CANopen® PRESSURE & TEMPERATURE SENSOR

Valid from firmware version v1826

TABLE OF CONTENTS:

TYPE: SMC

TECHNICAL DESCRIPTION

ADZ NAGANO GmbH
Gesellschaft für Sensortechnik

Bergener Ring 43
D-01458 Ottendorf-Okrilla

Telefon: +49 (0)35205-59 69 30
Telefax: +49 (0)35205-59 69 59

info@adz.de
www.adz.de

1 INTRODUCTION

1.1 Abbreviation index

The following table shows the used abbreviations within this document.

LSB	Least-significant bit/byte
MSB	Most-significant bit/byte
SDO	Service Data Object
PDO	Process Data Object
TPDO	Transmit PDO
NMT	Network Management Service
EMCY	Emergency Object
SYNC	Synchronization Object
LSS	Layer Setting Services
EDS	Electronic Datasheet
COB-ID	Communication Object Identifier

1.2 Bibliographical references

The following table contains all used references within this document.

[1] CAN in Automation, “DS-404 v1.2.0 Device profile for measuring devices and closed-loop controllers,” 2002.

[2] ADZ NAGANO GmbH, “SMC Series - ADZ NAGANO - Sensortechnik,” [Online]. Available: http://www.adz.de/smc.html. [Accessed 2017].

[3] CAN in Automation, “CiA 305: Layer setting services (LSS) and protocols,” 2013.

[4] CAN in Automation, “CANopen vendor-ID,” [Online]. Available: https://www.can-cia.org/services/canopen-vendor-id/. [Accessed 2 August 2017].

[5] CAN in Automation, “DS-301 v4.2.0 CANopen application layer and communication profile,” 2011.

[6] CAN in Automation, “DRP 303-2 v1.0, Representation of SI Units and Prefixes,” July 1999.

[7] H. Schmidt, “IEEE-754 Floating Point Converter,” [Online]. Available: https://www.h-schmidt.net/FloatConverter/IEEE754.html. [Accessed October 2017].

1.3 Preface

Please read this document conscientiously before using the *SMC pressure/temperature transducers*. At any time, keep this document at a place which all users have access to.

Please support us to improve this document and the *SMC pressure/temperature transducer* product.

Chapter 8 contains **useful application examples** helping to perform the first steps with the *SMC transducers*.

This documentation **is valid** from firmware version **v1825**

2 FEATURES OF THE SMC TRANSDUCER

2.1 General features

SMC transducers are intended to be used for pressure and/or temperature acquisition of liquid and gaseous media.

The pressure and temperature measurements are being digitalized, linearized and made available via the *CANopen* interface. Besides the *CANopen* standard feature set (CiA 301/404), *SMC transducers* offer many useful functions – amongst them are, for example:

- Full support of **CAN 2.0 A** and **CAN 2.0 B** identifiers,
- Two separate **measuring channels** for *SMC pressure transducers*: The *primary measuring channel* carries pressure measurements, the *secondary measuring channel* offers coarse temperature values (± 5K). *SMC temperature transducers* feature only one measuring channel.
- Almost all internal calculations are being performed using a *hardware floating-point unit*. Thus, calculating is performed at **highest precision and speed**. Impulse response goes down to $\approx 2ms$.
- **Stay-set indicators** (slave pointers) for keeping track of the minimum and maximum measurements applied to the sensor. Each measuring channel features its own set of stay-set indicators,
- Flexible **5-stage IIR filters** (one for each measuring channel) which fulfil almost every situation's needs. Their coefficients are typically configured as "floating average filter",
- A **Clamping unit** allowing to apply a limited range to all measurements which cannot be exceeded,
- Various interrupt sources for **event driven measurement** transmission,
- Possibility to replace the *Linear Scaling Unit* (CiA 404, [1]) with one of **numerous transfer functions**: Customers can, for example, order transducers that perform *deviations, integrations, polynomial functions* or *spheric/cylindric fill grade approximations*.

The nominal operating temperature ranges from **-40°C to 125°C**; a summary of all (electrical and non-electrical) limitations is located at section 7.1.

2.2 Optional hardware features

The following hardware features can optionally be included in the *SMC pressure/temperature transducers*:

- There are various options for **included CAN bus termination**:
  - Pure *resistive termination* (120Ω),
  - *Split termination* ($2x\ 60\Omega + 4.7nF$; should be preferred over the resistive termination),
  - *Switchable termination*. The termination state (enabled/disabled) can be configured using SDO; the operating temperature range will be limited to -40°C … 105°C.
- **Galvanic isolation** of the bus lines. This feature introduces the additional wire CANGND/GND_{CAN} which otherwise is not present. The operating temperature range will be limited to -40°C … 100°C.

3 THE CAN BUS

3.1 Transmission speed / bit rate

The bit rate represents the bus communication speed; it must be the same for all participants. There are physical and conceptional limits concerning the bit rate:

- The longer the bus wiring, the smaller the bit rate that can be applied. By rule of thumb, the wiring length should not exceed *25m at* $1\ Mbit/s$ and *500m at* $125\ kbit/s$,
- The faster the communication speed, the higher the "EMC pollution". That's why the bit rate should be chosen only as high as necessary. A good trade off will be made with a bit rate of $125\ kbit/s$,
- Due to timing reasons, the *SMC pressure/temperature transducers* do only support bit rate values within the following range: $[20\ kbit/s\ \ldots 1.5\ Mbit/s]$.

CANopen defines a set of *standard bit rates* of which the majority is supported by the *SMC transducers*. Refer to the bit rate table located at section 4.13. Besides these pre-defined bit rates, it is possible to define *customer specific bit rates* when placing the order.

For changing the bit rate, the *SMC transducers* implements the LSS service. While the *standard bit rates* are accessible at table selector 0, *customer defined* bit rates will be made available at the table selector 1. If desired, please explicitly point out to define a *customer defined* bit rate table (selector 1) when placing the order!

The *SMC pressure/temperature* transducers' standard bit rate is $125\ kBit/s$.

3.2 CAN frames

CAN frames represent the messages sent over a CAN bus. A CAN frame consists of multiple data fields; some of the most important fields are:

- *Identifier*, amongst other things used for expressing the priority of a CAN frame. According to CAN 2.0 A, the identifier has a length of 11 bits; CAN 2.0 B raised the length to 29 bits,
- *Data length code (DLC)* holds the amount of data bytes contained in the CAN frame,
- The *data* fields contain the actual frame data. There can be between 0 and 8 data bytes.

SMC pressure/temperature transducers support both **CAN 2.0 A** and **CAN 2.0 B** frames.

4 CANOPEN

4.1 Node-ID

The Node-ID identifies a CANopen device within an application network and therefore must be unique. Valid Node-IDs are of data type *uint8* and range from 1 to 127 (0x01 … 0x7F). *SMC pressure/temperature transducers* do support Node-ID configuration via SDO and LSS.

The standard Node-ID for *SMC transducers* is 32 (0x20).

4.2 COB-ID

There are many *communication objects* within a CANopen network (the word '*Communication Object*' can be substituted with '*CAN message'*). Those are, for example, objects for SDO, PDO, SYNC, EMCY and NMT. Each of these *communication objects* relates to a certain CAN Identifier which (accumulated with some additional control bits) results in the so-called COB-ID ("C*ommunication Object Identifier*").

All COB-IDs used and provided by the *SMC pressure/temperature transducers* can be modified via SDO.

4.3 Network Management Service (NMT) and boot-up behaviour

Each CANopen device implements the *NMT state machine*. Most of the network members act as a *NMT slave*; their state machine will most likely be controlled by the network *NMT master*. The *NMT state machine* provides the following states:

- Initialization,
- Pre-Operational,
- Operational,
- Stopped.

When booting-up, a *SMC pressure/temperature transducer* sends out a *boot-up message* and usually settles its *state machine* in '*Pre-Operational*' state. The device is now performing measurements, but is not able to automatically output them using the PDO service.
In order to enable PDO, the *NMT state machine* must be transitioned to '*Operational*' state. This can be either done by transmitting a *Start-Remote-Node* message or by configuring the device to directly settle in '*Operational*' state (this behaviour is referred to as *Autostart*; see section 6.1.20).

The following CAN message represents a *NMT Start-Remote-Node* message which transitions all present *NMT slaves* to '*Operational*' state in order to enable measurements output (PDO):

CAN-ID (hex)	DLC	Data bytes (hex)	
0x000	2	01	00

This NMT message transitions all present NMT slaves to 'Operational' state

A *CANopen* devices usually implements a set of network services. These services' accessibility depends on the *NMT state* the devices currently reside in. The following table shows the availability of services implemented by the *SMC transducers*.

NMT state	Available services
Initialization	-
Pre-Operational	NMT, HBC, TIME, LSS, EMCY, SDO,
Operational	NMT, HBC, TIME, LSS, EMCY, SDO, SYNC, PDO
Stopped	NMT, HBC, TIME, LSS

HBC … Heartbeat Consumer

A *NMT Slave* (like the *SMC* transducers) typically is not allowed to automatically transfer its *state machine* to other states. The only two exceptions are made…

1) with the *Autostart* behaviour which was already described above,
2) in case of emergency situations (sensor faults, etc.): The *NMT state machine* is able to transfer itself to '*Pre-Operational*' or '*Stopped*' (depending on the configurations made to object 0x1029, *Error Behaviour*; refer to section 6.1.17).

4.4 The Object Dictionary

The *Object Dictionary* plays a central role in *CANopen*. It represents a table of variables (objects) that can be accessed for configuring and inquiring a *CANopen* device's state. Each table entry can be accessed with a combination of a 16-bit index and an 8-bit sub-index – thus there are 65536 indices whereof each index features up to 256 sub-indices. The *Object Dictionary* is accessible using *Service Data Objects (SDO)*.

The *SMC pressure/temperature transducers' Object Dictionary* is split into three parts:

- DS-301,
- DS-404,
- Manufacturer specific objects.

Section 6 deals with the supported objects and describes how to use them.

4.5 Service Data Object (SDO)

Service Data Objects are used to read and write variables from the *Object Dictionary*. These variables may contain settings or status/informational data, depending on the address (*index and sub-index*) that is being accessed. Each *CANopen* device provides at least one SDO channel which two CAN Identifier are assigned to.

