carried along with data wires.

If a secondary CAN-bus network is connected to *Channel 2*, care must be taken to ensure that the ground potentials of the two networks can safely be connected through the common ground in the device.

0.3.1.3 Cable shielding

Shielding is not needed in all applications. If shielding is used, it is recommended that a short pig-tail be crimped to the shield end at each connector.

0.3.1.4 CAN ISO 11898-2

ISO 11898-2 defines the basic physical requirements of a high-speed CAN-bus network. Some of these are listed below:

- Max line length (determined by bit-rate)
- Line termination (120 ohm line termination at each end of data line)
- Twisted data lines
- Ground offsets in range -2V to +7V

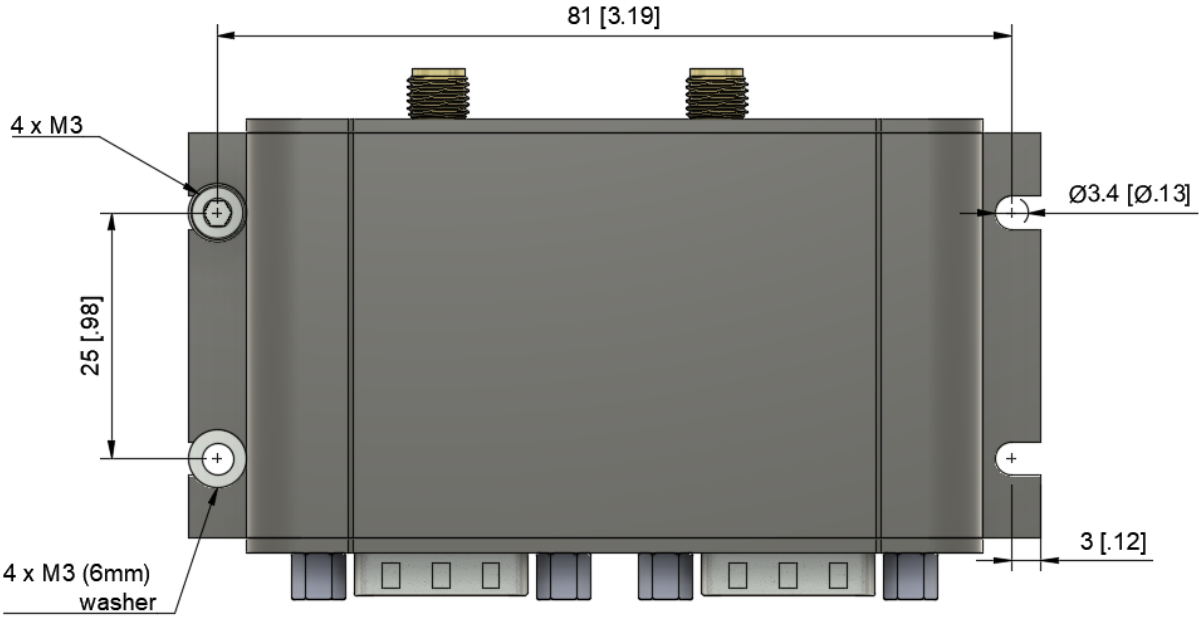
0.3.1.5 CAN-bus stub length

It is recommended that the CAN-bus stub length is kept short. The stub length is defined as the length from the ”main” data line wires to the connection point of the CAN-bus nodes.

0.3.1.6 Mounting

The device should be mounted in a way that minimizes vibration exposure and accounts for the IP-rating of the device.

Hardware version ≥ 00.03 uses flanges for easy and robust mounting. The flanges are designed for 4 x M3 screws and 4 x 6 mm washers.



`Mounting template (PDF)`

0.3.2 Connector

This section contains information on the device connectors.

Table of Contents

0.3.2.1 Pinout

The CANedge uses two D-sub9 connectors for supply, 2 x CAN, 2 x LIN, and 5 V output.

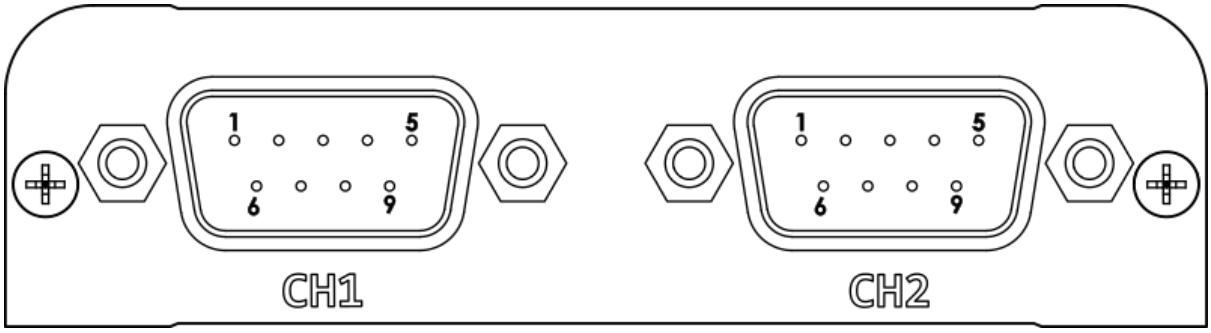


Fig. 1: Front view. Hardware version 00.03.

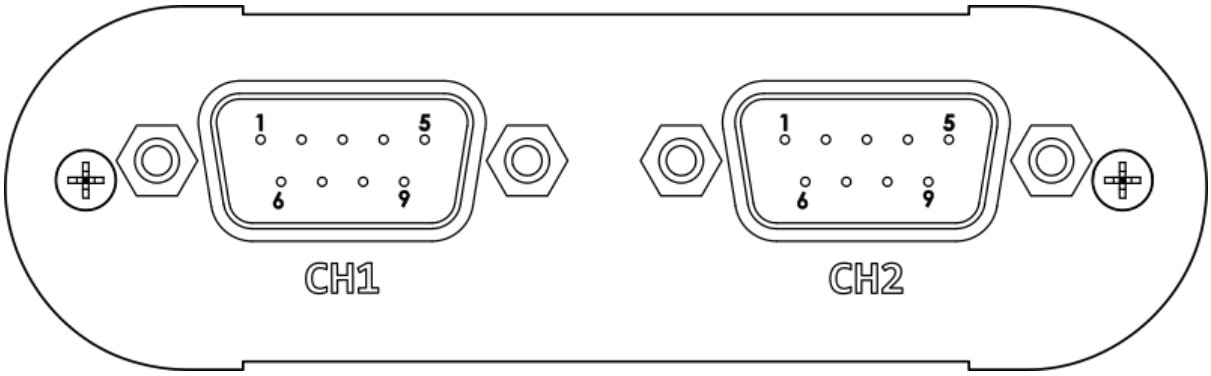


Fig. 2: Front view. Hardware version $\leq$ 00.02

Pin #	Channel 1 (CH1)	Channel 2 (CH2)
1	NC	5 V Supply Output
2	CAN 1 L	CAN 2 L
3	GND	GND
4	LIN Data 1	LIN Data 2
5	NC	NC
6	GND (optional)	GND (optional)
7	CAN 1 H	CAN 2 H
8	NC	NC
9	Supply & LIN1 VBAT	LIN2 VBAT

The hardware 00.00 pinout can be found here.

Supply

The supply (CH1 pin 9) is used to power the device. The supply is internally protected against reverse polarity and low-energy voltage spikes.

Refer to the *Electrical Specification* for more details on the device supply.

Warning

The supply line must be protected against high-energy voltage events exceeding device limits

GND

All GND (ground) pins are connected internally.

5 V Supply Output

The +5 V output can be used to power external devices. The power can be toggled via the device configuration. Refer to the *Electrical Specification* for more details on the 5 V output.

> **Danger**
>
> Connecting external input power to this pin can permanently damage the device

> **Warning**
>
> External protection (such as clamping diodes) must be installed if inductive loads are connected to the 5 V Supply Output

CAN L/H

> **Warning**
>
> CAN-bus requires no common reference (ground). However, it is recommended that GND (ground) is carried along with CAN-L/H to prevent that the common-mode voltage is exceeded (resulting in transceiver damage)

LIN VBAT

The LIN-bus positive reference. Supports systems operating from 4 V to 24 V.

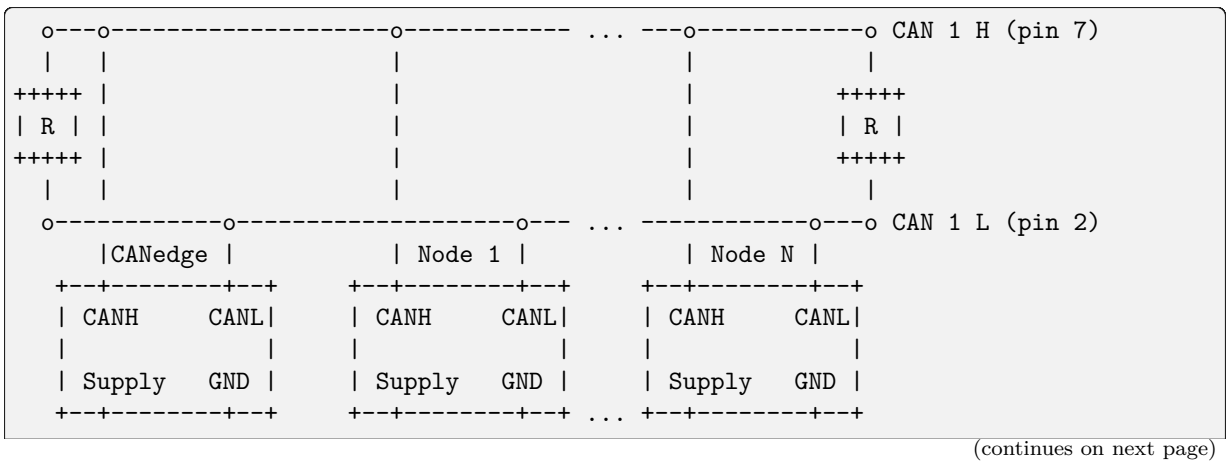
- LIN1 VBAT: Pin is shared with device supply and shares the supply input protection circuit
- LIN2 VBAT: Tolerates voltage spikes up to 48V. Spikes above this can damage the interface

LIN Data

LIN-bus single-wire data line referenced to LIN VBAT.

0.3.2.2 Wiring example

Below example illustrates how the CANedge CAN-bus 1 (channel 1) can be connected.



(continues on next page)

(continued from previous page)

```
    |       |          |       |          |       |
    o-------------------o------------ ... ---o-----------o SUPPLY (pin 9)
            |                   |                  |
            o-------------------o--- ... ------------o--o GND (pin 3)
```

0.3.3 LED

This section contains information on the device LEDs.

The LEDs are located at the back of the device as illustrated below.

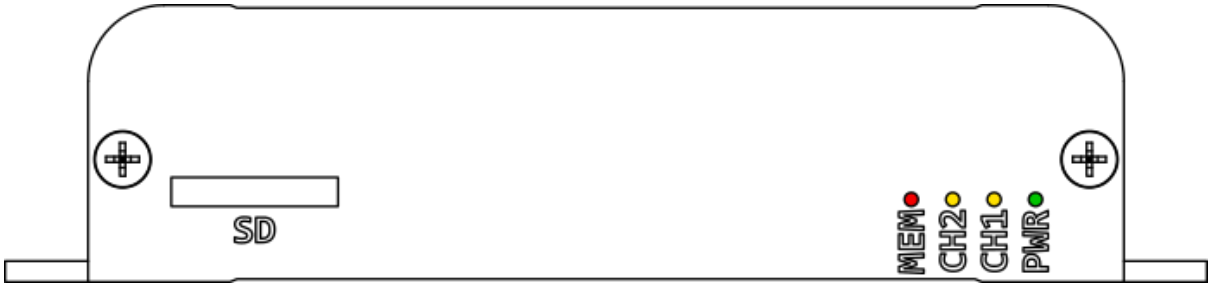


Fig. 3: Back view. Hardware version 00.03.

Fig. 4: Back view. Hardware version $\leq$ 00.02

LED Short Name	LED Color	Main Function
PWR	Green	Power
CH1	Yellow	Bus activity on connector 1 (CH1)
CH2	Yellow	Bus activity on connector 2 (CH2)
MEM	Red	Memory card activity

0.3.3.1 PWR

The *Power* LED is constantly on when the device is in normal operation. An exception is when the firmware is being updated (for more information see *Firmware*).

0.3.3.2 CH1 / CH2

The *Channel 1* / *Channel 2* LEDs indicate bus activity on Channel 1 and 2 respectively.

0.3.3.3 MEM

The *Memory* LED indicates activity on the memory card. Config file parsing, message logging, file upload etc. all generate activity on the memory card.

0.3.4 SD-card

The CANedge uses an extractable SD-card to store the file system (see *Filesystem* for more information).

See *Replacing SD-card* for information on how to replace the SD-card.

> **Warning**
>
> Never extract the SD-card while the device is on. Remove power first and wait a few seconds for the device to turn off.

0.3.4.1 Type

The CANedge uses a specifically selected industrial grade SD-card with special timing constraints to ensure safe shutdown when power is lost.

> **Warning**
>
> The device cannot be guaranteed to work if the pre-installed SD-card is replaced by a card of another type.

0.3.4.2 Lifetime

SD-card memory wears as any other flash based memory. The industrial grade SD-card provided with the CANedge has the following guaranteed minimum endurance numbers:

Size [GB]	TBW[1]	Lifetime @ 1MB/sec [years][2]	Lifetime @ 1MB/min [years]
8	24	0.8	47.9
32	96	3.2	191.5

0.3.5 Enclosure

This section contains information on the device enclosure.

> **Warning**
>
> Opening the enclosure can permanently damage the device due to e.g. ESD (electrostatic discharge) - and improper handling may void the warranty

0.3.5.1 Technical drawings

PDF drawings and 3D STEP files can be found in the online documentation.

0.3.6 Label

This section contains information on the device label.

> **Note**
>
> The QR-code can be scanned to simplify installation of a new device

A unique label is attached to each device. Examples of the labels are illustrated below.