There are two types of SDO accesses: *Expedited SDO* and *Segmented SDO*. Expedited SDO accesses are intended for transferring values not longer than 4 bytes – these accesses are the preferred way of performing SDO accesses to *SMC transducers*.
Segmented SDO accesses are necessary for transferring data that are longer than 4 bytes. This type of access will only be necessary for *reading string data* (VISIBLE_STRING).

A typical SDO access consists of five important things:

1) Address (*Node-ID*) of the *CANopen* device that is being accessed,
2) Access code (read/write, as well as the type of the accessed datum; will be discussed below),
3) Index of the object that is being accessed (16 bits),
4) Sub-index of the object that is being accessed (8 bits),
5) In case of write accesses: The data that shall be written (up to 4 bytes).

The table below shows the supported *data types* and their *SDO access codes* for reading/writing.

Data type	Length in bytes	Preferred SDO access type	Access code Reading	Access code Writing
BOOLEAN	1	Expedited	0x40	0x2F
Int8, Uint8	1	Expedited	0x40	0x2F
Int16, Uint16	2	Expedited	0x40	0x2B
Int32, Uint32	4	Expedited	0x40	0x23
Float32 (REAL)	4	Expedited	0x40	0x23
VISIBLE_STRING		Segmented	0x40	N/A

SDO is a confirmed service. This means that all read and write accesses will be answered. The *response's first data byte* does always contain an *error code* signalling if the SDO access was successful. The table below shows a set of possible *SDO error codes*.

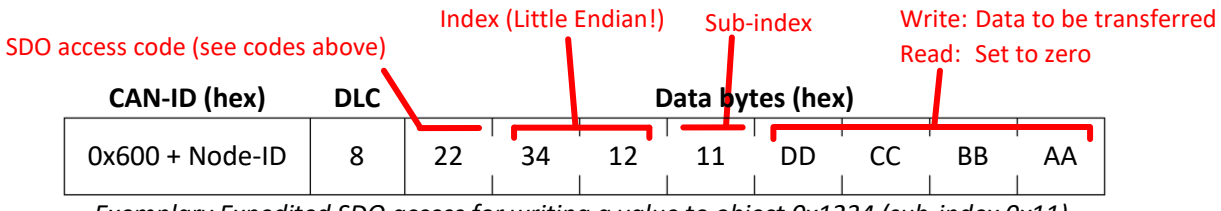
SDO error code	Description
0x41	Segmented read transmission initiated. Refer to 4.5.2.
0x42	Expedited reading succeeded, unspecified data length (or 0 bytes).
0x43	Expedited reading succeeded, 4 bytes available.
0x47	3 bytes available.
0x4B	2 bytes available.
0x4F	1 byte available.
0x60	Writing succeeded.
0x80	Error occurred. The last four data bytes contain more specific information.

The following sub-sections describe how to perform the two types of SDO accesses in order to read and write data from/to the *Object Dictionary*.

4.5.1 Expedited SDO

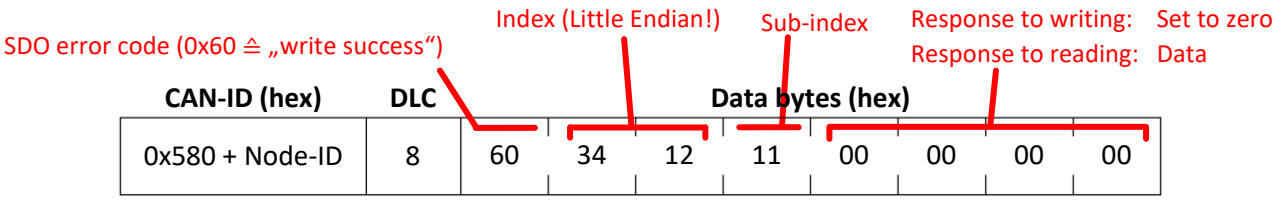
Expedited SDO will (as already described above) be used in order to access data values that are not longer than 4 bytes. This applies for all available data types – except for VISIBLE_STRING.

The following figure shows how to build up a CAN message containing a valid SDO access.



Exemplary Expedited SDO access for writing a value to object 0x1234 (sub-index 0x11).

A possible response to this *SDO write access* could be as follows:



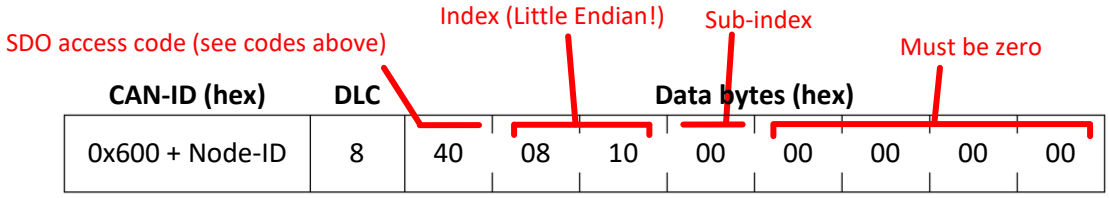
Exemplary SDO response to writing a value to object 0x1234 (sub-index 0x11).

4.5.2 Segmented SDO

Some data are larger than 4 bytes; their transmission will be done using *Segmented SDO*. Segmented transmissions are initiated in the same manner as *Expedited SDO*. Please be aware that the *SMC pressure/temperature transducers* do only support *Segmented SDO* in order to read values – writing is not possible.

The following example shows how to read the contents of object “Device Name” (0x1008, sub-index 0).

1) The following *CAN message* contains a regular SDO read access to the desired object.



*Step 1: Exemplary **initiation** of Segmented SDO access in order to read the contents of object “Device Name” (0x1008, sub-index 0).*

If the object *Device Name* actually holds a string and its length exceeds 4 characters, the *SDO response* contains the *error code* 0x41 – which indicates that a segmented transfer was initiated. The following message shows such an exemplary *SDO response*.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x580 + Node-ID	8	**41**	08	10	00	**0C**	00	00	00

*Step 1: The response code **0x41** indicates that a segmented transfer was initiated. The amount of readable data bytes is 12 (**0x0C**; data byte 5).*

2) Now, the next segment of 7 bits will be inquired. Therefore, the following message needs to be sent. The first data byte holds a specific value (0x60 or 0x70) that must be *toggled* with each inquiry. Initially, this toggling value is 0x60.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	**60/70**	00	00	00	00	00	00	00

*Step 2: Inquiring the next 7 data bytes. The first data byte must be toggled each time; initially it is **0x60**.*

The response's **first data byte** does contain some protocol overhead, splitting up into two parts: The uppermost four bits hold a toggling value, which assumes either 0x0 or 0x1. The lower four bits usually read zero; but in case of the last segment, they provide the amount of valid data bytes (1…7).

The following two messages show valid responses. Message **a** contains exemplary data originating from the middle of a sequence of transfers. Message **b** represents the last message out of a sequence of transfers.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x580 + Node-ID	8	**t0**	48	65	6C	6C	6F	20	57

*Step 2a: Exemplary data segment originating from the **middle** of a sequence of transfers. **t** stands for the uppermost four bits which contain the toggling value (0x0 or 0x1).*

CAN-ID (hex)	DLC	Data bytes (hex)							
0x580 + Node-ID	8	**t5**	6F	72	6C	64	21	00	00

*Step 2b: Exemplary data segment representing the **last message** out of a sequence of transfers. **t** stands for the uppermost four bits which contain the toggling value (0x0 or 0x1).*

When put together, the both messages' data contents result in the following *hexadecimal* byte sequence: *48 65 6C 6C 6F 20 57 6F 72 6C 64 21*, which stands for the ASCII-encoded string "*Hello World!*".

4.6 Process Data Object (PDO)

The main purpose of *Process Data Objects (PDO)* is to transfer process data (like pressure values) from one *CANopen* device to another. There are two kinds of PDOs which are a result of the specific data direction: inbound PDOs are referred to as *RPDOs*, outbound PDOs are called *TPDOs*. *SMC transducers* do support the *transmission of PDOs* and thus are *TPDO producers*.
When it's time for transferring a TPDO, the device generates the *CAN message's* data contents according to the individual TPDO mapping settings. TPDOs can only be transmitted as long as the *NMT state machine* (see 4.3) resides in the state '*Operational*'.

Each *SMC transducer* features two TPDOs. These TPDOs can be configured individually (refer to sections 6.1.18 and 6.1.19); examples for available settings are:

- Transmission trigger (timer, SYNC, event-driven),
- Enable/disable the transmission (TPDO2 is typically disabled by default),
- CAN identifier,
- TPDO mapping.

Both of the TPDOs each offer a set of *up to eight mappings* which can be assigned with several objects from DS-301, DS-404 and the manufacturer specific section. Their particular chapters (6.1, 6.2 and 6.3) offer tables summarizing which objects are available for mapping. By default, only the *Process Value (primary measuring channel)* is mapped to both TPDOs.

4.7 Synchronization Object (SYNC)

The *Synchronization Object (SYNC object)* is intended to synchronously initiate PDO transmission of multiple devices in the entire network. The SYNC object is supported by the *SMC transducers*.

4.8 Emergency Object (EMCY)

An *Emergency Object (EMCY)* is issued as soon as an error could be detected by a *CANopen* device. The *SMC transducers* additionally feature a volatile storage which holds up to fifty previously occurred errors; it is available at object index 0x1003 (see section 6.1.4).

The EMCY's *CAN Identifier* typically is 0x80 + Node-ID. The EMCY data bytes will be arranged as follows:

MSB LSB

16 bits	8 bits	16 bits	24 bits
EMCY code	*Error register (object 0x1001)*	*Additional information*	*Reserved (zero)*

The following table contains a set of possible EMCY codes:

EMCY code	Error class	Description
0x0000	-	Error reset or no error.
0x5030	Sensor error	Sensor error occurred. See field *Additional Information*.
0x8100	Communication error	Generic communication error.
0x8110		CAN overrun.
0x8120		CAN is now in state *Error Passive*.
0x8130		Heartbeat was not received.
0x8140		Recovered from *Bus Off*.
0x8150		CAN-ID collision.

In case of the EMCY code 0x5030 (*Sensor error*), the field *Additional Information* contains more precise information. It might be helpful to supply the manufacturer with the field's contents. The following table contains a set of exemplary values:

Additional Information	Description
0xC555	A/D watchdog error
0xCFCF	Bond wire broken

4.9 Time Stamp Object (TIME)

This *CANopen* device is capable of producing and consuming *Time Stamp Objects*. The *SMC transducers* implement an internal, milliseconds-based, permanently increasing time stamp which can (provided the device acts as *TIME* consumer) be synchronized with a network-wide time base.

In order to associate measurements and time stamps, the respective time stamp values are mappable to PDOs.

The *TIME* behaviour can be configured using the object *COB-ID time stamp* (0x1012; refer to section 6.1.11).

4.10 Heartbeat

For means of ensuring operability of network nodes, *CANopen* offers, inter alia, the automatic transmission of h*eartbeat* messages: each node can be configured to cyclically transmit a certain message in order to signal its readiness.

SMC transducers are able to generate and to consume *heartbeat* messages. The configuration is done by using objects 0x1016 and 0x1017 (refer to sections 6.1.13 and 6.1.14). The *CAN Identifier* used for heartbeat cannot be altered and calculates as follows:

$$\text{CAN-ID}_{\text{Heartbeat}} = 0x700 + \text{Node-ID}$$

4.11 Electronic Datasheet (EDS)

Most functions of the *SMC pressure/temperature transducers* are controlled by objects which are summarized within the *Object Dictionary*. The *Electronic Datasheet* provides a method to reflect the entries of the *Object Dictionary* to a standardized text-based file. These files can be read and interpreted by almost any software that implements *CANopen* functions.

The *SMC transducers'* EDS file reveals the manufacturer's default configuration; it can be obtained from [2].

4.12 Layer Setting Services (LSS)

The *SMC transducers* implement the *Layer Settings Services* (see [3]) in order to change *Node-ID* and *bit rate* of a CANopen node.

Be aware that, if LSS is used to change a device's **Node-ID**, some COB-IDs do not automatically adapt to the new Node-ID. If this is desired (what indeed most likely will be the case), usage of the object 0x4001 (see 6.3.7) should be given preference. Section 8.3 shows how to easily apply an alternative Node-ID.

When changing the **bit rate**, the user has the option to either use the pre-defined *CANopen* bit rate table (*table selector 0*; see Table 1), or to refer to a user-defined table (*table selector 1*). The user-defined table can hold up to 5 customer specific entries; please be aware that table must be defined when placing the order!

Bit rate	Bit rate code[1]
1000 kbit/s	0
800 kBit/s	1
500 kBit/s	2
250 kBit/s	3
125 kBit/s	4
100 kBit/s	5
50 kBit/s	6
20 kBit/s	7

Table 1: *Available standard bit rates according to [3] (bit rate table selector 0)*

Section 8.4 explains step-by-step how to configure a *SMC transducer's* bit rate using LSS.

[1] The *bit rate code* is used when configuring the *bit rate* using LSS.

5 DATA FLOW WITHIN THE SMC TRANSDUCER

SMC transducers feature a complex data path which will be applied to each measuring channel; see Figure 1. Each of the depicted objects offers at least two *sub-indices* in order to configure both measuring channels independently.

The *measuring channels'* assignment and thus the assignment of *sub-indices* depends on the kind of transducer:

- **Pressure transducers** offer two channels. The *primary measuring channel* refers to pressure, the *secondary channel* provides coarse temperature values,
- **Temperature transducers** offer only the *primary measuring channel*, which stands for temperature.

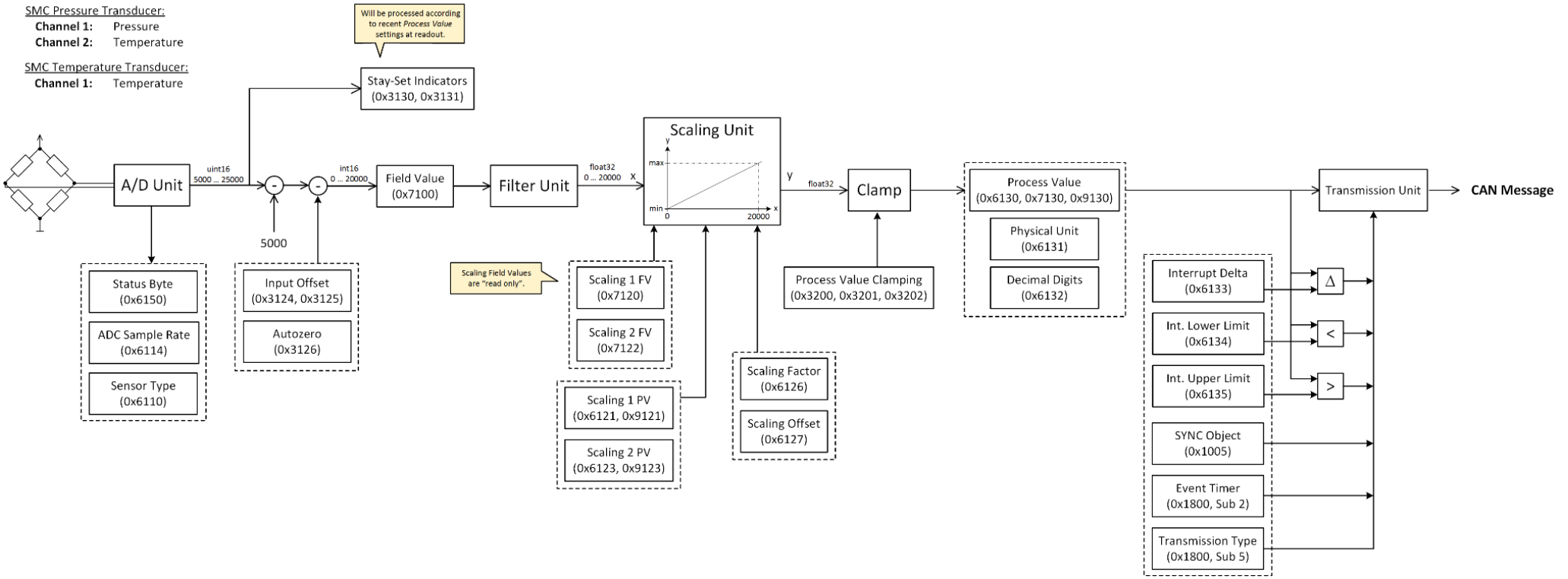


***Figure 1**: Data flow within the SMC transducers*

Almost all functional blocks from the data path are configurable using SDO. The respective object indices were annotated directly next to the blocks. While most of the functional blocks originate from [1], some of them can be found in the manufacturer specific objects section. For detailed descriptions regarding the objects depicted above, please refer to the sections 6.2 and 6.3.

Filter Unit

The Filter Unit is the only functional block that cannot be configured using SDO. DS-404 (refer to [1]), in fact, intends to support the configuration of certain filter types and therefore offers two objects; the IIR filter however is far too complex be reduced to two objects.

The implemented filter represents a 5-stage IIR filter ("*Infinite Impulse Response*"). Figure 2 illustrates its general logical structure.

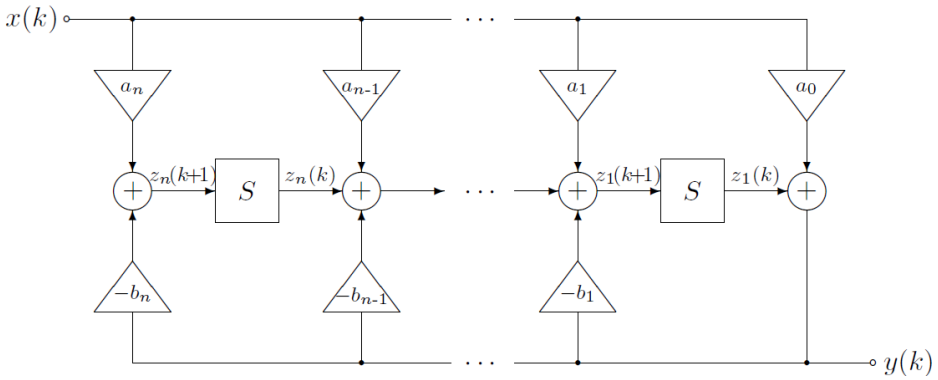


***Figure 2**: 5-stage IIR Filter Unit*

As seen above, the implementation of an IIR filter relies on multiple coefficients – in the case of a 5-stage filter, there are eleven coefficients. With skilful selection of suitable values, many different filter types can be realized, such as: *Butterworth, Chebyshev*, et cetera. By default, the function of a *floating average filter* is fulfilled. If desired, customers may define specific filter coefficients when placing the order.

Scaling Unit (and alternative transfer functions)

DS-404 (see [1]) suggests the implementation of a Scaling Unit in order to perform linear transformations on *Field Values*, which subsequently are referred to as *Process Values*. *SMC transducers* follow this suggestion; they implement the Scaling Unit as standard *transfer function*.

Alternatively, if desired, customers may ask to implement other *transfer functions*. Already implemented functions are, inter alia, *deviations, integrations, polynomial functions* or *spheric/cylindric fill grade approximations*. The handling of alternative transfer functions is – neither switching from one function to another, nor the configuration of single functions – not yet implemented in the form of SDO accesses. Thus, customers should be indicating their desire for an alternative transfer function when or before placing the order.

Input offset

The Input Offset bases on the *Field Value*, not on the *Process Value* (as proposed by DS-404; [1]). Benefit is data consistency, as the offset value does not change as soon as any settings are made to the Scaling Unit. In order to prevent confusion, the two objects 0x3124 and 0x3125 were introduced. Refer to section 6.3.1.

6 THE OBJECT DICTIONARY OF THE SMC TRANSDUCER

The following sections describe the accessible entries of the *SMC pressure/temperature transducer's* Object Dictionary.