[1] TBW: Terabytes Written
[2] A constant logging rate of 1 MB/sec is likely much much higher than in any practical logging use-case

0.3.6.1 Hardware version 00.03

The label contains the following information:

- Unique device ID: `EA9B650D`
- Hardware version: `00.03`
- Production date in format YYYYWW (WW = week number): `202301`
- QR-code containing production date and device ID: `202301;EA9B650D;XXXXXXXXXXXX`

0.3.6.2 Hardware version $\leq$ 00.02

The label contains the following information:

- Device type: `CANedge1`
- Production date in format YYYYWW (WW = week number): `201930`
- Hardware version: `00.00`
- Unique device ID: `EA9B650D`
- Data matrix (ECC200) containing production date and device ID: `201930;EA9B650D;XXXXXXXXXXXX`

0.4 Configuration

0.4.1 General

This page documents the *general* configuration.

Table of Contents

0.4.1.1 Configuration file fields

This section is autogenerated from the Rule Schema.

Device `general.device`

Meta data `general.device.meta`

Optional meta data string. Displayed in device file and log file headers. Example: Site1; Truck4; ConfigRev12

Type	Min length	Max length
string	0	30

Security `general.security`

Server public key `general.security.kpub`

Server / user ECC public key in base64 format. Shall match the encryption used for all protected fields.

Type	Min length	Max length
string	0	100

Debug `general.debug`

Debug functionality for use during installation and troubleshooting.

System log `general.debug.syslog`

System events logged to the SD-card. The log levels are listed in order of increasing amount of information logged. Should only be enabled if needed during installation or troubleshooting.

Type	Default	Options
integer	1	Disable (0): `0` Error (1): `1` Error + Warning (2): `2` Error + Warning + Info (3): `3`

Restart timer `general.debug.restart_timer`

Number of runtime hours after which the device automatically restarts (set 0 to disable). Example: Set to 24 to restart after one day of runtime.

Type	Default	Minimum	Maximum
integer	0	0	168

0.4.1.2 Configuration explained

This section contains additional information and examples.

Device meta data

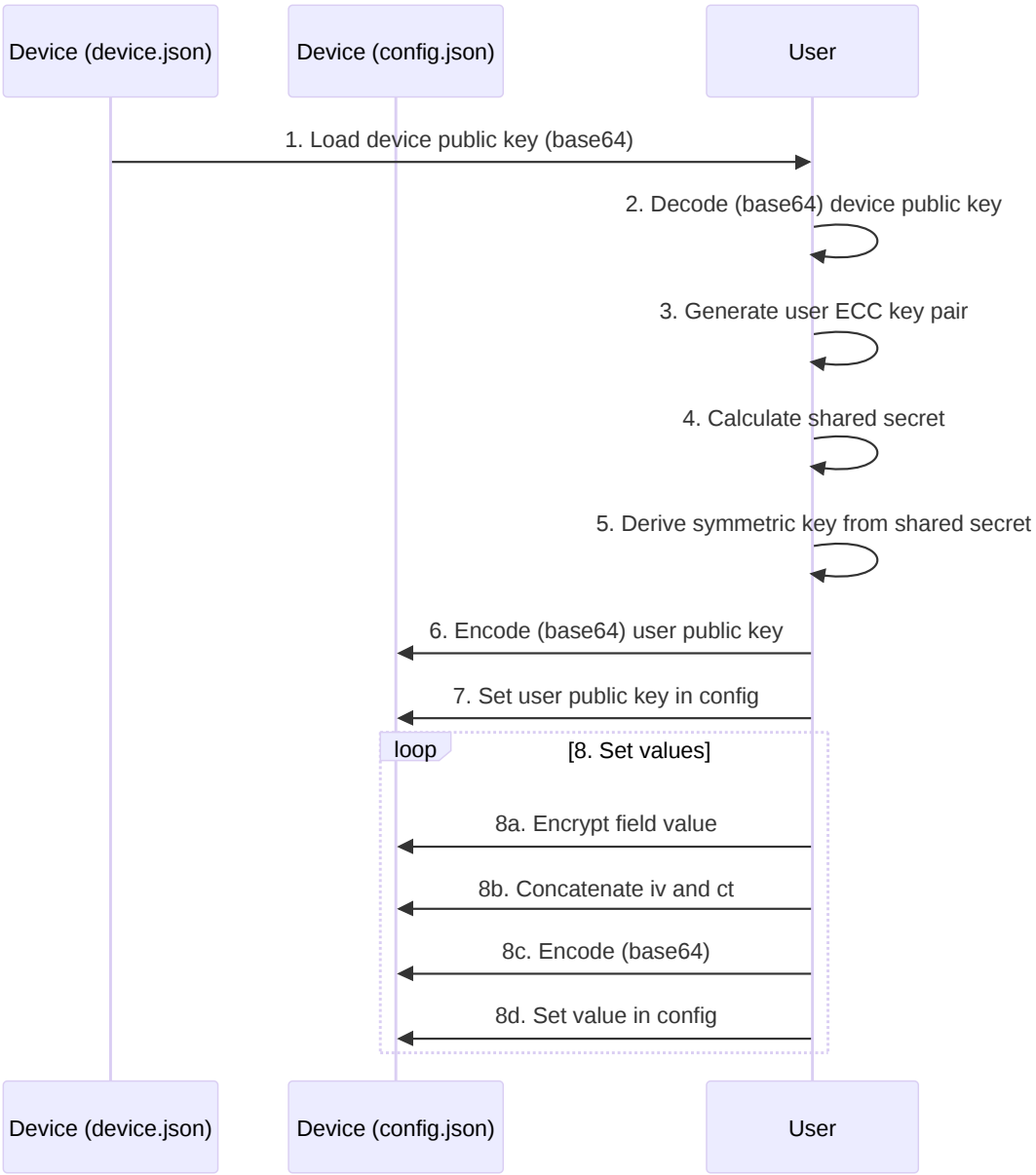
The device meta data is an optional string copied to the `device.json` file and log file headers.

Security

Some configuration field values can be encrypted to hide sensitive data stored in the Configuration File (passwords etc.). In this section, we provide a technical summary and provide resource suggestions for implementing the encryption.

The field encryption feature uses a key agreement scheme based on Elliptic Curve Cryptography (ECC) (similar to the one used in a TLS handshake). The scheme allows the device and user to compute the same shared secret, without exposing any secrets. The shared secret is in turn used to generate a symmetric key, which is used to encrypt / decrypt protected field values.

The following sequence diagram illustrates the process of encrypting configuration fields:



Below we explain the sequence:

1. Load device public key field (`kpub`) from the `device.json` file
2. Decode the device public key (base64)
3. Generate random user key pair (public and private) using curve `secp256r1`
4. Calculate shared secret using device public key and user private key
5. Derive shared symmetric key using HMAC-SHA256 with "config" as data and shared secret as key.

Use the first 16 bytes of the output

6. Encode user public key (used by the device to calculate the same shared symmetric key for decryption)
7. Set the encoded user public key in the device configuration file
8. Use AES-128 CTR to encrypt protected fields using the symmetric key. The resulting initialization vector (iv) and cipher text (ct) are concatenated (iv + ct), base64 encoded and stored in the configuration file

Note

The symmetric key shall match the public key set by the user in the configuration and protected fields shall be encrypted with this symmetric key

Note

By storing the symmetric key it is possible to change specific protected fields - without updating the user public key (and in turn all other protected fields)

Encryption tools

Tools are provided with the CANedge which can be used to encrypt sensitive fields.

Example Python code

You can batch-encrypt passwords across multiple devices using e.g. Python. Below we provide a basic code sample to illustrate how Python can be used to encrypt plain-text data. The example code is tested with `Python 3.7.2` and requires the `pycryptodome` crypto library:

`Python example code`

Debug

System log

A system log can be enabled to output system events to a file (`syslog.txt`) stored on the SD-card. The size of the log file is limited to 1 MB. The user can safely delete the log file at any time.

Note

Log levels 2-3 should only be enabled during installation or troubleshooting

System log verbosity levels:

0. **Disabled**
1. **Error**: Critical issues
2. **Warning (+ Error)**: Temporary or less critical issues
3. **Info (+ Error + Warning)**: Information generated by normal operation

Restart timer

The `restart_timer` can be used to restart the device automatically after a set number of hours. Set to zero to disable.

0.4.2 Logging

This page documents the *logging* configuration

Table of Contents

0.4.2.1 Configuration file fields

This section is autogenerated from the Rule Schema file.

File `log.file`

File split size (1 to 512 MB) `log.file.split_size`

Log file split size in MB. When the file split size is reached a new file is created and the logging continues. Closed log files can be pushed to a server if network is available. Small split sizes may reduce performance.

Type	Default	Minimum	Maximum
integer	50	1	512

File split time period (0 to 86400 seconds, 0 = disable) `log.file.split_time_period`

Log file split time period in seconds relative to midnight (00:00:00). When a split time is reached a new file is created and the logging continues. Closed log files can be pushed to a server if network is available. Small split time periods may reduce performance.

Type	Default	Minimum	Maximum	Multiple of
integer	0	0	86400	10

File split time offset (0 to 86400 seconds) `log.file.split_time_offset`

Log file split time offset in seconds. This value offsets the split_time_period relative to midnight (00:00:00). The set value shall be less than the split_time_period value.

Type	Default	Minimum	Maximum	Multiple of
integer	0	0	86400	10

Cyclic logging `log.file.cyclic`

With cycling logging mode enabled the oldest log file is deleted when the memory card becomes full, allowing the logging to continue.

Type	Default	Options
integer	1	Disable: `0` Enable: `1`

Compression `log.compression`

Level `log.compression.level`

Window size used during optional compression. Larger window sizes yield potentially better compression rates, but may reduce logging performance. Compressed log files need to be decompressed prior to processing.

Type	Default	Options
integer	0	Disable: `0` 256 bytes window: `256` 512 bytes window: `512` 1024 bytes window: `1024`

Encryption `log.encryption`

State `log.encryption.state`

Optional log file encryption. Encrypted log files need to be decrypted prior to processing. Decryption requires your encryption password in plain form - if this is lost, the encrypted data cannot be recovered.

Type	Default	Options
integer	0	Disable: `0` Enable: `1`

Error Frames `log.error_frames`

State `log.error_frames.state`

Specify whether to record error frames. Enabling this can negatively impact performance, as a potentially large number of additional frames may be recorded.

Type	Default	Options
integer	0	Disable: `0` Enable: `1`

0.4.2.2 Configuration explained

This section contains additional information and examples.

File split

File splitting can be based on file size or file size and time:

- `split_time_period = 0`: Split based on size only
- `split_time_period > 0`: Split based on both size and time - whichever is reached first

Limits

The file system limits should be considered when configuring the split size and time:

- SD-card size
- Max `1024` sessions
- Max `256` splits (log files) in each session

Above limits result in a maximum of `1024*256=262144` log files if fully utilised.

If the session count limit is reached, the logger will either:

- Stop logging if cyclic logging is disabled[1]

[1] Logging resumes if files are offloaded via a network connection

- Delete the oldest session if cyclic logging is enabled

If SD-card becomes full (no more space), the logger will either:

- Stop logging if cyclic logging is disabledPage 17, 1
- Delete the oldest split file from the oldest session if cyclic logging is enabled

Compression

Log files can be compressed on the device during logging using a variant of the LZSS algorithm based on heatshrink. Compressed files will have `*.MFC` as file extension. A high window size improves compression rates, but may cause message loss on very busy networks.

The table below lists results for J1939 and OBD data with different window size configurations[2]:

Window size (bytes)	J1939 % (range)	OBD % (range)
256	49.7 (47.1-51.4)	32.0 (30.3-32.8)
512	49.5 (46.3-51.6)	30.2 (29.6-31.1)
1024	41.4 (38.9-45.5)	30.0 (29.6-30.8)

Decompression can be done using an implementation of LZSS or using the tools provided with the CANedge.

Note

The split size set in `split_size` considers the size of the compressed data. I.e. if the split size is 10 MB, the resulting file sizes become 10 MB regardless if compression is used or not.

Encryption

Log files can stored as encrypted (AES-GCM) `*.MFE` files.

Note

It is recommended to use a 40+ character password for proper encryption

Decryption can be done using an implementation of the PBKDF2 algorithm or using the tools provided with the CANedge.

Error Frames

Enabling error frames will log errors across all interfaces, both CAN and LIN. Note that this can decrease the performance of the device due to the added logging load.

For more information on logging of CAN-bus errors, see configuration/can/error:CAN errors.

0.4.3 Real-Time-Clock

This page documents the *real-time-clock* configuration

Table of Contents

[2] Compressed size in percentage of original. Lower is better.

0.4.3.1 Configuration file fields

This section is autogenerated from the Rule Schema file.

Real-Time Clock (RTC) `rtc`

Time synchronization method `rtc.sync`

Internal real-time-clock synchronization method. The real-time-clock is maintained when the device is off.

Type	Default	Options
integer	0	Retain current time: `0` Manual update: `1` CAN-bus: `3`

Time zone (UTC-12 to UTC+14) `rtc.timezone`

Adjustment in full hours to the UTC time. Includes daylight savings time if applicable.

Type	Default	Minimum	Maximum
integer	0	-12	14

Adjustment (-129600 to 129600 seconds) `rtc.adjustment`

Adjustment in seconds to the UTC time. Can be used for fine tuning the internal time.

Type	Default	Minimum	Maximum
integer	0	-129600	129600

0.4.3.2 Configuration explained

This section contains additional information and examples.

The CANedge uses a real-time clock (RTC) with battery backup, which allows it to retain the absolute date & time when the device is not powered. The RTC enables the CANedge to add absolute timestamps to recorded messages.

Time-zone changes and minor adjustments can be done via the `timezone` and `adjustment` fields.

Synchronization methods (`sync`)

The RTC time can either be *retained*, *manually set* or *synchronized via CAN-bus*.

> **Note**
>
> When using an external synchronization source, the *TimeExternal signals* can be used to confirm that the device correctly receives and understands the time synchronization information.

Manual update

Manually changing the RTC is *only needed* if the RTC time has been completely reset (e.g. after a battery replacement). The following sequence explains how the RTC can be manually set:

1. Select the *manual* `sync` method and set the current UTC time
2. Power on the device and wait a few seconds to allow the device to read the manually set time
3. Power off the device
4. Change the `sync` method to *retain* the current time
5. Power on the device again
6. Verify that the new absolute time is now correctly retained across power cycles
7. Set timezone (`timezone`) and do minor adjustments (`adjustment`) if needed

Note

The internally stored *session counter* is lost when the battery is removed. See *Setting session counter* for information on how to set the session counter.

CAN-bus

The RTC can be synchronized based on a CAN-bus message. The interpretation of message data signals is configurable.

Time information can be provided via either physical CAN-bus channel.