6.1 Communication Profile Specific Entries (DS-301)

Index	Meaning	Mappable to TPDO?
0x1000	Device type	
0x1001	Error register	✓
0x1002	Manufacturer status register	
0x1003	Pre-defined error field	
0x1005	COB-ID SYNC	
0x1008	Manufacturer device name	
0x1009	Manufacturer hardware version	
0x100A	Manufacturer software version	
0x1010	Store parameters	
0x1011	Restore default parameters	
0x1012	COB-ID time stamp	
0x1014	COB-ID EMCY	
0x1016	Consumer heartbeat time	
0x1017	Producer heartbeat time	
0x1018	Identity object	
0x1020	Verify configuration	
0x1029	Error behaviour object	
0x1800 ... 0x1801	TPDO communication parameter	
0x1A00 ... 0x1A01	TPDO mapping parameter	
0x1F80	NMT startup behaviour	

6.1.1 0x1000 (Device Type)

This object provides information about the device type. Its contents are constant and thus not writable.

Sub-index	Data type	Access	Note
0	Uint32	CONST	0x80 02 01 94[2]

6.1.2 0x1001 (Error register)

Provides information on the error status of the A/D sensor part. It is part of the EMCY message and may be mapped to a TPDO.

Sub-index	Data type	Access	Note
0	Uint8	RO	Lowest bit ("Generic Error") contains 'High' in case of a device-internal A/D error.

[2] The lower 16 bits part holds the *device profile number* (404_{10}). The upper 16 bits contain additional information.

If this error field once contains an error, it can only be reset to zero with a *power-off, power-on sequence*. That is because of the high severity of A/D errors.

More fine-grained error information may be inquired using the *manufacturer status register* (0x1002; see 6.1.3).

6.1.3 0x1002 (Manufacturer status register)

This field combines the two *status bytes* of each temperature and pressure channel (0x6150; see 6.2.5).

The value contained regards

- the error status of the A/D path,
- beyond-maximum-limits information for *primary and secondary measuring channels* (usually pressure and temperature).

Sub-index	Data type	Access	Note
0	Uint32	RO	See below.

The data contents are assembled in the following manner:

MSB LSB

16 bits	8 bits	8 bits
Reserved for future use	*Secondary channel[3] status byte (usually temperature)*	*Primary channel[3] status byte (usually pressure)*

6.1.4 0x1003 (Pre-defined error field)

This object provides a history about the errors signalled via the EMCY message (see 4.8).

Sub-index	Data type	Access	Note
0	Uint32	RW	Read access: Number of errors (0x00…0xFE) Write access: Writing zero deletes history.
0x01 … 0xFE	Uint32	RO	See below.

Each newly triggered error is assigned to the sub-index 0x01. The sub-indices of all previous errors will be incremented by one. The 32-bit error field consists of two 16-bit values.

MSB LSB

16 bits	16 bits
Additional information	*Error code*

[3] In case of explicit temperature sensors, the *primary channel* refers to temperature. The secondary channel status byte may then be neglected.

The field "*Error code*" holds the EMCY error code defined by CANopen. In case of a sensor error, this field carries the value 0x5030 ("Sensor fault").

The field "*Additional information*" contains the same-named field originating from the EMCY message. In case of 0x5030 ("*Sensor fault*"), it might be helpful to supply the manufacturer with this field's contents.

6.1.5 0x1005 (COB-ID SYNC)

This field allows to configure the COB-ID of the synchronization object (SYNC).

Sub-index	Data type	Access	Note
0	Uint32	RW	See below.

The data contents are assembled in the following manner:

MSB LSB

1 bit	**1 bit**	**1 bit**	**18 bits** (set to zero)	**11 bits**
			29 bits	
Don't care	*Don't care*	*Frame type (0 = 11 bits, 1 = 29 bits)*	*CAN-ID (11 bits or 29 bits)*	

This CANopen device does not implement the ability to generate SYNC messages.

Typical CAN-ID according to CANopen specification: 0x80.

6.1.6 0x1008 (Manufacturer device name)

This object allows reading access to the *manufacturer device name* field.

Sub-index	Data type	Access	Note
0	VISIBLE_STRING	CONST	-

This string may be altered using the manufacturer-defined object 0x4022 (refer to 6.3.11). The maximum length of this (null-terminated) string is 60 characters.

6.1.7 0x1009 (Manufacturer hardware version)

This object provides the manufacturer hardware version.

Sub-index	Data type	Access	Note
0	VISIBLE_STRING	CONST	*Encoding*: "*vYYWW\0*"[4]

6.1.8 0x100A (Manufacturer software version)

This object provides the manufacturer software version.

Sub-index	Data type	Access	Note
0	VISIBLE_STRING	CONST	*Encoding*: "*vYYWW\0*"[4]

[4] '*YY*' corresponds to the year, '*WW*' to the week. The character '\0' represents the string null termination.

6.1.9 0x1010 (Store parameters)

This object lets the user save the current parameter set to EEPROM. Without saving, all changes will be overwritten at next boot-up.

Sub-index	Data type	Access	Note
0	Uint32	CONST	Always returns 1.
1	Uint32	RW	Reading returns 1; writing see below.

Writing the Uint32-value 0x65766173[5] to sub-index 1 will make the *SMC pressure/temperature transducer* save its current parameter set.

Provided the EEPROM access was successful, the CANopen device answers with the SDO code 0x60. It is strongly recommended to wait 500ms after this command to avoid memory corruptions.

6.1.10 0x1011 (Restore default parameters)

This object allows the user to restore all parameters to delivery-default-state.

Sub-index	Data type	Access	Note
0	Uint32	CONST	Always returns 1.
1	Uint32	RW	Reading returns 1; writing see below.

Writing the Uint32-value 0x64616F6C[6] to sub-index 1 will make the *SMC pressure/temperature transducer* overwrite its current parameter set and load the factory parameters. Additionally, the parameters will be assigned to the boot-up parameters section – a further *"Store parameters"* is not necessary.

An exception is made with the *bit rate* and *Node-ID*: These values will be preserved to avoid subsequent network inconsistencies.

It is strongly recommended to wait 500ms after this command to avoid memory corruptions.

Provided the EEPROM access was successful, the CANopen device answers with the SDO code 0x60.

6.1.11 0x1012 (COB-ID time stamp)

This field allows to configure the COB-ID of the timestamp object (TIME).

Sub-index	Data type	Access	Note
0	Uint32	RW	See below.

The data contents are assembled in the following manner:

[5] This value stands for the ASCII string "evas", which will be "save" in reverse. This procedure makes the CANopen device less susceptible for unintentional EEPROM write accesses.
[6] This value stands for the ASCII string "daol", which will be "load" in reverse.

MSB LSB

1 bit	1 bit	1 bit	**18 bits** (set to zero)	**11 bits**
			29 bits	
Consuming enabled	*Producing enabled*	*Frame type (0 = 11 bits, 1 = 29 bits)*	*CAN-ID (11 bits or 29 bits)*	

This CANopen device is both able to consume and to produce timestamp objects. Nevertheless, only one of both operating modes can be enabled at once.

Typical CAN-ID according to CANopen specification: 0x100.

Hint: The two manufacturer specific objects 0x3140 and 0x3141 allow read-only access to the current time stamp values. Both of them are mappable to TPDOs.

6.1.12 0x1014 (COB-ID EMCY)

This field allows to configure the COB-ID of the emergency object (EMCY).

Sub-index	Data type	Access	Note
0	Uint32	RW	See below.

The data contents are assembled in the following manner:

MSB LSB

1 bit	1 bit	1 bit	**18 bits** (set to zero)	**11 bits**
			29 bits	
Disable EMCY	*Always zero*	*Frame type (0 = 11 bits, 1 = 29 bits)*	*CAN-ID[7] (11 bits or 29 bits)*	

According to CANopen, the default CAN-ID of the EMCY object is: 0x80 + Node-ID.

6.1.13 0x1016 (Consumer heartbeat time)

This object allows to configure this CANopen device as Heartbeat consumer. The monitoring of the Heartbeat producer starts after the reception of the first Heartbeat.

Sub-index	Data type	Access	Note
0	Uint32	CONST	Always returns 1.
1	Uint32	RW	See below.

The data contents are assembled in the following manner:

[7] When altering the device Node-ID, none of any configured CAN-IDs will change automatically.

MSB LSB

8 bits	8 bits	16 bits
Always zero	*Producer Node-ID (0 = Feature Disabled)*	*Heartbeat time (milliseconds)*

It is recommended to choose a slightly higher *Heartbeat time* value than the actual *producer Heartbeat time* is.

If no Heartbeat message could be received within the specified time, the following two actions will be performed:

1) Transmission of an EMCY message. The EMCY code is 0x8130.
2) Alteration of the transducer's CANopen state machine. Depends on the contents of object 0x1029 ("Error behaviour"; see 6.1.17). The default target state is *Pre-Operational Mode*.

6.1.14 0x1017 (Producer heartbeat time)

This object allows to configure this CANopen device as Heartbeat producer.

Sub-index	Data type	Access	Note
0	Uint16	RW	Producer Heartbeat time (milliseconds)

If a time of zero is chosen, the *producer Heartbeat* feature is disabled.

6.1.15 0x1018 (Identity object)

This object provides general identification information on the CANopen device.

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 4.
1	Uint32	RO	Vendor-ID
2	Uint32	RO	Product Code
3	Uint32	RO	Revision Number
4	Uint32	RO	Serial Number

The *Vendor-ID* is uniquely assigned to every vendor selling CANopen products. The value 0 indicates an invalid *Vendor-ID*. A list of all *Vendor-IDs* can be found at [4].

6.1.16 0x1020 (Verify configuration)

This object holds the *date and time of the last configuration*.

Sub-index	Data type	Access	Note
0	Uint32	CONST	Always returns 2.
1	Uint32	RW	The number of days since January 1, 1984.
2	Uint32	RW	The number of ms after midnight.

Each time changes are made to any CANopen registers, both date and time values are automatically reset to zero. This may be used to recognize (un-)intended manipulation of the device configuration.

6.1.17 0x1029 (Error behaviour object)

In case of device failures, a transition of the CANopen state machine may be desired. This object allows to configure the state that will be switched to.

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 3.
1	Uint8	RW	Communication error behaviour
2	Uint8	RW	(not evaluated)
3	Uint8	RW	Analog input error behaviour

The following table shows the values that may be contained within the sub-indices:

Value	Targeted state
0	*Pre-Operational* (if currently *Operational*)
1	No alteration of the state machine desired.
2	*Stopped*

By default, the value 0 is assigned to all sub-indices.