The synchronization method depends on the time difference between the RTC time and the external time provided via CAN-bus:

- Time difference exceeds `tolerance`: The RTC time is directly set to the external time (discrete jump in time)
- Time difference within `tolerance`: The RTC time slowly tracks the external time (continuous time)[1]

The synchronization message data is assumed to include the external time and optionally a *valid* flag indicating if the external time should be applied or not:

- *Valid signal* (optional): 1: Time signal is valid, else: Time signal is invalid
- *Time signal* (mandatory): The current UTC time as *Epoch* (floating-point number of seconds since 01/01/1970 00:00:00 UTC)

Warning

Avoid using a high-frequency CAN-bus message for time synchronization. If the frequency of the time message is high, consider using pre-scalers to reduce the period to e.g. 1 minute.

The configuration of the signals uses a concept similar to that used by *.DBC* files. In case a *.DBC* file is available (describing the interpretation of the synchronization message), the information from the file can be used directly for configuration. For more information see Section configuration/signal:Signal.

Example 1: Using both the *valid* signal and *time* signal (time message generated by a *CANmod.GPS* device).

- The valid signal is 1 bit starting at index 0. The factor and offset are chosen such that the decoded signal becomes *1* when the time signal is valid.

[1] Continues tracking requires that an updated external time is available at least once each hour

- The time signal is 40 bit starting at index 8. After applying *factor* and *offset* the result becomes Epoch in seconds.

Signal	Type	Byteorder	Bitpos	Length	Factor	Offset
Valid	Unsigned	Intel	0	1	1	0
Time	Unsigned	Intel	8	40	0.001	1577840400

Example 2: Same as Example 1 but without using the *valid* signal.

- The valid signal length is set to 0. With a factor of 0 and offset of 1, the result always becomes 1 (valid)

Signal	Type	Byteorder	Bitpos	Length	Factor	Offset
Valid	Unsigned	Intel	0	0	0	1
Time	Unsigned	Intel	8	40	0.001	1577840400

Note

If a valid signal is not included in the data, a constant valid signal can be enforced by setting the `factor` to `0` and `offset` to `1`.

0.4.4 Secondary port

This page documents the *secondary port* configuration

Table of Contents

0.4.4.1 Configuration file fields

This section is autogenerated from the Rule Schema file.

Power schedule `secondaryport.power_schedule`

The daily power schedule is defined by a number of power-on from/to intervals. Define no power-on intervals to keep always off. Define one interval with from/to both set to 00:00 to keep always on. Time format is HH:MM (1 minute resolution)

Type	Default	Max items
array	[]	5

Item `secondaryport.power_schedule.item`

From `secondaryport.power_schedule.item.from`

Power-on FROM time in format HH:MM. Shall be before power-on TO time. E.g. at midnight 00:00

Type	Default
string	00:00

To `secondaryport.power_schedule.item.to`

Power-on TO time in format HH:MM. Shall be after power-on FROM time. E.g. at midday 12:00.

Type	Default
string	00:00

0.4.4.2 Configuration explained

This section contains additional information and examples.

> **Note**
>
> Power out scheduling has resolution of 1 min and 1 min tolerance

> **Note**
>
> Power scheduling uses adjusted local time (as set in the configuration)

Example: Secondary port power is scheduled to be on daily in the interval `00:00-04:00` and `12:00-16:00`. Secodary port configuration:

```
"secondaryport": {
  "power_schedule": [
    {
      "from": "00:00",
      "to": "04:00"
    },
    {
      "from": "12:00",
      "to": "16:00"
    }
  ]
}
```

The power is turned off when the time changes from 03:59 to 04:00 and 15:59 to 16:00.

0.4.5 CAN

This page documents the *CAN* configuration.

The CANedge supports two physical CAN-bus channels and one internal virtual channel. The internal channel is used for *internally generated signals*.

The configuration options of CAN Channel 1 and CAN Channel 2 are identical[1]. The internal channel supports a limited set of configuration options.

The CANedge can detect and log CAN-bus errors if enabled in *Logging*. For more information, see configuration/can/error:CAN errors.

The CAN configuration is split into the following sections:

[1] All channels can be configured individually.

0.4.5.1 General

This page documents the *general* configuration

Table of Contents

Configuration file fields

This section is autogenerated from the Rule Schema file.

Can.general `can.general`

Reception (rx) initial state `can.general.rx_state`

The initial state of CAN-bus reception. Can be changed using the control signal.

Type	Default	Options
integer	1	Disable: `0` Enable: `1`

Transmission (tx) initial state `can.general.tx_state`

The initial state of CAN-bus transmissions. Can be changed using the control signal.

Type	Default	Options
integer	1	Disable: `0` Enable: `1`

Configuration explained

This section contains additional information and examples.

The `rx_state` / `tx_state` initial states are primarily used in conjunction with the *Control Signal*. E.g. transmission of messages from the CANedge can be initialized as *disabled* using `tx_state` and later changed to *enabled* by a defined Control Signal.

0.4.5.2 Physical

This page documents the *Physical* (*PHY*) configuration.

The *Physical* section configures parameters related to the CAN-bus.

Configuration file fields

This section is autogenerated from the Rule Schema file.

Mode `can.phy.mode`

Device CAN bus mode. Configures how the device interacts with the bus. In Normal mode, the device can receive, acknowledge and transmit frames. In Restricted mode, the device can receive and acknowledge, but not transmit frames. In Bus Monitoring mode, the device can receive, but not acknowledge or transmit frames. It is recommended to always use the most restrictive mode possible.

Type	Default	Options
integer	1	Normal (receive, acknowledge and transmit): `0` Restricted (receive and acknowledge): `1` Monitoring (receive only): `2`

Automatic retransmission `can.phy.retransmission`

Retransmission of frames that have lost arbitration or that have been disturbed by errors during transmission.

Type	Default	Options
integer	1	Disable: `0` Enable: `1`

CAN FD specification `can.phy.fd_spec`

Configures the CAN FD specification used by the device. Shall match the specification used by the CAN bus network.

Type	Default	Options
integer	0	ISO CAN FD (11898-1): `0` non-ISO CAN FD (Bosch V1.0.): `1`

Bit-rate configuration mode `can.phy.bit_rate_cfg_mode`

Configures how the CAN bus bit-rate is set. Modes Auto-detect and Bit-rate support all standard bit-rates. Non-standard bit-rate configuration can be set using Bit-timing. It is recommended to set the bit-rate manually if it is known.

Type	Default	Options
integer	0	Auto-detect: `0` Bit-rate (simple): `1` Bit-timing (advanced): `2`

Configuration explained

This section contains additional information and examples.

Mode

The `mode` field configures to what extend the CANedge is allowed to communicate on the CAN-bus.

> **Note**
>
> It is recommended to use the most restrictive mode possible.

Retransmission

The `retransmission` configures how the CANedge should react when message transmissions fail. Failed transmissions can either be aborted or retried.

Bit-rate configuration mode

The `bit_rate_cfg_mode`, selects how the bit-rate is configured. In most cases, a simple *bit-rate* can be set. For more advanced cases, the more extensive *bit-timing* can be used.

Bit-rate / bit-timing

The input clock to the CAN-bus controllers is set to 40 MHz (480 MHz prescaled by 12).

The bit-rate modes `Auto-detect` and `Bit-rate (simple)` support the following list of bit-rates[1][2]:

Bitrate	BRP	Quanta	Seg1	Seg2	SJW
5k	100	80	63	16	4
10k	50	80	63	16	4
20k	25	80	63	16	4
33.333k	10	120	95	24	4
47.619k	8	105	83	21	4
50k	10	80	63	16	4
83.333k	4	120	95	24	4
95.238k	4	105	83	21	4
100k	5	80	63	16	4
125k	4	80	63	16	4
250k	2	80	63	16	4
500k	1	80	63	16	4
800k	1	50	39	10	4
1M	1	40	31	8	4
2M	1	20	15	4	4
4M	1	10	7	2	2

In `Auto-detect` mode, the device attempts to determine the bit-rate from the list of detectable bit-rates. Depending on factors such as data patterns, bit-rate deviation etc. it may not always be possible to detect the bit-rate automatically.

Warning

It is recommended to set the bit-rate manually when possible

Warning

Bit-rate auto-detect cannot be used to detect a CAN FD switched bit-rate

In mode `Bit-timing (advanced)`, the bit-rate timing can be set directly. The following equations can be used to calculate the bit-timing fields:

- Input clock: $CLK = \frac{480000000}{12} = 40000000 = 40$ MHz
- Quanta: $Q = 1 + SEG_1 + SEG_2$
- Bit-rate: $BR = \frac{CLK/BRP}{Q}$

[1] All bit-rate configurations use a Sample-Point (SP) of 80%.
[2] The CAN-FD Secondary-Sample-Point (SSP) is set to the same as the CAN-FD Sample-Point (SP).

- Sample point: $SP = 100 \cdot \frac{1+SEG_1}{Q}$

Examples

Example: Matching bit-timing settings based on different input clock frequency (`CLK`).

Settings to match (based on a 80 MHz input clock):

- Bit-rate: 2M
- Quanta: 40
- SEG1: 29
- SEG2: 10
- Sample point: 75%

Above settings are based on an input clock with frequency:

$$CLK = BR \cdot Q = 2000000 \cdot 40 = 80 \text{ MHz}$$

The CANedge uses a 40 MHz input clock. To obtain a bit-rate of 2 M with a 40 MHz input clock, the number of quanta is calculated as:

$$Q = \frac{CLK/BRP}{BR} = \frac{40000000/1}{2000000} = 20$$

To obtain a sampling point of 75%, SEG1 is calculated as:

$$SEG_1 = \frac{SP \cdot Q}{100} - 1 = \frac{75 \cdot 20}{100} = 14$$

Now, SEG2 is calculated as:

$$SEG_2 = Q - SEG_1 - 1 = 20 - 14 - 1 = 5$$

The equivalent bit-timing settings using the 40 MHz input clock of the CANedge becomes:

- BRP: 1
- SEG1: 14
- SEG2: 5

0.4.5.3 Filter

This page documents the *filter* configuration

Configuration file fields

This section is autogenerated from the Rule Schema file.

Receive filters `can.filter`

Filter remote request frames `can.filter.remote_frames`

Controls if remote request frames are forwarded to the message filters. If `Reject` is selected, remote request frames are discarded before they reach the message filters.

Type	Default	Options
integer	0	Reject: `0` Accept: `1`

Id `can.filter.id`

Filters are checked sequentially, execution stops with the first matching filter element. Max 128 11-bit filters and 64 29-bit filters.

Max items
192

Name `can.filter.id.name`

Optional filter name.

Type	Max length
string	16

State `can.filter.id.state`

Disabled filters are ignored.

Type	Default	Options
integer	1	Disable: `0` Enable: `1`

Type `can.filter.id.type`

Action on match, accept or reject message.

Type	Default	Options
integer	0	Acceptance: `0` Rejection: `1`

ID format `can.filter.id.id_format`

Filter ID format. Filters apply to messages with matching ID format.

Type	Default	Options
integer	0	Standard (11-bit): `0` Extended (29-bit): `1`

Filter method `can.filter.id.method`

The filter ID matching mechanism.

Type	Default	Options
integer	0	Range: 0 Mask: 1

From (range) / ID (mask) (HEX) `can.filter.id.f1`

If filter method is Range, this field defines the start of range. If filter method is Mask, this field defines the filter ID.

Type	Default	Max length
string	0	8

To (range) / mask (mask) (HEX) `can.filter.id.f2`

If filter method is Range, this field defines the end of range. If filter method is Mask, this field defines the filter mask.

Type	Default	Max length
string	7FF	8

Configuration explained

This section contains additional information and examples.

The following uses a mix of binary, decimal and hexadecimal number bases. For more information on the notation used, see to *Number bases*.

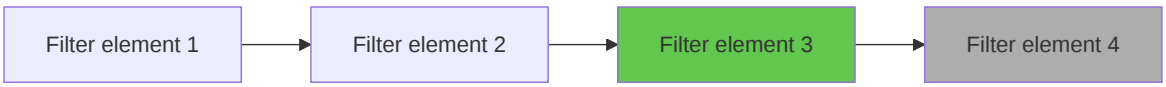
Note

In the following, it is convenient to do some calculations using binary numbers (base 2). However, the configuration file generally accepts either decimal or hexadecimal numbers.

Filter processing

The filter elements in the list of filters are processed sequentially starting from the first element. Processing stops on the first filter match or when the end of the filter list is reached.

Example: A message matches filter element 3. Filter element 4 is not evaluated.



Messages matching no filters are rejected as default.

Note

The default Configuration File contains filters accepting all incoming CAN messages

Filter name

Filters can be assigned a optional *name*. The name is not used when processing a filter.

Filter state

The *state* of filter elements can be *Enable* or *Disable*. Disabled filter elements are ignored, as if they are not in the list of filters. If there are no enabled filters in the list then all messages are rejected.

By disabling a filter element (instead of deleting the element) it can be easily enabled at a later time.

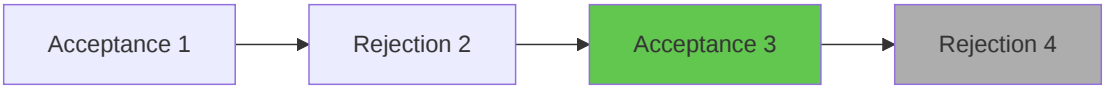
Filter types

Filter elements can be either *Acceptance* or *Rejection*:

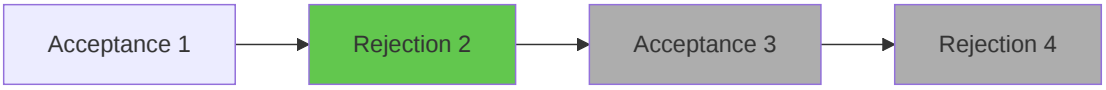
- If a message matches an *Acceptance* filter it is accepted
- If a message matches a *Rejection* filter it is discarded
- If a message does not match a filter, the next filter in the list is processed

The filter list can hold a combination of *Acceptance* and *Rejection* filter elements. The first matching filter element determines if a message is accepted or rejected. *Acceptance* and *Rejection* filters can be combined to generate a complex message filtering mechanism.

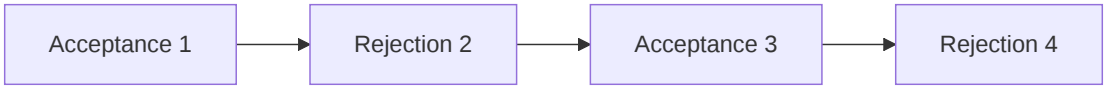
Example: A message matches acceptance filter 3. Rejection filter 4 is not evaluated. The message is accepted.



Example: A message matches rejection filter 2. The following filters are not evaluated. The message is rejected.



Example: A message does not match any filters. The message is rejected.



Filter ID format

A filter can only match with messages using the same ID format. Standard (11-bit) ID filters can match standard ID messages and extended (29-bit) ID filters can match extended ID messages.

Filter method

Acceptance and *Rejection* filters can be defined by range or mask. In either case, both the message type (standard / extended) and ID are compared to the filter.

Filter range method

With the *Range* method, the filter defines a range of IDs which are compared to the message ID. Message IDs within the range (both start and end included) match the filter.

The fields *F1* and *F2* respectively define the start (from) and the end (to) of the range.

Example: Standard ID filter with range from = 1, to = 10:

ID format	ID (DEC)	Match
Standard	0	No
Standard	1	Yes
Standard	10	Yes
Standard	11	No
Extended	1	No

Filter mask method

With the *Mask* method, the filter defines an ID and Mask which are compared to the message ID.

The fields *F1* and *F2* respectively define the *Filter-ID* and the *Filter-mask*.

A message matches a mask filter if the following condition is true[1]:

```
filter_id & filter_mask == message_id & filter_mask
```

Below provides some practical examples of filters using the *mask* method.

Example: Filter configuration which accepts one specific message ID: $2000_{10} = 11111010000_2$. The filter ID is set to the value of the message ID to accept. The filter mask is set to all ones, such that all bits of the filter are considered, as given in (1).

$$\begin{array}{ll} \text{Filter ID} & 11111010000_2 \\ \text{Filter mask} & \&11111111111_2 \\ \hline \text{Masked filter} & 11111010000_2 \end{array} \quad (1) \qquad \begin{array}{ll} \text{Message ID} & 11111010000_2 \\ \text{Filter mask} & \&11111111111_2 \\ \hline \text{Masked ID} & 11111010000_2 \end{array} \quad (2)$$

To test if the message passes the filter, we apply the filter mask to the message ID as given in (2). The masked filter and the masked ID are equal - the message matches the filter.

Example: Filter configuration which accepts two message IDs:

- $2000_{10} = 11111010000_2$
- $2001_{10} = 11111010001_2$

Note that the two binary numbers are identical except for the rightmost bit. To design a filter which accepts both IDs, we can use the mask field to *mask out* the rightmost bit - such that it is not considered when the filter is applied. In (1) the mask is set such that the rightmost bit is not considered (indicated by red color).

$$\begin{array}{ll} \text{Filter ID} & 11111010000_2 \\ \text{Filter mask} & \&1111111111\mathbf{0}_2 \\ \hline \text{Masked filter} & 11111010000_2 \end{array} \quad (1) \qquad \begin{array}{ll} \text{Message ID} & 11111010001_2 \\ \text{Filter mask} & \&1111111111\mathbf{0}_2 \\ \hline \text{Masked ID} & 11111010000_2 \end{array} \quad (2)$$

To test if the messages pass the filter, we apply the mask to the message ID 11111010001_2 as given in (2). The masked filter and the masked ID are equal - the message matches the filter. Note that both

[1] `&` is used as the bitwise AND operation

11111010000_2 and 11111010001_2 match the filter, as the rightmost bit is not considered by the filter (the rightmost bit is masked out).

Example: J1939 - filter configuration which accepts PGN 61444 (EEC1) messages.

J1939 message frames use 29-bit CAN-IDs. The Parameter Group Number (PGN) is defined by 18 of the 29 bits. The remaining 11 bits define e.g. the priority and source-address of the message. It is often useful to configure a filter to accept a specific PGN while ignoring the remaining 11 bits - this can be done using the filter mask.

Below, the left red bits represent the 3-bit priority, the green bits the 18-bit PGN and the right red bits the 8-bit source address of the 29-bit CAN-ID.

$$000111111111111111111100000000_2 = 3FFFF00_{16}$$

Message ID bits in positions with zero bits in the filter mask are ignored. By using $3FFFF00_{16}$ as filter mask, the priority and source are ignored.

To specifically accept PGN 61444 ($F004_{16}$) messages, the message ID is set to $F00400_{16}$ - note the the final 8-bit 00_{16} represents the source address which is ignored by the filter mask (these bits can be any value).

Filter mask $3FFFF00_{16}$ can be used for all J1939 PGN (PDU2) messages. To accept specific PGNs, the message ID is adjusted. To accept one specific PGN (as in the example above), the message ID is set to the specific PGN with 00_{16} appended to represent the ignored source address field.

Filter list examples

Below examples demonstrate how filters can be combined into a list of filters.

Example: The filter list is set up to accept standard messages with **even** IDs in range $500_{10} - 1000_{10}$ (500, 502, ... 998, 1000):

The following two filters are used to construct the wanted filter mechanism:

- Rejection filter which rejects all odd message IDs
- Acceptance filter which accepts all message IDs in range $500_{10} - 1000_{10}$

The rejection filter is setup to reject all odd messages by using *Mask* filtering. The filter is set up with:

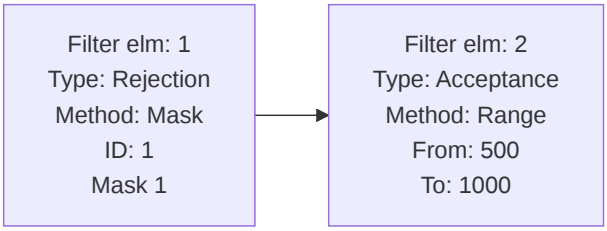
- Filter ID: $1_{10} = 00000000001_2$
- Filter Mask: $1_{10} = 00000000001_2$

Above rejection filter rejects all messages with the rightmost bit set (all odd IDs).

The acceptance filter is set up to accept all messages in range $500_{10} - 1000_{10}$ by using *Range* filtering. The filter is set up with:

- Filter from: 500_{10}
- Filter to: 1000_{10}

The filter list is constructed with the rejection filter first, followed by the acceptance filter.



Note that messages are first processed by the rejection filter (rejects all odd messages), then proccessed by the acceptance filter (accepts all message in range). If none of the filters match, the default behavior is to reject the message. It is in this case important that the rejection filter is placed before the acceptance filter in the list (processing stops on first match).

Filter list test table:

Message ID	Filter elm 1	Filter elm 2	Result
498_{10}	Ignore	Ignore	Reject
499_{10}	Reject		Reject
500_{10}	Ignore	Accept	Accept
501_{10}	Reject		Reject
999_{10}	Reject		Reject
1000_{10}	Ignore	Accept	Accept
1001_{10}	Reject		Reject
1002_{10}	Ignore	Ignore	Reject

Message Prescaling

Message prescaling can be used to reduce the number of received messages for a given message ID. Prescaling is applied to the messages accepted by the associated filter. The list of filters can be assigned a mixture of prescaler types.

Applying filters can dramatically reduce log file size, resulting in prolonged offline logging and reduced data transfer time and size.

The prescaling type can be set to:

- `None`: Disables prescaling
- `Count`: Prescales based on message occurrences
- `Time`: Prescales based on message period time
- `Data`: Prescales based on changes in the message data payload

The first message with a given ID is always accepted regardless of prescaling type.

Note

A maximum of 100 unique message IDs can be prescaled for each CAN-bus channel (the first 100 IDs received by the CANedge). Additional unique IDs are not prescaled

Count

Count prescaling reduces the number of messages with a specific ID by a constant factor (prescaling value). A prescaling value of 2 accepts every 2nd message (with a specific ID), a value of 3 every 3rd and so on up to 256[2].

Example: Count prescaling applied to ID 600_{10} with a scaling value of 3.

ID (DEC)	ID occurrences	Result
600_{10}	1	Accept
600_{10}	2	Reject
600_{10}	3	Reject
600_{10}	4	Accept
600_{10}	5	Reject

Time

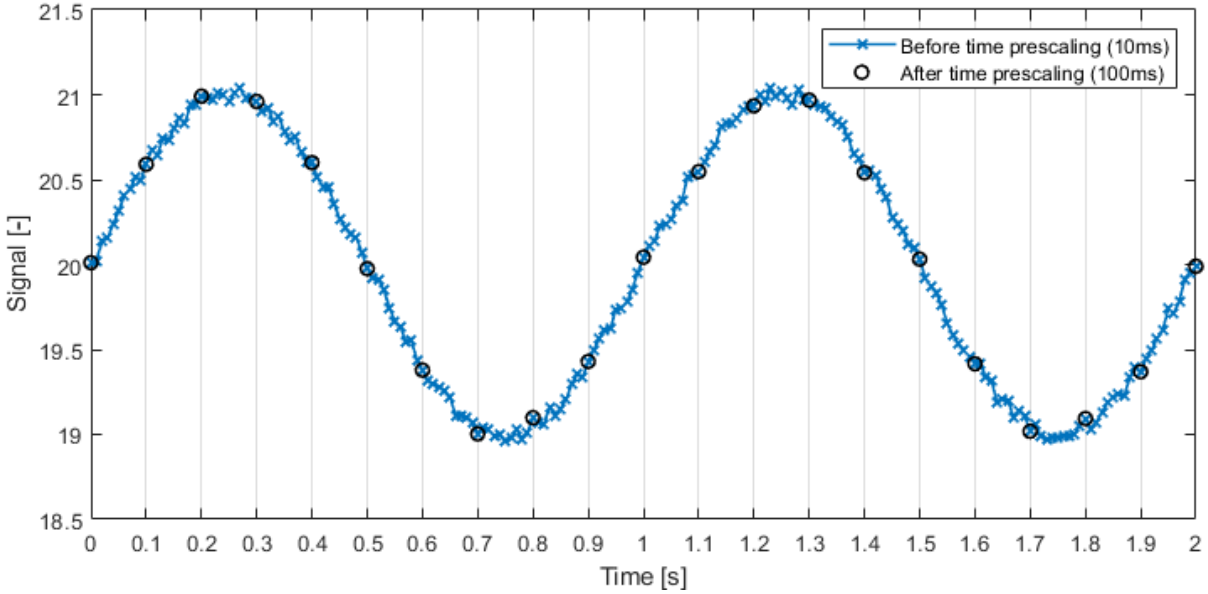
Time prescaling sets a lower limit on the time interval (period time) of a specific message ID. This is done by rejecting messages until at least the prescaler time has elapsed[3]. The prescaler timer is reset

[2] A scaling factor of 1 effectively disables prescaling
[3] Note that messages are not *resampled* to a specific fixed period time

each time a message is accepted. The prescaling value is set in milliseconds[4] with a valid range 1-4194304 (0x400000).

This prescaler type is e.g. useful if a slowly changing signal (low frequency signal content) is broadcasted on the CAN-bus at a high frequency[5].

Example: A slowly changing temperature measurement broadcasted every 10 ms (100Hz). Prescaled to a minimum time interval of 100ms (prescaler value set to 100).



Example: Time prescaling applied to ID 700_{10} with a time interval of 1000ms.

ID (DEC)	Message timestamp [ms]	Prescaler timer [ms]	Result
700_{10}	**200**	0	Accept
700_{10}	700	500	Reject
700_{10}	1000	800	Reject
700_{10}	**1200**	1000 -> 0 (reset)	Accept
700_{10}	1300	100	Reject
700_{10}	**3200**	2000 -> 0 (reset)	Accept
700_{10}	**4200**	1000 -> 0 (reset)	Accept
700_{10}	**5200**	1000 -> 0 (reset)	Accept

Data

Data prescaling can be used to only accept messages when the data payload changes. A mask can be set to only consider changes in one or more specific data bytes. The mask works on a byte level. The mask is entered in hex up to 8 bytes long (16 hex characters). Each byte contains 8 bits, allowing for the mask to be applied to any of the maximum 64 data bytes (CAN FD).

This prescaler can be used to only receive a message when a part of the payload has changed.

Examples of data masks:

- `""`: An empty mask triggers on any data change (equivalent to mask value `FFFFFFFFFFFFFFFF`)
- `1`: Triggers on changes to the first data byte (binary `1`)
- `2`: Triggers on changes to the second data byte (binary `10`)
- `3`: Triggers on changes to the first or second data byte (binary `11`)

[4] It is not possible to do sub-millisecond time prescaling
[5] Higher frequency than needed to get a good representation of the signal content

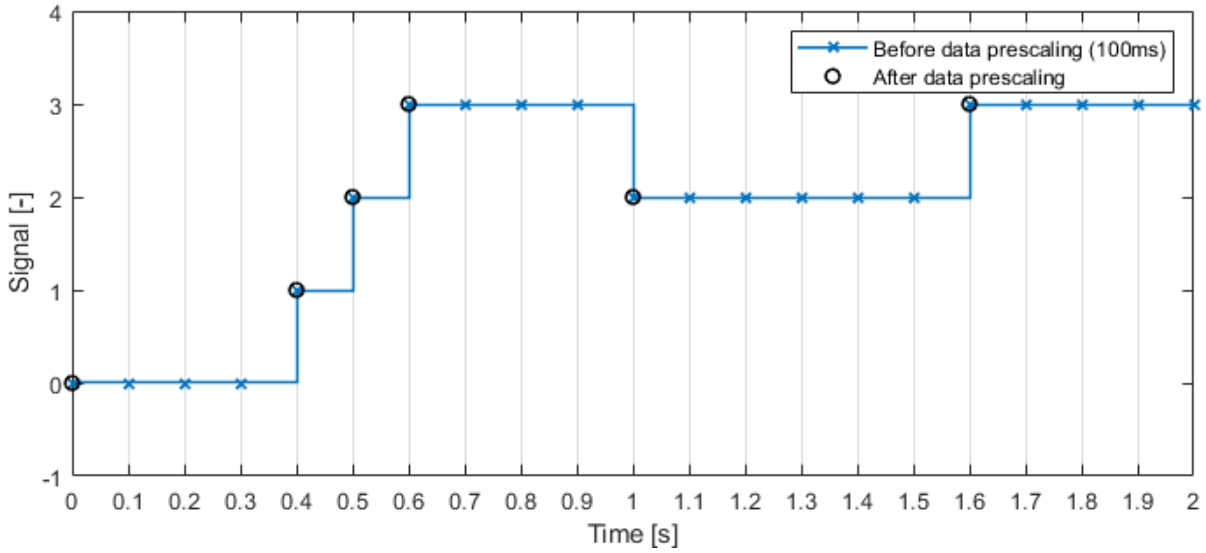
- `9`: Triggers on changes to the first or fourth data byte (binary `1001`)
- `FF`: Triggers on changes to any of the first 8 data bytes (binary `11111111`)
- `100`: Triggers on changes to the 9th data byte (binary `100000000`)

If the data payload contains more data bytes than entered in the mask, then changes to the additional bytes are ignored by the prescaler.