6.1.18 0x1800 … 0x1801 (TPDO communication parameter)

These objects allow to configure several parameters that specify, how and when PDOs are being transmitted. Object 0x1800 configures TPDO1, 0x1801 configures TPDO2.

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 5.
1	Uint32	RW	*Identifier parameters*. See below.
2	Uint8	RW	*Transmission type*. See below.
3	-	-	(Reserved)
4	-	-	(Reserved)
5	Uint16	RW	*Event timer*. See below.

Identifier parameters

The *Identifier parameters* are assembled in the following manner:

MSB LSB

1 bit	**1 bit**	**1 bit**	**18 bits** (set to zero)	**11 bits**
			29 bits	
Disable TPDO	*Don't care (Always one)*	*Frame type (0 = 11 bits, 1 = 29 bits)*	*CAN-ID[8] (11 bits or 29 bits)*	

[8] When altering the device Node-ID, none of any configured CAN-IDs will change automatically.

RTR frames are not supported for PDOs (refer to *IG CANopen 2003-05-12*). Thus, bit 30 will always be set to one.
The default CAN-ID (refer to [5]) calculates as follows:

0x180 + Node-ID (TPDO1)
0x280 + Node-ID (TPDO2)

Transmission type

The *transmission type* decides about when a PDO is transmitted. The following table shows the available values.

Transmission type	Description
0	Synchronous, acyclic
1 … 240 (0x1 … 0xF0)	Synchronous, cyclic every n^{th} SYNC
254 (0xFE)	Event-driven (Event timer)
255 (0xFF)	Event-driven (Measuring event)

The configuration of the *measuring events* is part of the DS-404 objects. Refer to 6.2.8 and 0.

Event timer

The *event timer* cyclically triggers the PDO transmission. The time granularity is milliseconds. For enabling the timer, both of the two following conditions must be fulfilled:

- *Transmission type* is configured to either 0xFE or 0xFF (0xFE is recommended to not interfere with *measuring events*),
- The configured *event timer* value is not zero.

6.1.19 0x1A00 … 0x1A01 (TPDO mapping parameter)

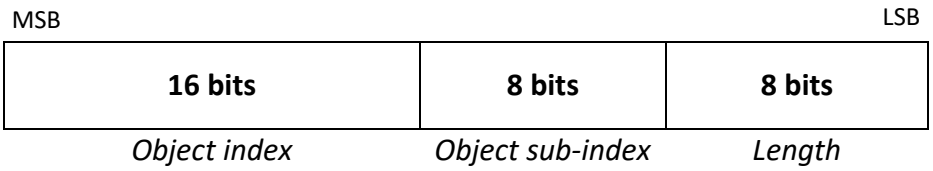
These objects allow to configure the data contents of TPDOs. Object 0x1A00 configures TPDO1, 0x1A01 configures TPDO2.

Each TPDO offers 8 *mapping entries*. These *mapping entries* can be accessed and modified using the sub-indices 1 to 8. There are several objects that are mappable to a TPDO – the tables in sections 6.1, 6.2 and 6.3 reveal all available objects.

Sub-index	Data type	Access	Note
0	Uint8	RW	Contains the number of enabled mapping entries for this TPDO (max. 8). Zero disables the mapping (and the TPDO).
1 … 8	Uint32	RW	The specific mapping entry. See below.

The order of mapped objects within the resulting TPDO is given by the used sub-index numbers. This means that the first *mapping entry* will be mapped to the first TPDO byte(s).

The *mapping entries* can only edited if the mapping is disabled (sub-index 0 carries the value zero). A *mapping entry* contains the following data:



The field "*Length*" must always correspond to the actual length (bit count) of the desired object. Allowed values are: 0x08, 0x10, 0x20, 0x40.

6.1.20 0x1F80 (NMT startup behaviour)

This object allows to configure the *startup behaviour* of the *SMC pressure/temperature transducer*. Usually, a CANopen device resides in NMT state *Pre-Operational* after power-on. In order to transition to *Operational* state, a NMT network master is necessary. To circumvent this, the node can be configured to automatically switch to *Operational* state.

Sub-index	Data type	Access	Note
0	Uint32	RW	The allowed values are: 0x00 … Reside in *Pre-Operational*. 0x02 … Switch to *Operational* and transmit NMT message "*Start All Nodes*". 0x08 … Switch to *Operational*.

6.2 Device Profile Specific Entries (DS-404)

Each of the following objects features two sub-indices. They correspond to the two measuring channels of the SMC pressure transducer: The *primary channel* (sub-index #1) usually refers to the pressure value, the *secondary channel* refers to the auxiliary temperature channel.

In case of explicit temperature sensors, the *primary channel* refers to temperature. The secondary channel may then be neglected.

Index	Meaning	Data Type	Mappable to TPDO?
General data			
0x6110	Sensor Type	Uint16	
0x6114	ADC Sample Rate	Uint32	
0x6131	Physical Unit	Uint32	
0x6132	Decimal Digits Process Value	Uint8	
0x6150	Status Byte (“Analog Input Status”)	Uint8	✓
Linear input scaling			
0x6121	Input Scaling – Process Value 1 (min)	Float32	
0x6123	Input Scaling – Process Value 2 (max)	Float32	
0x9121	Input Scaling – Process Value 1 (min)	Int32	
0x9123	Input Scaling – Process Value 2 (max)	Int32	
0x7120	Input Scaling – Field Value 1 (min)	Int16	
0x7122	Input Scaling – Field Value 2 (max)	Int16	
0x6126	Scaling Factor	Float32	
0x6127	Scaling Offset	Float32	
Sensor readings			
0x6130	Process Value	Float32	✓
0x7130	Process Value	Int16	✓
0x9130	Process Value	Int32	✓
0x7100	Field Value	Int16	✓
Configuration of event-triggered PDO transmission			
0x6133	Process Value Interrupt Delta	Float32	
0x6134	Process Value Interrupt Lower Limit	Float32	
0x6135	Process Value Interrupt Upper Limit	Float32	

6.2.1 0x6110 (Sensor Type)

This object provides the *sensor types* of the two measuring channels.

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 2.
1	Uint16	CONST	*Sensor type* of the primary measuring channel. Typically 90 ("*Pressure transducer*").
2	Uint16	CONST	*Sensor type* of the secondary measuring channel. Typically 100 ("*Temperature transducer*").

6.2.2 0x6114 (ADC Sample Rate)

This object contains the ADC sample rate in *microseconds*. This value is just for informational purposes and cannot be changed.

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 2.
1	Uint32	CONST	*ADC sample rate* of the primary measuring channel. Typically 1000.
2	Uint32	CONST	*ADC sample rate* of the secondary measuring channel. Typically 1000.

6.2.3 0x6131 (Physical Unit)

This object contains the *physical unit value* for the two measuring channels. Altering these fields does not influence any of the in-sensor calculations.

Pressure measuring channels will by default always carry the value 0x004E0100 ("*bar*"); *temperature measuring channels* will carry 0x002D0100 ("*°C*").

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 2.
1	Uint32	RW	Contains the *physical unit* of the primary measuring channel.
2	Uint32	RW	Contains the *physical unit* of the secondary measuring channel.

The units are defined according to the CANopen standard (refer to [1] and [6]).

6.2.4 0x6132 (Decimal Digits Process Value)

This object specifies the number of *decimal digits after the decimal point* when using *sensor readings* (process value!) of the data types *int8*, *int16* and *int32*. The *float32* data type won't be affected.

The allowed value range is from 0 to 9.

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 2.
1	Uint8	RW	Primary measuring channel.
2	Uint8	RW	Secondary measuring channel.

Example:
A *process value* of 1.92 will be converted to 192 if the *decimal digits process value* is set to 2.

6.2.5 0x6150 (Status Byte)

This object reflects the internal status of the measuring device. It is mappable to TPDOs.

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 2.
1	Uint8	RO	*Status byte* of the primary measuring channel. See below.
2	Uint8	RO	*Status byte* of the secondary measuring channel. See below.

Each channel's status byte will be assembled in the following manner:

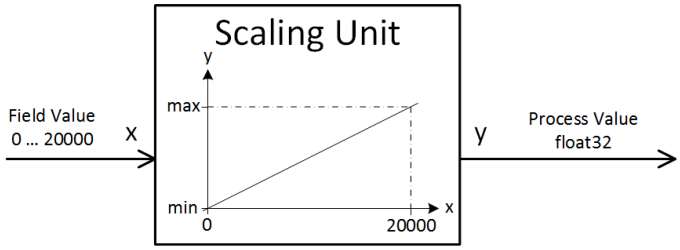
MSB LSB

5 bits	**1 bit**	**1 bit**	**1 bit**
Reserved	*Neg. overload*	*Pos. overload*	*Not valid (e.g. sensor break)*

The status bytes of the two measuring channels will be put into the two lower bytes of the *manufacturer status register* (object 0x1002; refer to section 6.1.3).

6.2.6 Input Scaling

The *input scaling* objects enable the user to scale the output values (*Process Values*). The following image shows the linear scaling unit of the SMC data flow (refer to section 5).



Configure scaling using min/max process values

The objects "*Input Scaling – Field Value 1/2*" (0x7120, 0x7122) do correspond to the values 0 and 20000. They are CONST and thus cannot be changed. As the underlying *field value* data type is *int16*, these objects are only available as *int16* types.

The desired *process value* range may be configured using the objects "*Input Scaling – Process Value 1/2*". The underlying data type is *float32*, hence there are objects for *float32* and *int32* access.

Configure scaling using linear function coefficients
The scaling is performed using a linear transformation according to the following formula:

$$y = Gain \cdot x + Offset \qquad \text{with } x = Field\ Value,\ y = Process\ Value$$

Instead of configuring the desired min/max process values, the user may directly manipulate the *Gain/Factor* and *Offset* parameters (0x6126, 0x6127). These parameters are accessible as *float32* value types.

Important: It is recommended to always write the *Offset* value at first! Afterwards, the *Gain/Factor* value should be written. That is, because writing to *Gain/Factor* performs some calculations where the *Offset* value is used.

Hint: For consistency purposes, the *min/max process values* and the *linear coefficients* will always be in sync!