Warning

Data prescaling assumes that a the payload length of a specific message ID is constant

Example: A discretely changing signal is broadcast every 100 ms (10Hz). A data prescaler is used such that only changes in the signal are logged.



Example: Data prescaling applied to ID 800_{10} with empty mask (all changes considered). D0-D3 is a 4-byte payload (with D0 the first data byte).

ID (DEC)	D0	D1	D2	D3	Result
800_{10}	00	11	22	33	Accept
800_{10}	00	11	22	33	Reject
800_{10}	00	**BB**	22	33	Accept
800_{10}	**AA**	BB	22	33	Accept
800_{10}	AA	BB	22	**DD**	Accept
800_{10}	AA	BB	22	DD	Reject

Example: Data prescaling applied to ID 800_{10} with mask `1` (considering only changes to the 1st data byte). D0-D3 is a 4-byte payload (with D0 the first data byte).

ID (DEC)	D0	D1	D2	D3	Result
800_{10}	00	11	22	33	Accept
800_{10}	00	11	22	33	Reject
800_{10}	00	**BB**	22	33	Reject
800_{10}	**AA**	BB	22	33	Accept
800_{10}	AA	BB	22	**DD**	Reject
800_{10}	AA	BB	22	DD	Reject

Example: Data prescaling applied to ID 800_{10} with mask `8` (considering only changes to the 4th data byte). D0-D3 is a 4-byte payload (with D0 the first data byte).

ID (DEC)	D0	D1	D2	D3	Result
800_{10}	00	11	22	33	Accept
800_{10}	00	11	22	33	Reject
800_{10}	00	**BB**	22	33	Reject
800_{10}	**AA**	BB	22	33	Reject
800_{10}	AA	BB	22	**DD**	Accept
800_{10}	AA	BB	22	DD	Reject

Example: Data prescaling applied to ID 800_{10} with mask `9` (considering only changes to the 1st or 4th data byte). D0-D3 is a 4-byte payload (with D0 the first data byte).

ID (DEC)	D0	D1	D2	D3	Result
800_{10}	00	11	22	33	Accept
800_{10}	00	11	22	33	Reject
800_{10}	00	**BB**	22	33	Reject
800_{10}	**AA**	BB	22	33	Accept
800_{10}	AA	BB	22	**DD**	Accept
800_{10}	AA	BB	22	DD	Reject

0.4.5.4 Transmit

This page documents the *transmit* configuration.

The CANedge can be configured to automatically schedule and transmit a list of static messages. Each transmit message can be configured as either *single-shot* or *periodic*.

Configuration file fields

This section is autogenerated from the Rule Schema file.

Transmit messages `can.transmit`

List of CAN bus messages transmitted by the device. Requires a CAN-bus physical mode supporting transmissions.

Type	Max items
array	64

Item `can.transmit.item`

Name `can.transmit.item.name`

Optional transmit message name.

Type	Max length
string	16

State `can.transmit.item.state`

Disabled transmit messages are ignored.

Type	Default	Options
integer	1	Disable: `0` Enable: `1`

ID Format `can.transmit.item.id_format`

ID format of the transmit message.

Type	Default	Options
integer	0	Standard (11-bit): `0` Extended (29-bit): `1`

Frame format `can.transmit.item.frame_format`

Frame format of the transmit message.

Type	Default	Options
integer	0	Standard: `0` Standard RTR: `2` FD: `1`

Bit-Rate Switch `can.transmit.item.brs`

Determines if an FD message is transmitted using a switched bit-rate.

Type	Default
integer	0

Include in log `can.transmit.item.log`

Determines if the transmitted message is included in the log file.

Type	Default	Options
integer	0	Disable: `0` Enable: `1`

Period (10 ms steps) `can.transmit.item.period`

Time period of the message transmission. 0: single shot, >0: periodic. Unit is ms.

Type	Minimum	Maximum	Multiple of
integer	0	4294967290	10

Delay (10 ms steps) `can.transmit.item.delay`

Offset message within the period or delay a single shot message. If multiple messages are transmitted by the device, it is recommended to offset each separately to reduce peak load on bus. If period > 0, delay < period. If single-shot, delay can be up to max value. Unit is ms.

Type	Minimum	Maximum	Multiple of
integer	0	4294967290	10

Message ID (hex) `can.transmit.item.id`

ID of message to transmit in hex. Example: 1FF.

Type
string

Messages Data (hex) `can.transmit.item.data`

Data bytes of message to transmit. RTR frames only use the number of bytes do determine the DLC. Example: 01020304 or 0102030405060708.

Type	Max length
string	128

Configuration explained

This section contains additional information and examples.

Period and delay

Transmit messages can be configured as either *single-shot* or *periodic*.

The *period* value determines if a message is single-shot (0) or periodic (>0). The interpretation of the *delay* value depends on whether the message is single-shot or periodic (see more below).

Note

When possible, it is recommended to spread multiple transmit messages in time by using the *delay* value.

Single-shot

A transmit message is *single-shot* when the *period* is set to 0. In this case, the *delay* is the time between boot-up[1] and the single-shot transmission. Delaying a single-shot message can be useful if e.g. the receiving node has a long boot-up time. The maximum value allows for a delay of up to several hours.

Example: Four *single-shot* transmit messages are configured with different *delay* values.

#	Period [ms]	Delay [ms]
A	0	0
B	0	20
C	0	40
D	0	60

Below figure illustrates the transmission scheduling (arrow-heads symbolize transmissions).

```
A -->------------------------------------------ ...

B ------------>------------------------------- ...

C ---------------------->---------------------- ...

D ------------------------------->------------ ...
```

(continues on next page)

[1] The single-shot delay starts from when the internal transmission scheduling starts shortly after power is applied.

(continued from previous page)

```
    |    +    |    +    |    +    |    +    |    +    |
t --+----+----+----+----+----+----+----+----+----+-- ...
    |    +    |    +    |    +    |    +    |    +    |
    0 ms      20 ms     40 ms     60 ms     80 ms
.
```

Note

Note how the transmissions do not coincidence.

Periodic

A transmit message is *periodic* when the *period* value is more than zero. In this case, the *delay* is the time from the start of each period to the transmission (the *offset* within the period). Consequently, the *delay* value **must be less than** the *period* value.

Example: Four periodic transmit messages are configured with *delay* values set to zero.

#	Period [ms]	Delay [ms]
A	10	0
B	20	0
C	30	0
D	40	0

Below figure illustrates the transmission scheduling (arrow-heads symbolize transmissions).

```
A -->---->---->---->---->---->---->---->---->-- ...

B -->--------->--------->--------->--------->-- ...

C -->-------------->-------------->------------ ...

D -->------------------->------------------->-- ...

    |    +    |    +    |    +    |    +    |    +    |
t --+----+----+----+----+----+----+----+----+----+-- ...
    |    +    |    +    |    +    |    +    |    +    |
    0 ms      20 ms     40 ms     60 ms     80 ms
.
```

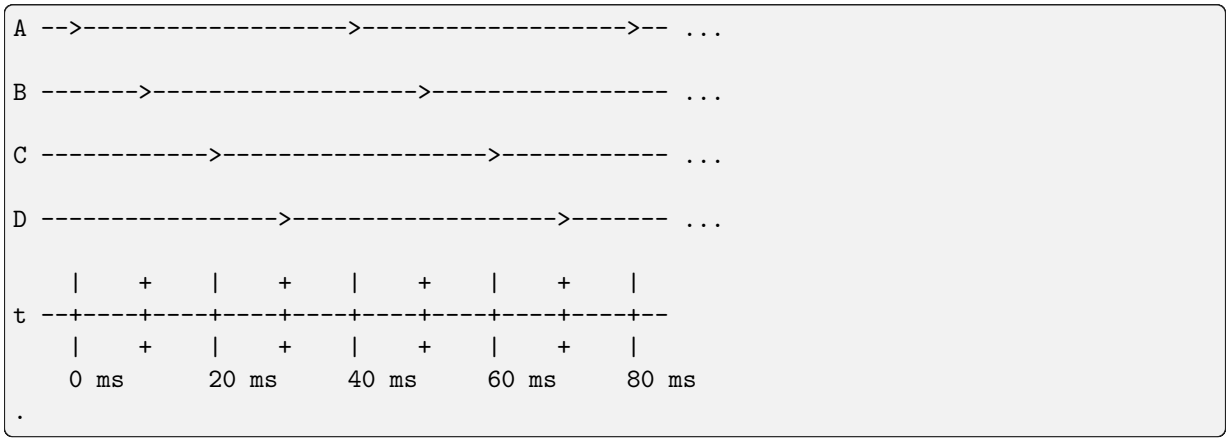
Note

Note how the transmissions periodically coincide in time. Avoid this by using *delay* values.

Example: Four periodic transmit messages are configured with the same *period* value and different *delay* values.

#	Period [ms]	Delay [ms]
A	40	0
B	40	10
C	40	20
D	40	30

Below figure illustrates the transmission scheduling (arrow-heads symbolize transmissions).



Note

Note how the transmissions do not coincidence.

0.4.5.5 Heartbeat

This page documents the *heartbeat* configuration

Table of Contents

Configuration file fields

This section is autogenerated from the Rule Schema file.

State `can.heartbeat.state`

Enable to periodically transmit heartbeat signal.

Type	Default	Options
integer	0	Disable: `0` Enable: `1`

ID Format `can.heartbeat.id_format`

ID format of heartbeat message.

Type	Default	Options
integer	1	Standard (11-bit): `0` Extended (29-bit): `1`

ID (hex) `can.heartbeat.id`

ID of heartbeat message in hex. Example: 1FF.

Type	Default
string	00435353

Configuration explained

This section contains additional information and examples.

Note

The heartbeat cannot be disabled using the control signal

Note

The heartbeat feature requires a CAN-bus physical mode supporting transmissions

Payload format

The device can transmit a 1 Hz periodic heartbeat signal. The signal payload contains logging state (enabled/disabled), the device time and space left on the SD-card in MB.

The interpretation of the 8-byte data payload of the heartbeat signal is given below:

Byte No.	0	1	2-5	6-7
Interpretation	Fixed 0xAA	State	Epoch time	Space left

- Byte 0 has the reserved value 0xAA
- The Epoch time is time-zone and offset adjusted
- Multi-byte fields should be interpreted MSB (Most-SignificantByte) first
- The `State` holds information on the current `rx_state` / `tx_state`:
  - 0: RX disabled, TX disabled
  - 1: RX enabled, TX disabled
  - 2: RX disabled, TX enabled
  - 3: RX enabled, TX enabled

Heartbeat with payload: `AA 03 5D 78 FB 8B 1D 93`

Byte No.	0	1	2-5	6-7
Interpretation	Fixed	State	Epoch time	Space left
Payload	0xAA	0x03	0x5D78FB8B	0x1D93

- Fixed: 0xAA

- State: RX and TX enabled
- Epoch time: $5D78FB8B_{16} = 1568209803_{10}$ -> 11/09/2019 13:50:03
- Space left: $1D93_{16} = 7571_{10}$ MB

Heartbeat with payload: `AA 00 5D 78 FB 8B 00 00`

Byte No.	0	1	2-5	6-7
Interpretation	Fixed	State	Epoch time	Space left
Payload	0xAA	0x00	0x5D78FB8B	0x0000

- Fixed: 0xAA
- State: RX and TX disabled
- Epoch time: $5D78FB8B_{16} = 1568209803_{10}$ -> 11/09/2019 13:50:03
- Space left: $0000_{16} = 0_{10}$ MB

0.4.5.6 Control

This page documents the *control* configuration

Table of Contents

Configuration file fields

This section is autogenerated from the Rule Schema file.

Can.control `can.control`

Control reception (rx) state `can.control.control_rx_state`

Control CAN-bus reception state (including logging)

Type	Default	Options
integer	0	Disable: `0` Enable: `1`

Control transmission (tx) state `can.control.control_tx_state`

Control CAN-bus transmission state (including logging)

Type	Default	Options
integer	0	Disable: `0` Enable: `1`

Start `can.control.start`

Message `can.control.start.message`

Channel `can.control.start.message.chn`

CAN-bus channel

Type	Default	Options
integer	0	CAN internal: `0` CAN 1: `1` CAN 2: `2`

ID format `can.control.start.message.id_format`

ID format of message.

Type	Default	Options
integer	0	Standard (11-bit): `0` Extended (29-bit): `1`

ID (hex) `can.control.start.message.id`

ID of message in hex. Example: 1FF.

Type	Default
string	0

ID mask (hex) `can.control.start.message.id_mask`

ID mask in hex. Example: 7FF.

Type	Default
string	7FF

Signal `can.control.start.signal`

Signal type `can.control.start.signal.type`

Type	Default	Options
integer	0	Unsigned: `0`

Signal byteorder `can.control.start.signal.byteorder`

Can be Motorola (big endian) or Intel (little endian)

Type	Default	Options
integer	1	Motorola: `0` Intel: `1`

Signal bit position `can.control.start.signal.bitpos`

Type	Default	Minimum	Maximum
integer	0	0	512

Signal bit length `can.control.start.signal.length`

Type	Default	Minimum	Maximum
integer	0	0	64

Signal scaling `can.control.start.signal.factor`

Type	Default
number	0

Signal offset `can.control.start.signal.offset`

Type	Default
number	0

Trigger high (dec) `can.control.start.trigger_high`

Type	Default
number	0

Trigger low (dec) `can.control.start.trigger_low`

Type	Default
number	0

Stop `can.control.stop`

Message `can.control.stop.message`

Channel `can.control.stop.message.chn`

CAN-bus channel

Type	Default	Options
integer	0	CAN internal: `0` CAN 1: `1` CAN 2: `2`

ID format `can.control.stop.message.id_format`

ID format of message.

Type	Default	Options
integer	0	Standard (11-bit): `0` Extended (29-bit): `1`

ID (hex) `can.control.stop.message.id`

ID of message in hex. Example: 1FF.

Type	Default
string	0

ID mask (hex) `can.control.stop.message.id_mask`

ID mask in hex. Example: 7FF.

Type	Default
string	7FF

Signal `can.control.stop.signal`

Signal type `can.control.stop.signal.type`

Type	Default	Options
integer	0	Unsigned: `0`

Signal byteorder `can.control.stop.signal.byteorder`

Can be Motorola (big endian) or Intel (little endian)

Type	Default	Options
integer	1	Motorola: `0` Intel: `1`

Signal bit position `can.control.stop.signal.bitpos`

Type	Default	Minimum	Maximum
integer	0	0	512

Signal bit length `can.control.stop.signal.length`

Type	Default	Minimum	Maximum
integer	0	0	64

Signal scaling `can.control.stop.signal.factor`

Type	Default
number	0

Signal offset `can.control.stop.signal.offset`

Type	Default
number	0

Trigger high (dec) `can.control.stop.trigger_high`

Type	Default
number	0

Trigger low (dec) `can.control.stop.trigger_low`

Type	Default
number	0

Configuration explained

This section contains additional information and examples.

The control signal can used to control the device message reception (effectively the logging) and / or the transmission (effectively the processing of the transmit list) for each CAN-bus channel. The control signal has a flexible configuration allowing for integration with many protocols. The control signal can e.g. be used to start / stop logging based on some application parameters, such as speed, RPM, geofence, time-of-day or discrete events.

The configuration of the signals uses a concept similar to that used by *.DBC* files. In case a *.DBC* file is available (describing the interpretation of the control message signals), the information from the file can be used directly for configuration. For more information see Section configuration/signal:Signal.

Control signal overview:

- A control signal can be configured for each CAN-bus channel
- A control signal can be based on messages from any channel
- One message ID is used for start and one for stop. These can be different or the same
- The message payload is decoded on the device, making it easy to set start / stop ranges

The start / stop ranges follow the following logic:

- If the start / stop ranges do not overlap, they are evaluated individually
- If the start range lies within the stop range, then start takes precedence (see examples below)
- If the stop range lies within the start range, then stop takes precedence (see examples below)

Note

File splitting is not affected by the control signal (i.e. the control signal does not force additional log file splits)

Note

The control signal can only be used if accepted by the CAN-bus filter

Note

The initial states of message reception and transmission are set in configuration section *General*.

Examples

Example: Start / stop ranges not overlapping.

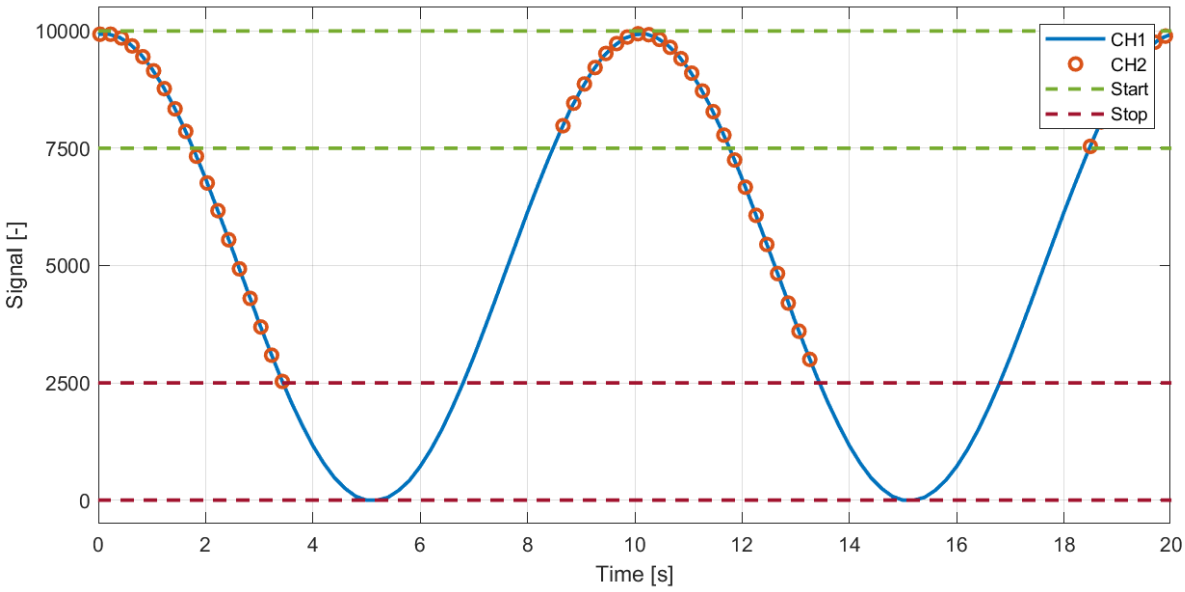
Can e.g. be used to start logging when speed signal exceeds some value and stop when it drops below some other value.

Start trigger:

- High: 10000
- Low: 7500

Stop trigger:

- High: 2500
- Low: 0



Example: Start / stop ranges not overlapping.

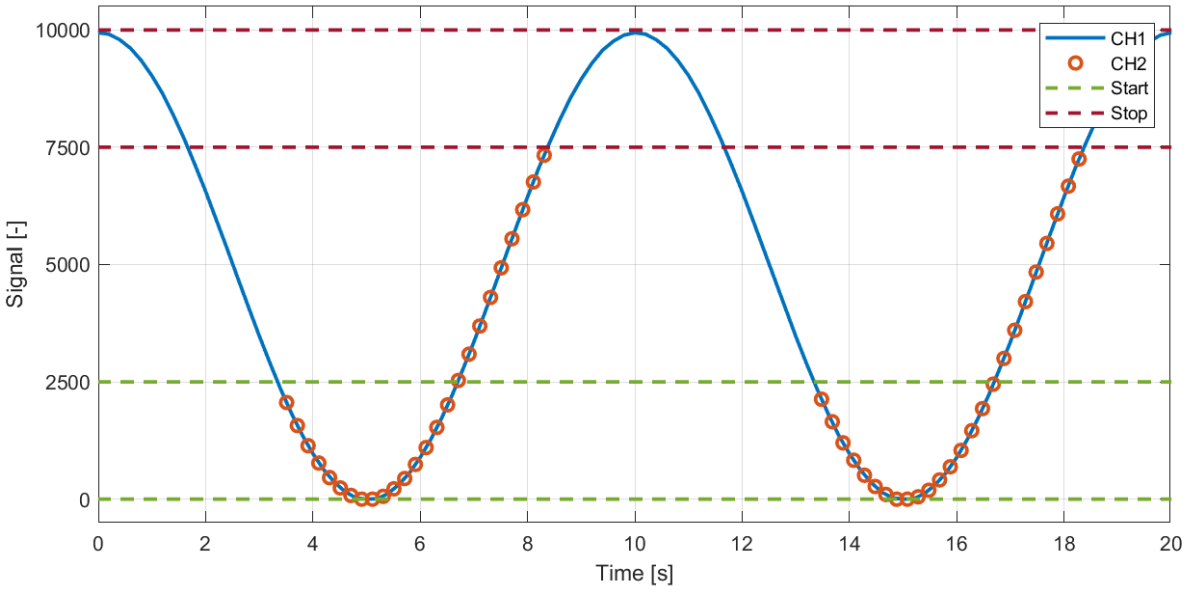
Can e.g. be used to start logging when pressure signal drops below some value and stop when it again raises above some other value.

Start trigger:

- High: 2500
- Low: 0

Stop trigger:

- High: 10000
- Low: 7500



Example: Start range lies within stop range, start takes precedence.

Can e.g. be used to start logging when a temperature signal lies within some range and stop when outside.

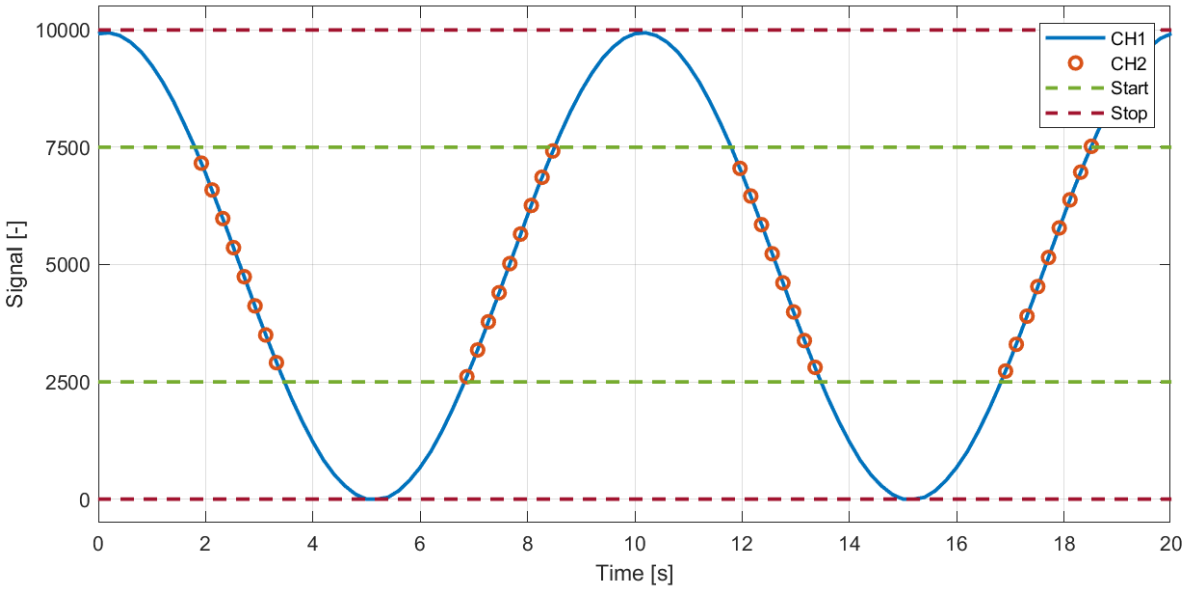
Start trigger:

- High: 7500

- Low: 2500

Stop trigger:

- High: 10000
- Low: 0



Example: Stop range lies within start range, stop takes precedence.

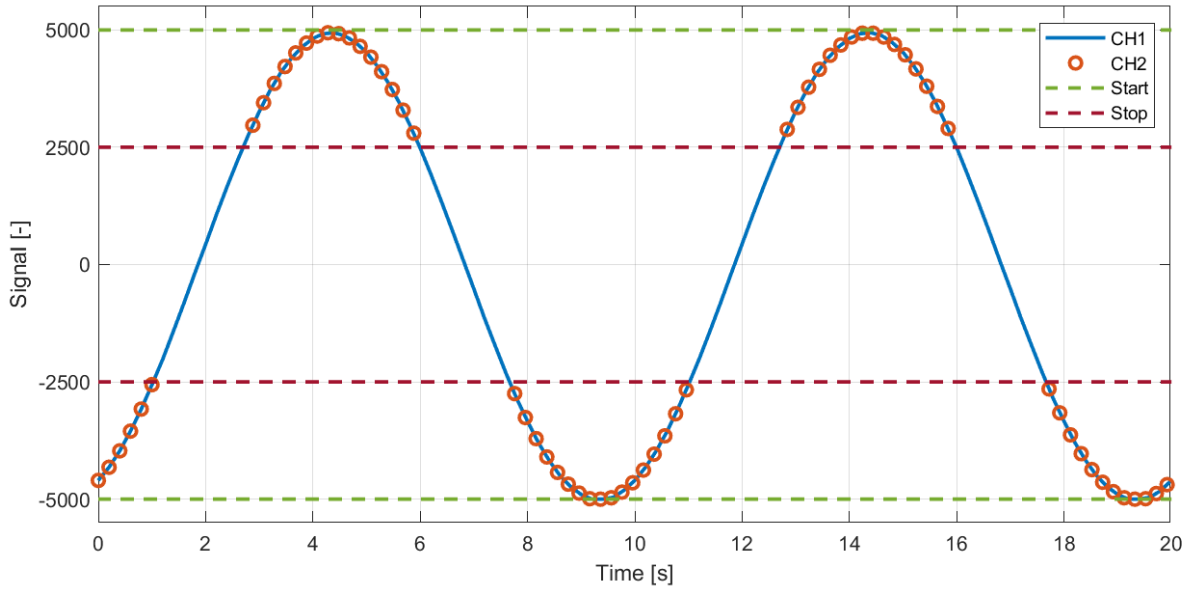
Can e.g. be used to start logging when the absolute value of an acceleration signal exceeds a certain value.

Start trigger:

- High: 5000
- Low: -5000

Stop trigger:

- High: 2500
- Low: -2500



0.4.6 LIN

The configurations of LIN Channel 1 and LIN Channel 2 are identical.

The LIN configuration is split into the following sections:

0.4.6.1 Physical

This page documents the *physical* configuration

Table of Contents

Configuration file fields

This section is autogenerated from the Rule Schema file.

Mode `lin.phy.properties.mode`

Device LIN bus mode.

Type	Default	Options
integer	0	Subscriber: `0` Publisher: `1`

Bit-rate `lin.phy.properties.bit_rate`

Type	Default	Options
integer	19200	2400: `2400` 9600: `9600` 10400: `10400` 19200: `19200`

Configuration explained

This section contains additional information and examples.

0.4.6.2 Frame Table

This page documents the *frame table* configuration

Table of Contents

Configuration file fields

This section is autogenerated from the Rule Schema file.

Name `lin.frames.items.name`

Optional frame name.

Type	Max length
string	16

Frame ID (hex) `lin.frames.items.id`

ID of frame in hex. Example: 0F.

Type	Max length
string	2

Frame Length (decimal) `lin.frames.items.length`

Length of the frame in decimal.

Type	Minimum	Maximum
integer	1	8

Checksum Type `lin.frames.items.checksum_type`

Type of the checksum used on the LIN frame.

Type	Default	Options
integer	0	Enhanced: `0` Classic: `1`

Configuration explained

This section contains additional information and examples.

The LIN controller expects default data lengths and checksums as explained in *LIN*. LIN-frames using a different configuration (length, checksum or both) can be explicitly configured using the *frame table*.

Note

LIN frames satisfying the default expected configuration do not need to be inserted in the *frame table*.

0.4.6.3 Transmit

This page documents the *transmit* configuration

Table of Contents

Configuration file fields

This section is autogenerated from the Rule Schema file.

Name `lin.transmit.items.name`

Optional transmit rule name.

Type	Max length
string	16

State `lin.transmit.items.state`

Disabled transmit rules are ignored.