6.2.7 Sensor readings

The completely scaled output values are available as various data types: *float32*, *int16* and *int32*. The *int16* and *int32* values additionally will be comma-shifted – according to the settings of object 0x6132 ("*Decimal Digits Process Value*"; refer to section 6.2.4).

The *field values* correspond to the values before scaling and filtering. They typically range from 0 to 20000. As the underlying data type is *int16*, the *field values* are only accessible as *int16*.

For all F*ield* and P*rocess Values*, there is the option to be outputted in byte-orders *Little Endian* or *Big Endian*; refer to the manufacturer specific object 0x4000. According to *CANopen*, the default byte-order is *Little Endian*.

All sensor readings objects are mappable to TPDOs. Examples on how to obtain measurements (using SDO) can be found at section 8.6.

6.2.8 0x6133 (Process Value Interrupt Delta)

This object allows to configure the *Process Value Interrupt Delta*. It is used to trigger PDO transfer if the P*rocess Value* has changed more than the defined delta since last PDO transmission. A value of 0 disables this feature.

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 2.
1	Float32	RW	Primary measuring channel.
2	Float32	RW	Secondary measuring channel.

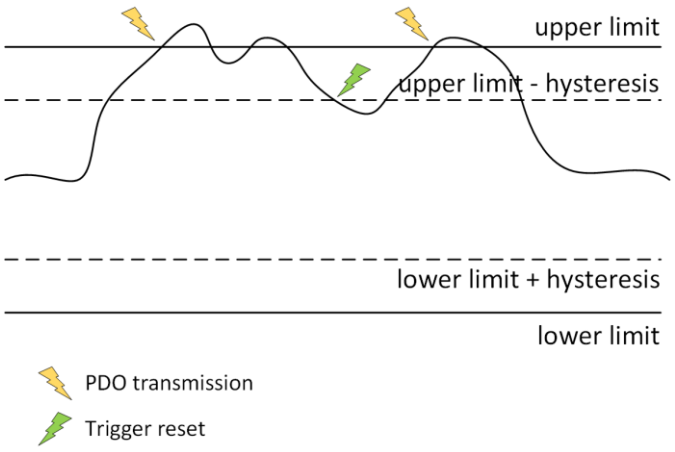
In order to use this feature, it is necessary to set the PDO transmission type to 0xFF. Refer to section 6.1.18.

6.2.9 0x6134, 0x6135 (Process Value Interrupt Lower/Upper Limit)

These two objects allow to configure the lower and upper limits for *event triggered PDO transmission*.

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 2.
1	Float32	RW	Primary measuring channel.
2	Float32	RW	Secondary measuring channel.

Each time the lower/upper limits are exceeded, a PDO transmission will be triggered. To prevent continuous PDO transmissions in certain cases, this feature implements a hysteresis which equals 1% of the total measuring span. Before a next PDO transmission triggers, the P*rocess Value* must go *inside the limits* and *beyond the hysteresis*. The illustration below shows this procedure.



In order to use this feature, it is necessary to set the PDO transmission type to 0xFF. Refer to section 6.1.18.

Disabling this feature can be achieved by assigning “very high”/”very low” values to the *higher/lower limit* registers. The highest possible *float32* value is 0x7F7FFFFF; the lowest possible *float32* value is 0xFF7FFFFF.

6.3 Manufacturer Specific Entries

Index	Meaning	Data Type	Mappable to TPDO?
Offset compensation[9] (based on *Field Values*)			
0x3124	Input Offset	Int16	
0x3125	Input Offset	Float32	
0x3126	Autozero	Uint32	
Stay-set indicators (slave pointers)			
0x3130	Uptime Stay-Set Indicators	Float32	
0x3131	Lifetime Stay-Set Indicators	Float32	
Time stamps (based on values supplied by TIME producer)			
0x3140	Time Stamp – Days	Uint16	✓
0x3141	Time Stamp – Milliseconds	Uint32	✓
Process value clamping			
0x3200	Process Value Clamping Info	Uint8	
0x3201	Process Value Clamping Min	Float32	
0x3202	Process Value Clamping Max	Float32	
Additional fields			
0x3150	Date of Calibration (UNIX Time)	Uint64	
0x4000	Measurements Endianness	Uint8	
0x4001	Node-ID	Uint8	
0x4010	Hardware Settings	Uint8	
0x4020	User Comment String	VISIBLE_STRING	
0x4021	Maintenance String	VISIBLE_STRING	
0x4022	Device Name String	VISIBLE_STRING	
0x4030	User-Defined Serial Number	Uint32	
0x4031	Time of Maintenance	Uint32	

[9] The objects used for *offset compensation* do intentionally differ from the objects suggested by DS-404. That is to prevent confusion, because the offset refers to *Field Values* instead of *Process Values*. Refer to section 5.

6.3.1 Offset Compensation

The *SMC pressure measuring sensor* features an offset compensation. Its objects each do have two sub-indices: The first sub-index allows access to the *primary measuring channel*; the second index allows access to the *secondary measuring channel*.

0x3124, 0x3125 (Input Offset)

The *Input Offset* will be subtracted from the *Field Value* before it will be scaled in the *Scaling Unit* (refer to section 5). *Field Value's* data type is *int16*; it ranges from 0 to 20000. Additionally, the *Input Offset* values can be accessed as *float32*.

0x3126 (Autozero)

The *Autozero* can be used to assume the current measurement as "zero"[10]. Therefore, the recent *Field Value* will be copied into the *Input Offset* register.

Sub-index	Data type	Access	Note
0	Uint32	CONST	Always returns 2.
1	Uint32	W	Primary measuring channel. See below.
2	Uint32	W	Secondary measuring channel. See below.

For initiating the *Autozero* procedure, the u*int32* value 0x6F72657A[11] needs to be written to the desired sub-index.

6.3.2 Stay-set indicators (slave pointers)

Stay-set indicators (or slave pointers) can be used to find minimum/maximum values within a pressure or temperature application. There are two sets of *stay-set indicators*, accessible at different objects:

- *Uptime stay-set indicators (0x3130)*: Keep track of the applied minimum/maximum pressure and temperature as long as device is powered. Will be reset at reboot.
- *Lifetime stay-set indicators (0x3131)*: Applied minimum/maximum pressure and temperature values are persistent will not be erased at reboot.

All captured values refer to the point before subtraction of the *Input Offset* values (see "data flow", section 5). Thus, the stay-set indicator values will not be affected in case *Input Offset* or *scaling* are being altered.

At readout, *Scaling, Input Offset and Clamping* will be applied **according to the respective recent *Process Value* settings**.

The both objects' sub-indices are structured as follows:

Sub-index	Data type	Access	Note
0	Uint32	RW	Always returns 4. Write access see below.
1	Float32	RO	Minimum value for *primary measuring channel*
2	Float32	RO	Maximum value for *primary measuring channel*
3	Float32	RO	Minimum value for *secondary measuring channel*

[10] Only *Field Values* differing at most ±10% from the actual "zero" value can be assumed as the new zero point.
[11] This value stands for the ASCII string "orez", which will be "zero" in reverse.

4	Float32	RO	Maximum value for *secondary measuring channel*

Writing the u*int32* value 0x74657372[12] to sub-index 0 resets the *stay-set indicators*. It is not necessary to subsequently store the parameter set (object 0x1010).

6.3.3 Time stamps

These objects allow read access to the *current time stamp values* of this CANopen device. Provided there is a TIME producer present within the application network, the hereby obtained values will correspond to the supplied *time stamp* values. The values are in particular:

- *Days since 1 January 1984* (0x3140),
- *Milliseconds after midnight* (0x3141).

These objects are intended to be mapped into TPDOs.

6.3.4 Process value clamping

After *scaling*, the *Process Value* may optionally be clamped – refer to "data flow" (section 5). There are three objects for handling the *Process value clamping*:

- Process Value Clamping Info (0x3200),
- Process Value Clamping Min (0x3201),
- Process Value Clamping Max (0x3202).

Each of the objects features two sub-indices. The first sub-index refers to the *primary measuring channel* (typically pressure); the second sub-index refers to the *secondary measuring channel* (typically temperature in case of *SMC pressure transducer*).

0x3200 (Clamping Info)

This object allows to inquire the activation state of the clamping feature and, in addition, to disable it.

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 2.
1	Uint8	RW	Primary measuring channel. See below.
2	Uint8	RW	Secondary measuring channel. See below.

Read access: Returns 1 if clamping is enabled. Otherwise 0.
Write access: Writing 0 disables the *Process Value* clamping.

0x3201, 0x3202 (Clamping Min, Max)

These two objects allow to manipulate the applied clamping limits.

Hint: Disabling the clamping does assign the least and the largest possible *float32* values to the minimum/maximum registers. These are 0x7F7FFFFF and 0xFF7FFFFF.

[12] This value stands for the ASCII string "tesr", which will be "rset" (alias "reset") in reverse.

Sub-index	Data type	Access	Note
0	Uint32	CONST	Always returns 2.
1	Float32	RW	Primary measuring channel.
2	Float32	RW	Secondary measuring channel.

6.3.5 0x3150 (Date of Calibration)

This object stores the date of calibration. The format is *Unix Time Format* (seconds passed since 1 January 1970, 00:00 UTC).

Sub-index	Data type	Access	Note
0	Uint64	RO	-

6.3.6 0x4000 (Measurements Endianness)

This object controls the measurements' endianness (also known as byte-order). It applies to all *Field and Process Values* outputted via SDO and TPDO.

Sub-index	Data type	Access	Note
0	Uint8	RW	0 = Little Endian 1 = Big Endian

According to the CANopen specification, the default endianness is *Little Endian*.

6.3.7 0x4001 (Node-ID)

This object enables the user to change the device's Node-ID without using LSS. In addition, it offers an automatic COB-ID assignment, which might be useful in many cases.

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 2.
1	Uint8	RW	Returns current Node-ID. Writing: See below.
2	Uint8	RW	Returns current Node-ID. Writing: See below.

Writing to sub-index 1:

When writing a valid Node-ID[13] to sub-index 1, the new Node-ID will be applied. None of the COB-IDs will be changed implicitly. Using this sub-index has the same effect as changing the Node-ID via LSS.

Writing to sub-index 2:

When writing a valid Node-ID[13] to sub-index 2, the new Node-ID will be applied **and** CANopen *default-values* will be written to the following *COB-ID registers*. All COB-IDs' uppermost two bits will be preserved for consistency purposes.

- COB-ID SYNC = 0x80

[13] Valid Node-IDs are values between 1 and 127.

- COB-ID TIME = 0x100
- COB-ID EMCY = 0x80 + Node-ID
- COB-ID TPDO1 = 0x180 + Node-ID
- COB-ID TPDO2 = 0x280 + Node-ID

After successfully adapting the new Node-ID, no *SDO success message* will be transmitted. Instead, the device simulates a complete reboot process including the transmission of a *boot-up message*. Be aware that the CANopen *state-machine* will be reset according to the settings of the object "*NMT startup behaviour*" (0x1F80).

Hint: Changing the Node-ID using this object does not automatically persist the changes to EEPROM. This needs to be done separately using object 0x1010 (refer to section 6.1.9).

6.3.8 0x4010 (Hardware Settings)

This object enables the user to inquire information about the *hardware options* of this *SMC pressure/temperature transducer* and to enable *the switchable bus termination* (if available).

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 3.
1	Uint8	CONST	Returns if *switchable termination* is available. 0 = Not available 1 = Available
2	Uint8	RW	If *switchable termination* exists, this sub-index allows to *enable/disable* it. 0 = Termination disabled (or not available) 1 = Termination enabled.
3	Uint8	CONST	Returns if *galvanic bus isolation* is available. 0 = Not available 1 = Available

6.3.9 0x4020 (User Comment String)

Allows to store an additional *user comment string*.

Sub-index	Data type	Access	Note
0	VISIBLE_STRING	RW	-

Strings consisting of less than 80 symbols should be null-terminated.

This object supports transmission of multiple string characters at once. It also supports segmented SDO transfers (refer to section 4.5.2).

6.3.10 0x4021 (Maintenance String)

Allows to store an additional *maintenance string*.

Sub-index	Data type	Access	Note
0	VISIBLE_STRING	RW	-

Strings consisting of less than 60 symbols should be null-terminated.

This object does support transmission of multiple string characters at once. It also does support segmented SDO transfers (refer to section 4.5.2).

6.3.11 0x4022 (Device Name String)

Allows to read/modify the d*evice name string*. This string corresponds to the *Manufacturer Device Name*, accessible at object 0x1008 (refer to section 6.1.6).

Sub-index	Data type	Access	Note
0	Uint8	CONST	Returns the highest available sub-index (60).

Strings consisting of less than 60 symbols should be null-terminated.

This object does support transmission of multiple string characters at once. It also does support segmented SDO transfers (refer to section 4.5.2).

6.3.12 0x4030 (User-Defined Serial Number)

This object contains a 64-bit *user-defined serial number*. Its lower 32 bits correspond to the serial number used at LSS and at object 0x1018 (sub-index 4).

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 2.
1	Uint32	RW	Lower 32 bits
2	Uint32	RW	Higher 32 bits

Write access to sub-index 1 (*lower 32 bits*) sets the *higher bits part* to zero. Thus, sub-index 1 should always be written at first.

6.3.13 0x4031 (Time of Maintenance)

This object contains a 64-bit storage for the *time of maintenance*. The suggestion is to use the *Unix Time Format* (seconds passed since 1 January 1970, 00:00 UTC).

Sub-index	Data type	Access	Note
0	Uint8	CONST	Always returns 2.
1	Uint32	RW	Lower 32 bits
2	Uint32	RW	Higher 32 bits

Write access to sub-index 1 (*lower 32 bits*) sets the *higher bits part* to zero. Thus, sub-index 1 should always be written at first.

7 ELECTRICAL CHARACTERISTICS

7.1 Typical and maximum operating conditions

Symbol	Parameter	Conditions		Min	Typ	Max	Unit
V_{CC}	Supply voltage	No galvanic isolation		9[14]	32	35	V
		Galvanic isolation		11[14]	32	35	V
I_{CC}	Supply current	No galvanic isolation	V_{CC} = 9V	12		14	mA
		No galvanic isolation	V_{CC} = 32V	3		5	mA
		Galvanic isolation	V_{CC} = 11V	30		40	mA
		Galvanic isolation	V_{CC} = 32V	7		12	mA
ϑ_{op}	Operating temperature	No switchable termination, No galvanic isolation included		-40		125	°C
		Switchable termination included		-40		100	°C
		Galvanic isolation included		-40		105	°C
$V_{CANH/L}$	Voltage on pins CANH, CANL	With respect to GND/GND_{CAN}		-32		32	V

7.2 General wiring restrictions

Please keep in mind the following suggestions in order to design a fully functional CAN network.

1) Device must be connected to ground potential at the pressure port.
2) Termination is necessary at both CAN bus ends.
3) The bus wires must be of twisted pair type.
4) Keep signal wires separated from cables with voltages > 60V.
5) Signal wire diameter of 1.5mm² must not be exceeded per core.
6) Avoid proximity to large electrical machinery or use shielded cables.

7.3 Miswiring protection

The device is protected against miswiring at voltages as high as 32 volts.

[14] Absolute minimum voltage. Pay special attention to wiring resistances when designing the application network.

8 INITIAL OPERATION AND APPLICATION EXAMPLES

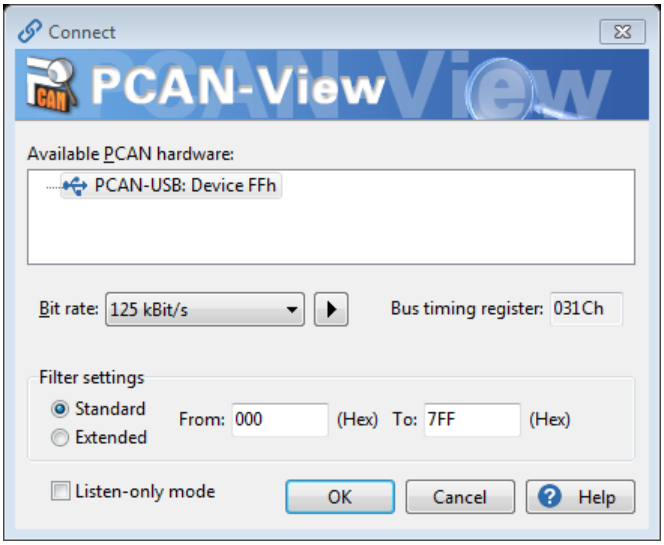
8.1 General

The demonstration shown in the following chapter uses the combination of *PCAN-USB* Dongle and the free software tool *PCAN-View*. Both are supplied by *Peak System Technik GmbH* (http://www.peak-system.com).

8.2 Connection check and usage of the PCAN-View software

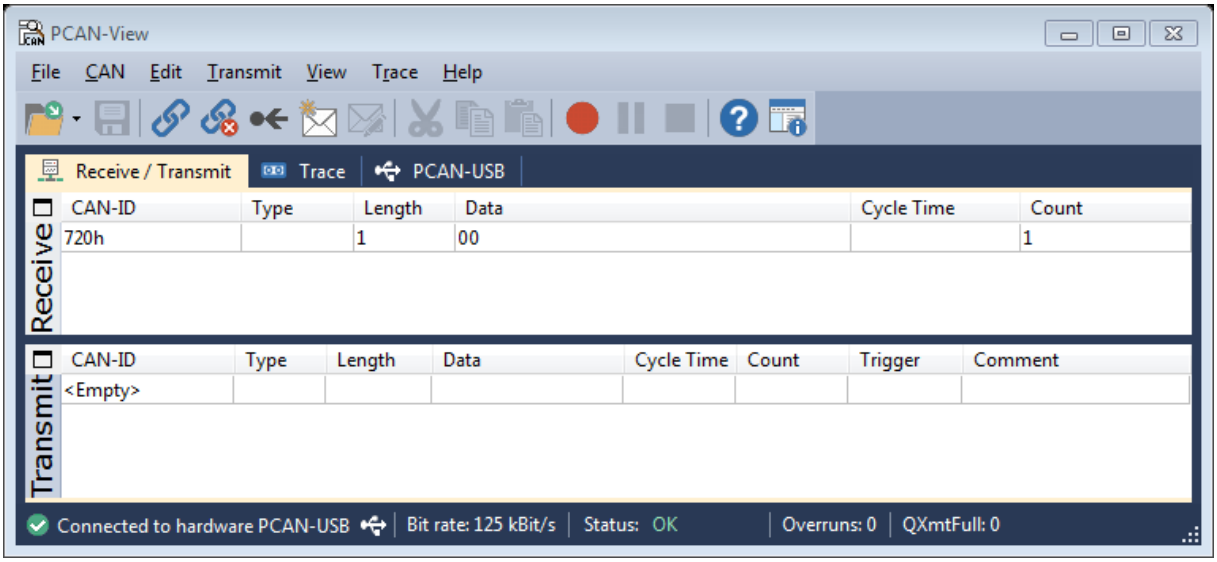
Initial configuration steps

When starting *PCAN-View*, the software lists all PCAN devices attached to the PC. After choosing the device, the *bit rate* has to be chosen. The bit rate depends on the settings made when ordering the *SMC pressure/temperature transducer*.



Checking the connection to *SMC pressure/temperature transducer*

When powering up, each CANopen device provides a so-called *boot-up message*. Its CAN-ID calculates as follows: $CAN\text{-}ID_{Boot\text{-}up}$ = 0x700 + Node-ID.