Type	Default	Options
integer	1	Disable: `0` Enable: `1`

Frame ID (hex) `lin.transmit.items.id`

Type	Max length
string	2

Data (hex) `lin.transmit.items.data`

Type	Max length
string	16

Configuration explained

This section contains additional information and examples.

The interpretation of the *transmit list* depends on the configuration of *LIN bus mode*:

Publisher mode

The number of bytes entered in the `data` field determines the interpretation of the transmission frame:

Length of `data` is zero

The transmit is a *SUBSCRIBE* frame, meaning that a *Subscriber* on the bus is expected to provide the data payload (satisfying the *frame table*).

Length of `data` is above zero

The transmit is a *PUBLISH* frame, meaning that the CANedge provides the data payload.

In *Publisher* mode, the CANedge schedules the frame transmissions configured by the `period` and `delay`.

> **Warning**
>
> Be aware that transmit uses `period` and `delay` to schedule transmissions. This is a different concept than what is used by *LDF* files.

Subscriber mode

In *Subscriber* mode, the CANedge awaits a *SUBSCRIBE* frame with a matching ID from the bus *Publisher* node. The number of bytes provided shall satisfy the *frame table*.

Warning

If the transmit list contains multiple frames using the same ID, then only the first entry is used.

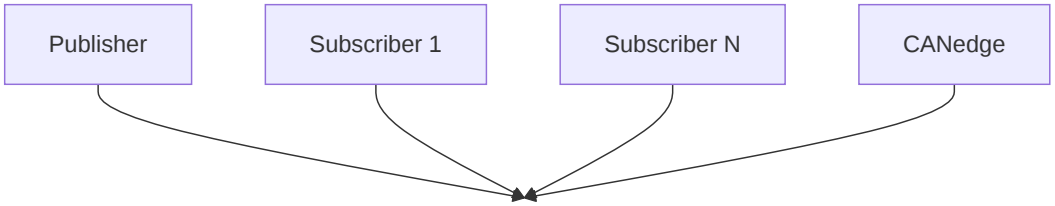
0.4.6.4 Topology

A LIN-bus consists of a *Publisher* node and one or more *Subscriber* nodes. The *Publisher* controls scheduling of messages on the LIN-bus, and the *Subscriber* nodes react to the emitted messages.

A message on the LIN-bus can either be a *PUBLISH* message, in which case *Publisher* node transmits both the message ID and data, or a *SUBSCRIBE* message, where the *Publisher* node only emits the message ID and one of the *Subscriber* nodes fill the data section of the message.

The configuration of the LIN network shall ensure that each message has one producer, such that each *PUBLISH* message is filled with data by the *Publisher*, while each *SUBSCRIBE* message has a node connected to the network which can provide the data for the message.

An example of the bus topology with the CANedge connected as a subscriber is illustrated below:



The CANedge is primarily intended to act as a *Subscriber* on the LIN-bus. In lieu of a *Publisher* node, the CANedge can be configured to emulate a simple *Publisher* node. In this case, the scheduling of messages on the network has to be done through the transmit configuration for the interface. Since only static data can be entered in the configuration, the simple *Publisher* node emulation cannot perform dynamic operations based on the LIN-bus activity.

0.4.6.5 Data length

Unless configured otherwise, the device assumes that the length of the LIN frame data payload is always defined by the message ID (bits 5 and 6 of the identifier), as defined in the table below:

Message ID	Data length
00-31 (0x00-0x1F)	2
32-47 (0x20-0x2F)	4
48-63 (0x30-0x3F)	8

This can be overridden in the configuration of the frame table.

0.4.6.6 Checksum

Supports LIN 1.3 *classic* checksum and LIN 2.0 *enhanced* checksum format. By default, all frames except ID 0x3C and 0x3D use enhanced checksum. This can be overridden on a frame by frame basis in the configuration of the frame table.

0.4.6.7 LIN Errors

The CANedge can detect and log errors on the LIN-bus if enabled in *Logging configuration*. The detected errors are categorized as follows:

- Checksum errors
- Receive errors
- Synchronization errors
- Transmission errors

The amount of associated data depends on the type of error. E.g. synchronization errors cannot contain information about the message ID, as it happens before that field is transmitted, and checksum information is not embedded in other cases than the checksum error case.

Checksum Errors

Checksum errors denotes that the node has calculated a different checksum than the one embedded in the LIN message on the bus. This can be an indicator of wrong configuration for the frame ID in the CANedge frame table.

Example: In case no information is known about the LIN bus in advance, the default frame table can be used with error logging enabled to help reverse engineer the actual frame table. Any message IDs deviating from the standard table (and present on the LIN-bus) will get a logged entry. These IDs can then be reconfigured in the CANedge frame table, in an attempt to find the correct settings.

Note that it can be necessary to change both message length and checksum model in order to get a valid configuration.

Receive Errors

Receive errors are logged when a fixed part of the LIN message is not as expected, or that the node detects a mismatch between the value being transmitted and the value sensed on the LIN-bus.

Synchronization Errors

Synchronization errors indicates an invalid synchronization field in the start of the LIN message, or that there is a too large deviation between the configured bitrate for the node and the detected bitrate from the synchronization field.

Transmission Errors

Transmission errors can only occur for IDs registered as *SUBSCRIBER* messages. If there is no node on the LIN-bus responding to a *SUBSCRIBER* message, a transmission error is logged.

The CANedge device uses a JSON file placed on the memory card for configuration.

The JSON format makes it easy to configure the device using custom tools, scripts, JSON editors or plain text editors. The configuration rules (min, max, ...) are defined using a JSON Schema, which is also stored on the memory card.

The Rule Schema serves as a guide for populating the Configuration File - and for automatically validating a Configuration File. Both the Configuration File and Rule Schema are automatically generated by the device if either is not found on the memory card. .. note:: The default configuration can be restored by deleting the existing Configuration File from the memory card and powering the device

Note

JSON files and JSON Schema rules are supported by most programming/scripting languages, making it easy to automate generation/validation of the device configuration in custom tools

Naming

The config and schema are placed in the root of the memory card and named as follows:

- Configuration File: `config-[FIRMWARE_MAJOR].[FIRMWARE_MINOR].json`
- Rule Schema: `schema-[FIRMWARE_MAJOR].[FIRMWARE_MINOR].json`

With `[FIRMWARE_MAJOR]` and `[FIRMWARE_MINOR]` taken from the device firmware version.

The firmware patch number is not included in the file naming as patches are guaranteed not to change the structure of the device configuration. For more information on the firmware versioning system, refer to the *Firmware* section.

Example: If the firmware version is 01.02.03, then the config and schema files are named `config-01.02.json` and `schema-01.02.json`, respectively.

0.5 Filesystem

0.5.1 Device file

A Device File (`device.json`) is located in the root of the SD-card with info on the device. The content of the Device File is updated when the device powers on.


```
{
  "id": "4F07A3C3",
  "type": "0000001D",
  "kpub": "l27UKi4ehjpxxEdmRstBk5UaqSGQYnfylzUNs9EOoJfDodvr/
↪PqNnMrz61IxzrBfFTmuhw2K2cJ4q60iFiYM8w==",
  "fw_ver": "01.01.02",
  "hw_ver": "00.03",
  "cfg_ver": "01.01",
  "cfg_name": "config-01.01.json",
  "cfg_crc32": "9ECC0C10",
  "sch_name": "schema-01.01.json",
  "log_meta": "Truck1",
  "space_used_mb": "36/7572",
  "sd_info": "000353445341303847801349A26A0153",
  "sd_used_lifespan": "2",
  "reset_cause": ""
}
```


Additional content may be added to the `device.json` in future firmware updates.

0.5.1.1 Fields explained

Base

- `id`: Device unique ID number
- `type`: Device type (CANedge1 = 0000001D)
- `kpub`: Device public key in Base-64 format
- `fw_ver`: Firmware version
- `hw_ver`: Hardware version
- `cfg_ver`: Configuration File version
- `cfg_name`: Configuration File name
- `cfg_crc32`: Configuration File checksum
- `sch_name`: Configuration Rule Schema name
- `log_meta`: Configurable device string (e.g. application name)
- `space_used_mb`: The SD-card used space of the total in MB (`[used]/[total]`)
- `sd_info`: Information about the SD card, including unique serial number in hex
- `sd_used_lifespan`: The SD-card self-reported health in percent of lifetime used, or `?` if unavailable
- `reset_cause`: For debugging purposes

0.5.2 Log file

This page documents the log files stored on the device SD-card.

Table of Contents

0.5.2.1 Format

The CANedge logs data in the industry standard MDF4 format, standardized by *ASAM*. MDF4 is a binary format which allows compact storage of huge amounts of measurement data. It is specifically designed for bus frame logging across e.g. CAN-bus, LIN-bus and Ethernet. MDF4 is widely adopted by the industry and supported by many existing tools.

Specifically, the CANedge uses MDF version 4.11 (file extension: `*.MF4`).

Timestamps

Each record is timestamped with 50 us resolution.

Finalization & sorting

The CANedge stores log files as *unfinalized* and *unsorted* to enable power safety. Finalization[2] and sorting[3] can be done as a post-processing step to speed up work with the files.

Note

It may be necessary to finalize/sort a log file before it is loaded into some MDF tools

Additional metadata about the device is captured in the files, including many of the fields exposed in the device file.

- `serial number`: Device unique ID number
- `device type`: Device type (CANedge1 = 0000001D)
- `firmware version`: Firmware version
- `hardware version`: Hardware version
- `config crc32 checksum`: Configuration File checksum
- `storage total`: The SD-card total space in kB
- `storage free`: The SD-card free space in kB
- `storage id`: The SD-card identifier
- `session`: File session counter
- `split`: File split counter
- `comment`: Configurable device string (e.g. application name)

[2] The MDF file header includes information on how to finalize the MDF file before use
[3] Sorting refers to an organization of the log records which enable fast indexing. It is not related to sorting of timestamps.

0.5.2.2 Naming

Log files are organized by the following path structure:

`LOG/[DEVICE_ID]/[SESSION_COUNTER]/[SPLIT_COUNTER].[FILE_EXTENSION]`

The path is constructed from the following parts:

- `LOG`: Static directory name used to store log files
- `DEVICE_ID`: Globally unique device ID
- `SESSION_COUNTER`: Increased by one for each power cycle[1]
- `SPLIT_COUNTER`: Resets to 1 on each power cycle and increased by one for each file split
- `FILE_EXTENSION`: The file extension selected in the configuration (`MF4 | MFC | MFE | MFM`)

For details on log file splits and related limits, see the *Logging Configuration* section.

File extension

The default extension is `MF4`. With compression/encryption enabled the extension changes:

Compression enabled	Encryption enabled	File extension
		.MF4
X		.MFC
	X	.MFE
X	X	.MFM

With both compression and encryption enabled, the data is first compressed, then encrypted.

For details on compression and encryption, see the *Logging Configuration* section.

Path example

Example: Log file path: `LOG/3B912722/00000004/00000189.MF4`

- `LOG`: The static directory common for all log files
- `3B912722`: The unique ID of the device which generated the log file
- `00000004`: Generated during the 4th session / power cycle
- `00000189`: Is log file number 189 of the session
- `MF4`: File type

0.5.2.3 Generic header

While plain MDF files are saved as MF4, encryption and/or compression uses a custom header to identify and store relevant information for the files. All file headers consist of a generic 20 byte header, followed by any specialized fields.

The generic header starts with an identifying sequence of the ASCII code for `Generic File`[4]. Following are information of the header version (`V Ge`, currently 0x01), file type version (`V FT`), file type (`FT`) and file sub-type (`FTI`). Finally, the device ID is stored. All numbers stored in the generic header are unsigned and big endian formatted.

```
| <-                    8 bytes                        -> |
|  Byte | Byte | Byte | Byte | Byte | Byte | Byte | Byte  |
|   'G'    'e'    'n'    'e'    'r'    'i'    'c'   ' '-> |
```

(continues on next page)

[1] The session counter is also increased by one if the counter of splits in one session exceeds 256

[4] `Generic File` maps to 12 bytes of ASCII, with no zero termination of the string.

(continued from previous page)

```
| <-'F'    'i'    'l'    'e' | V Ge | V FT |  FT  |  FTI  |
|    Device ID (Uint32, BE)  |
```

If required, a generic file may contain a footer as well, as specified by the format.

Encrypted files

Encrypted files have a file type of 0x11. The device supports AES encryption in Galois Counter Mode (GCM), with a file sub-type of 0x01. The current version of the format is 0x00. The encrypted file header stores three additional fields:

- The 12 bytes long initialization vector
- The number of hashing iterations for the key, stored as a 32 bit unsigned number in big endian format
- 16 bytes of salt data for the hashing of the key

```
| <-                    8 bytes                        -> |
| Byte | Byte | Byte | Byte | Byte | Byte | Byte | Byte |
|                       IV/Nonce                        -> |
| <-       IV/Nonce          |  Iterations (Uint32, BE)  |
|                          Salt                        -> |
| <-                       Salt                           |
```

The encrypted file contains an additional footer. This stores the 16 byte tag generated when AES runs in GCM mode. When decrypting, this tag should be checked to ensure the validity of the decrypted data. There is no alignment requirement for the footer.

```
| <-                    8 bytes                        -> |
| Byte | Byte | Byte | Byte | Byte | Byte | Byte | Byte |
|                         GCM Tag                       -> |
| <-                      GCM Tag                          |
```

Compressed files

Compressed files have a file type of 0x22. At present, the only supported compression format is heatshrink based. This is denoted by a file sub-type of 0x01. The current version of the format is 0x01. The additional header data are two unsigned 32 bit numbers: Lookahead and window sizes.

```
| <-                    8 bytes                        -> |
| Byte | Byte | Byte | Byte | Byte | Byte | Byte | Byte |
|   Lookahead (Uint32, BE)  |    Window (Uint32, BE)    |
```

Following the header is the compressed data stream. Following the data stream is a footer with a checksum over the compressed data. There is no alignment requirement for the footer. The checksum format is often found online as CRC32 JAM or JAMCRC.

```
| <-      4 bytes        -> |
| Byte | Byte | Byte | Byte |
|     CRC32 (Uint32, BE)    |
```

Encrypted and compressed files

If the file is both encrypted and compressed, it has been processed in two steps/streams. First the data is piped through a compression step, next it is piped through an encryption step. Each step can have its own version.

The SD-card filesystem is organized as illustrated by below example[1]:

```
/
├─ config-XX.XX.json
├─ uischema-XX.XX.json
├─ schema-XX.XX.json
├─ device.json
├─ meta
│  └─ ...
└─ LOG
   └─ [DEVICE_ID]
      ├─ 00000001
      │  ├─ 00000001.MF4
      │  ├─ 00000002.MF4
      │  └─ ...
      ├─ 00000002
      │  ├─ 00000001.MF4
      │  ├─ 00000002.MF4
      │  └─ ...
      └─ ...
```

- `config-XX.XX.json`: *Configuration file* (device configuration)
- `schema-XX.XX.json`: *Rule Schema file* (configuration rules)
- `uischema-XX.XX.json`: UI Schema file (configuration presentation)
- `device.json`: *Device file* (device information)
- `LOG/`: Directory containing log files (see *Naming* for more information)
- `meta/`: Temporary folder for setting the internally stored *session counter* (see *Setting session counter* for more information)

Note

Default *Configuration*, *Schema*, *UISchema*, and *Device* files are automatically re-created if deleted by the user.

Note

The device will store the information in the *meta* folder internally and delete the folder if present during startup

0.5.3 Replacing SD-card

The SD-card is **not** *locked* to the device. If the card is replaced (see *SD-card hardware requirements*), be aware of the following points:

- If the card is replaced by a card from another CANedge, it is recommended to clear the card
- The configuration file can optionally be copied to the new card (else a default is automatically created)

[1] `XX.XX` is replaced by the firmware *MAJOR* and *MINOR* version numbers

0.5.4 Setting session counter

Warning

Manually setting the session counter is usually only relevant when the internal battery has been replaced.

To manually set the session counter, create the `meta` folder in the root of the SD-card. Inside the folder, create a file called `meta_log.json` with the following template:

```
{
   "session": 123
}
```

Replace 123 with the desired next session counter value.

0.6 Internal signals

This page documents the signals internally generated by the CANedge.

The signals are available through the *internal* CAN-bus channel. The signal messages can be filtered, scaled, etc. as with the physical CAN-bus channels. See *CAN* for more information on CAN-bus channel configuration.

The CAN-internal database file (.DBC) can be downloaded from the online documentation.

> **Note**
>
> Multiple variants of the CANedge share the same signal database. Not all signals are available for all variants.

The remaining of this section is autogenerated from the database (DBC) file.

0.6.1 Messages

Message	Format	ID (DEC)	ID (HEX)	Bytes	Description
TimeExternal	Standard	5	0x005	8	Time received, event
GnssStatus	Standard	101	0x065	1	GNSS status, 5 Hz
GnssTime	Standard	102	0x066	6	GNSS time, 5 Hz
GnssPos	Standard	103	0x067	8	GNSS position, 5 Hz
GnssAltitude	Standard	104	0x068	4	GNSS altitude, 5 Hz
GnssAttitude	Standard	105	0x069	8	GNSS attitude, 5 Hz
GnssDistance	Standard	106	0x06A	3	GNSS distance, 1 Hz
GnssSpeed	Standard	107	0x06B	5	GNSS speed, 5 Hz
GnssGeofence	Standard	108	0x06C	2	GNSS geofence(s), 1 Hz
ImuAlign	Standard	110	0x06E	7	IMU alignment, 1 Hz
ImuData	Standard	111	0x06F	8	IMU data, 5 Hz

0.6.2 Signals

0.6.2.1 TimeExternal signals

Signal	Start	Length	Factor	Offset	Unit	Description
InternalEpoch	0	32	1	15778404	*s*	Internal epoch time
ExternalEpoch	32	32	1	15778404	*s*	External epoch time

0.6.2.2 GnssStatus signals

Signal	Start	Length	Factor	Offset	Unit	Description
FixType	0	3	1	0		Fix type
Satellites	3	5	1	0		Number of satellites used

FixType values

Value	Description
0	No fix
1	Dead reckoning only
2	2D-fix
3	3D-fix
4	GNSS + dead reckoning combined
5	Time only fix

0.6.2.3 GnssTime signals

Signal	Start	Length	Factor	Offset	Unit	Description
TimeValid	0	1	1	0		Time validity
TimeConfirmed	1	1	1	0		Time confirmed
Epoch	8	40	0.001	15778404	*s*	Epoch time

TimeValid values

Value	Description
0	Invalid
1	Valid

TimeConfirmed values

Value	Description
0	Unconfirmed
1	Confirmed

0.6.2.4 GnssPos signals

Signal	Start	Length	Factor	Offset	Unit	Description
PositionValid	0	1	1	0		Position validity
Latitude	1	28	1e-06	-90	*deg*	Latitude
Longitude	29	29	1e-06	-180	*deg*	Longitude
PositionAccuracy	58	6	1	0	*m*	Position accuracy

PositionValid values

Value	Description
0	Invalid
1	Valid

0.6.2.5 GnssAltitude signals

Signal	Start	Length	Factor	Offset	Unit	Description
AltitudeValid	0	1	1	0		Altitude validity
Altitude	1	18	0.1	-6000	*m*	Altitude
AltitudeAccuracy	19	13	1	0	*m*	Altitude accuracy

AltitudeValid values

Value	Description
0	Invalid
1	Valid

0.6.2.6 GnssAttitude signals

Signal	Start	Length	Factor	Offset	Unit	Description
AttitudeValid	0	1	1	0		Attitude validity
Roll	1	12	0.1	-180	*deg*	Vehicle roll
RollAccuracy	13	9	0.1	0	*deg*	Vehicle roll accuracy
Pitch	22	12	0.1	-90	*deg*	Vehicle pitch
PitchAccuracy	34	9	0.1	0	*deg*	Vehicle pitch accuracy
Heading	43	12	0.1	0	*deg*	Vehicle heading
HeadingAccuracy	55	9	0.1	0	*deg*	Vehicle heading accuracy

AttitudeValid values

Value	Description
0	Invalid
1	Valid

0.6.2.7 GnssDistance signals

Signal	Start	Length	Factor	Offset	Unit	Description
DistanceValid	0	1	1	0		Distance valid
DistanceTrip	1	23	1	0	*m*	Distance traveled since last reset

DistanceValid values

Value	Description
0	Invalid
1	Valid

0.6.2.8 GnssSpeed signals

Signal	Start	Length	Factor	Offset	Unit	Description
SpeedValid	0	1	1	0		Speed valid
Speed	1	20	0.001	0	m/s	Speed m/s
SpeedAccuracy	21	19	0.001	0	m/s	Speed accuracy

SpeedValid values

Value	Description
0	Invalid
1	Valid

0.6.2.9 GnssGeofence signals

Signal	Start	Length	Factor	Offset	Unit	Description
FenceValid	0	1	1	0		Geofencing status
FenceCombined	1	2	1	0		Combined (logical OR) state of all geofences
Fence1	8	2	1	0		Geofence 1 state
Fence2	10	2	1	0		Geofence 2 state
Fence3	12	2	1	0		Geofence 3 state
Fence4	14	2	1	0		Geofence 4 state

FenceValid values

Value	Description
0	Invalid
1	Valid

FenceCombined values

Value	Description
0	Unknown
1	Inside
2	Outside

Fence1 values

Value	Description
0	Unknown
1	Inside
2	Outside

Fence2 values

Value	Description
0	Unknown
1	Inside
2	Outside

Fence3 values

Value	Description
0	Unknown
1	Inside
2	Outside

Fence4 values

Value	Description
0	Unknown
1	Inside
2	Outside

0.6.2.10 ImuAlign signals

Signal	Start	Length	Factor	Offset	Unit	Description
AlignStatus	0	3	1	0		IMU-mount alignment status
AlignXYError	3	1	1	0		IMU-mount X or Y alignment error
AlignZError	4	1	1	0		IMU-mount Z alignment error
AlignError	5	1	1	0		IMU-mount singularity error
AlignZ	8	16	0.01	0	*deg*	IMU-mount Z angle
AlignY	24	16	0.01	-90	*deg*	IMU-mount Y angle
AlignX	40	16	0.01	-180	*deg*	IMU-mount X angle

AlignStatus values

Value	Description
0	Idle
1	Ongoing
2	Coarse
3	Fine

AlignXYError values

Value	Description
0	No error
1	Error

AlignZError values

Value	Description
0	No error
1	Error

AlignError values

Value	Description
0	No error
1	Error

0.6.2.11 ImuData signals

Signal	Start	Length	Factor	Offset	Unit	Description
ImuValid	0	1	1	0		IMU status
AccelerationX	1	10	0.125	-64	m/s^2	IMU X acceleration with a resolution of 0.125 m/s^2
AccelerationY	11	10	0.125	-64	m/s^2	IMU Y acceleration with a resolution of 0.125 m/s^2
AccelerationZ	21	10	0.125	-64	m/s^2	IMU Z acceleration with a resolution of 0.125 m/s^2
AngularRateX	31	11	0.25	-256	deg/s	IMU X angular rate with a resolution of 0.25 deg/s
AngularRateY	42	11	0.25	-256	deg/s	IMU Y angular rate with a resolution of 0.25 deg/s
AngularRateZ	53	11	0.25	-256	deg/s	IMU Z angular rate with a resolution of 0.25 deg/s

ImuValid values

Value	Description
0	Invalid
1	Valid

0.7 Firmware

0.7.1 Download Firmware Files

See the online documentation for the latest Firmware Files and changelog.

Firmware Files can be downloaded from the online documentation.

This page describes how to upgrade the device firmware.

Table of Contents

0.7.2 Firmware versioning & naming

The device firmware versioning is inspired by the semantic versioning system.

Each firmware is assigned three two digit numbers: `MAJOR`, `MINOR`, `PATCH`:

- `MAJOR`: Incompatible changes (e.g. requires major changes to the Configuration File)
- `MINOR`: New backwards-compatible functionality (e.g. new fields in the Configuration File)
- `PATCH`: Backwards-compatible bug fixes (e.g. no changes to the Configuration File)

The firmware files available for download are zipped with naming as follows:

`firmware-[MAJOR].[MINOR].[PATCH].zip`

Example:

`firmware-01.02.03.zip`

0.7.3 Firmware Update

The device supports in-the-field firmware updates.

Note

The firmware update process is power safe (tolerates power failures). However, it is recommended to ensure that the process completes

0.7.3.1 Update process

The firmware update process begins when the device is powered and has been prepared with a new Firmware File:

1. Power is applied to device
2. The green LED comes on (can take a few seconds)
3. If the firmware is valid, the green LED blinks 5 times, else the red LED blinks 5 times
4. The green LED remains solid while the firmware is updated (~30 sec)
5. If the update is successful, the green LED blinks 5 times, else the red LED blinks 5 times

6. The updated firmware is started and the device is ready for logging
7. If any external modules need to be updated, then these updates are applied now (see *Update of external modules*)

Note

The green LED comes on later than usual when a firmware update is initiated

Note

The device automatically removes any Firmware Files when the update has completed. Firmware Files should never be manually deleted during the update process.

Update of external modules

External modules are updated while the device is (partly) operational. **Updating external modules can take from a few minutes and up to 1 hour**. If power is lost during update of external modules, the update resumes next time the device powers on.

0.7.3.2 Configuration update

If a device is updated to a firmware version with a different `MAJOR` or `MINOR` number, then the Configuration File also needs to be updated (i.e. with an updated name and structure matching the new firmware). The Configuration File is named as described in the *Configuration* section. A default Configuration File and corresponding Rule Schema are contained in the firmware-package (zip).

To modify an existing Configuration File, it can be useful to load the new Rule Schema in an editor together with the old Configuration File. After making the necessary updates, save the modified Configuration File with a name matching the new version.

Note

The firmware can be updated without providing a new compatible Configuration File. In this case, the device creates a default Configuration File on the SD-card

0.7.3.3 Update from SD-card

The firmware can be updated by placing a Firmware File on the SD-card and powering the device:

1. Download the firmware zip (Firmware File + Configuration File + Rule Schema)
2. Place the `firmware.bin` file on the SD-card (root directory)
3. If `MAJOR`/`MINOR` is different, update the Configuration File and place it on the SD-card
4. Power on the device and wait for the update process to complete

Note

An incompatible firmware image is deleted and does not break the device

Example: Current firmware: `01.01.01`, new firmware: `01.01.02`

1. Download `firmware-01.01.02.zip` and unzip it
2. Copy `firmware.bin` to the SD-card
3. The `MAJOR` and `MINOR` versions are unchanged (no need to update the Configuration File)

4. Power on the device and wait for the update process to complete

Example: Current firmware: `01.01.01`, new firmware: `01.02.01`

1. Download `firmware-01.02.01.zip` and unzip it
2. Copy `firmware.bin` to the SD-card
3. Update the Configuration File (or use the default created by the firmware update)
4. Power on the device and wait for the update process to complete

0.8 Legal information

0.8.1 Usage warning

Warning

Carefully review the below usage warning before installing the product

The use of the CANedge device must be done with caution and an understanding of the risks involved. The operation of the device may be dangerous as you may affect the operation and behavior of a data-bus system.

Improper installation or usage of the device can lead to serious malfunction, loss of data, equipment damage and physical injury. This is particularly relevant when the device is physically connected to an application that may be controlled via a data-bus. In this setup you can potentially cause an operational change in the system, turn on/off certain modules and functions or change to an unintended mode.

The device should only be used by persons who are qualified/trained, understand the risks and understand how the device interacts with the system in which it is integrated.

0.8.2 Terms & conditions

Please refer to our general terms & conditions.

0.8.3 Electromagnetic compatibility

The CANedge has been tested in accordance with CE, FCC and IC standards.

Certificates are available in the online documentation.

0.8.4 Voltage transient tests

The CANedge has passed below ISO 7637-2:2011 tests, performed by TÜV SÜD[1]:

```
ISO 7637-2:2011: Voltage transient emissions test on supply lines
ISO 7637-2:2011: Transient immunity test on supply lines
```

0.8.5 Contact details

For any questions regarding our products, please contact us:

CSS Electronics
EU VAT ID: DK36711949
Soeren Frichs Vej 38K (Office 35), 8230 Aabyhoej, Denmark
contact[AT]csselectronics.com
+45 91252563
www.csselectronics.com

The CANedge1 enables stand-alone logging of data from CAN- and LIN-bus to an SD-card.

The device offers a range of configuration options incl. message filtering, pre-scaling, transmit messages, cyclic logging, compression, encryption and more.

The CANedge is based on open standards: No proprietary tools required. No subscription models. No vendor lock-in. Users can leverage the CANedge tools - or integrate with custom applications.

[1] Test performed using the hardware version $\leq$ 00.02 enclosure

Fig. 5: Hardware version 00.03