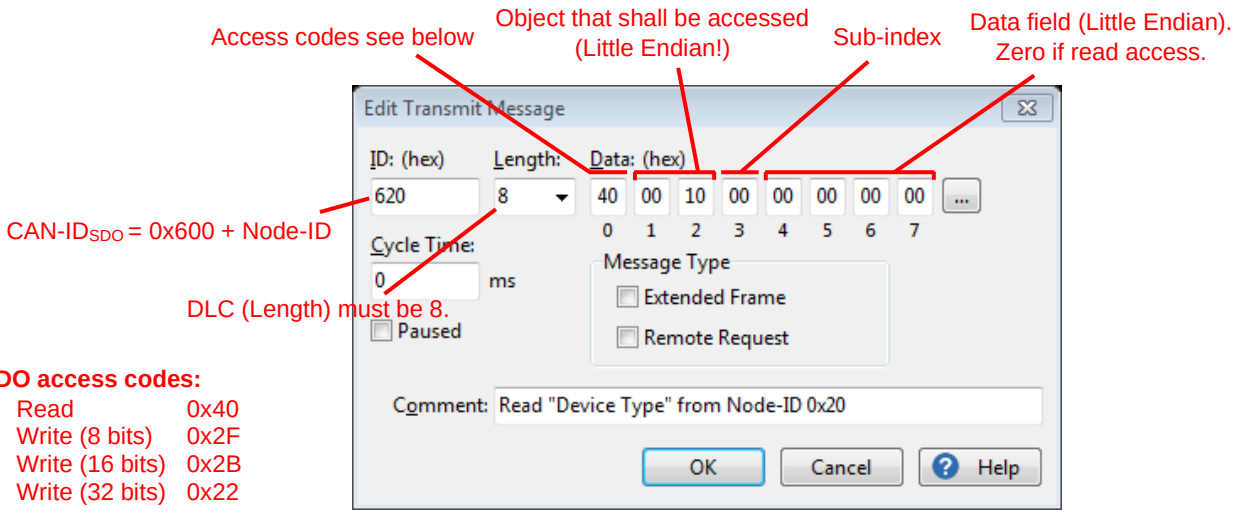


The picture above shows the *boot-up* message sent out by an *SMC pressure/temperature transducer*. Its Node-ID (in this case) obviously is 0x20.

Message transmission (here: SDO read access)

New *transmit messages* can be created by navigating to the menu "*Transmit*" → "*New Message*".

A window shows up enabling the user to configure a new message. The screenshot below contains exemplary settings to read a value via SDO from Node-ID 0x20. For more detailed information on SDO, refer to section 4.5.



SDO access codes:

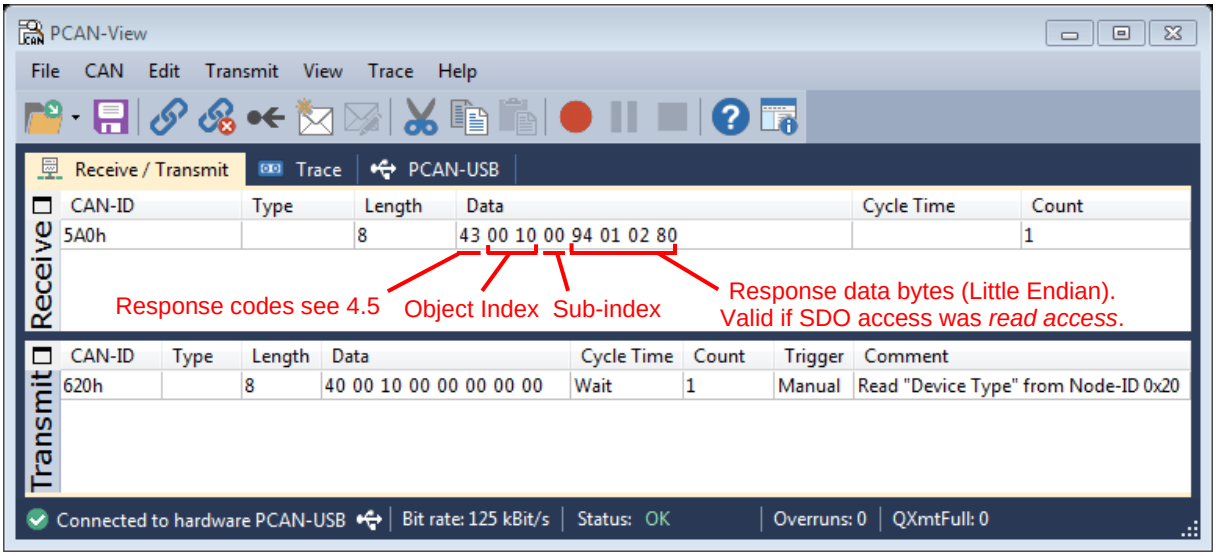
- Read 0x40
- Write (8 bits) 0x2F
- Write (16 bits) 0x2B
- Write (32 bits) 0x22

After pressing *OK*, a new entry is inserted in the *lower window area* ("Transmit"). Selecting the entry and hitting *Space* triggers the message transmission – the field *Count* will increase.

Interpreting SDO responses

After transmitting a SDO message, the *SMC pressure/temperature transducer* answers with an SDO response. The picture below shows an exemplary SDO response after transmitting a *SDO read access message*.

The *SDO response CAN-ID* calculates as follows: CAN-ID$_{SDO,response}$ = 0x580 + Node-ID.



The response code 0x43 (see 4.5) means that the *SDO read access* was successful and that there are *four data bytes for interpretation*. These four bytes usually must be interpreted as *Little Endian* – which means that the outputted value is 0x80020194.

Its data type depends on the accessed object, in this case (object 0x1000, sub-index 0x00; refer to section 6.1.1) the data type is *uint32*.

8.3 Configuration of the Node-ID

General

The Node-ID identifies a CANopen device within an application network and therefore must be unique. Valid Node-IDs are of data type *uint8* and range from 1 to 127 (0x01 … 0x7F). *SMC pressure/temperature transducers* do support Node-ID configuration via SDO and LSS.

Setting via SDO (should be preferred over LSS)

When configuring the Node-ID, it is most likely preferred that the COB-IDs (SYNC, EMCY, TIME, TPDO) do change as well. If the configuration takes place using SDO, all supported COB-IDs *can* be adapted to the new Node-ID (for more options, refer to section 6.3.7).

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Old Node-ID	8	2F	01	40	02	**XX**	00	00	00

*SDO message for changing the Node-ID (where **XX** is the new Node-ID)*

In response, the *SMC pressure/temperature transducer* answers with a *boot-up message* using the new Node-ID.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x700 + New Node-ID	1	00							

Response message after changing Node-ID via SDO

Be aware that the changes are only stored in RAM. For writing them to EEPROM, the procedure described in 8.9 must be performed.

Setting via LSS

Using the LSS protocol for changing the Node-ID is the standard method the CANopen standard proposes. The disadvantage when using LSS is, that none of the COB-IDs (SYNC, EMCY, TIME, TPDO) will change according to the new Node-ID – each of them needs to be configured manually (which is possible). This is what most likely makes *SDO the preferred method* of configuring the Node-ID.

When using the *following method* for configuring the Node-ID, it is absolutely necessary to use a 1:1 wiring to the CANopen device. Otherwise, multiple CANopen devices might end up having the same Node-ID. The configuration consists of multiple steps:

1) At first, the CANopen device has to be transferred to LSS mode. The therefore used command is "*Switch Mode Global*".

CAN-ID (hex)	DLC	Data bytes (hex)							
0x7E5	8	04	01	00	00	00	00	00	00

Step 1: Transmission of "Switch Mode Global"

2) Then the Node-ID will be inquired using the command "*Inquire Node-ID*" (optional).

CAN-ID (hex)	DLC	Data bytes (hex)							
0x7E5	8	5E	00	00	00	00	00	00	00

Step 2: Transmission of "Inquire Node-ID"

The device will answer with a message containing its Node-ID. In case of no response, the *current bit rate settings* might be incorrect.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x7E4	8	5E	**XX**	00	00	00	00	00	00

*Step 2: Reception of the Node-ID (where **XX** is the current Node-ID)*

3) Now, the Node-ID can be configured. This will be done using the command "*Configure Node-ID*".

CAN-ID (hex)	DLC	Data bytes (hex)							
0x7E5	8	11	**XX**	00	00	00	00	00	00

*Step 3: Transmission of "Configure Node-ID" (where **XX** is the new Node-ID)*

The device will now answer with a message containing potential *error codes*.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x7E4	8	11	**EE**	00	00	00	00	00	00

*Step 3: Response message (where **EE** is the error code; 00 means "no error")*

4) With the next message the changes will to be written to EEPROM. It is strongly recommended to wait 500ms after this command to avoid memory corruptions.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x7E5	8	17	00	00	00	00	00	00	00

Step 4: Transmission of "Store Configuration"

The device will now answer with a message containing potential *error codes*.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x7E4	8	17	**EE**	00	00	00	00	00	00

*Step 4: Response message (where **EE** is the error code; 00 means "no error")*

5) The new settings are not active, yet. There are two ways to activate them. The user may choose one of them.
   a. *Unpower and repower* the device. The new configuration will be loaded at boot-up.
   b. Perform a *soft-reboot* of the device firmware. This is done using the *NMT command* "*Node Reset*".

CAN-ID (hex)	DLC	Data bytes (hex)							
0	2	81	**XX**						

*Step 5.b: Perform a "soft-reboot" of the device firmware to enable configuration. **XX** is the "old" Node-ID of the device.*

8.4 Configuration of the bit rate

General

Adjusting the *bit rate* can be achieved using LSS. Section 4.12 provides a list of the available *bit rates* including their corresponding *LSS-codes*.

Setting via LSS

When using the *following method* for configuring the *bit rate*, it is absolutely necessary to use a 1:1 wiring to the CANopen device. The configuration consists of multiple steps:

1) At first, the CANopen device has to be transferred to LSS mode. The therefore used command is "*Switch Mode Global*".

CAN-ID (hex)	DLC	Data bytes (hex)							
0x7E5	8	04	01	00	00	00	00	00	00

Step 1: Transmission of "Switch Mode Global"

2) Then the Node-ID will be inquired using the command "*Inquire Node-ID*" (optional).

CAN-ID (hex)	DLC	Data bytes (hex)							
0x7E5	8	5E	00	00	00	00	00	00	00

Step 2: Transmission of "Inquire Node-ID"

The device will answer with a message containing its Node-ID. In case of no response, the *current bit rate settings* might be incorrect.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x7E4	8	5E	**XX**	00	00	00	00	00	00

*Step 2: Reception of the Node-ID (where **XX** is the current Node-ID)*

3) Now, the *bit rate* can be configured. This will be done using the command "*Configure Bit Timing Parameters*".

CAN-ID (hex)	DLC	Data bytes (hex)							
0x7E5	8	13	**TT**	**BB**	00	00	00	00	00

*Step 3: Transmission of "Configure Bit Timing Parameters" (where **TT** is the bit table selector and **BB** is the bit rate code). **TT** is typically zero; for **BB** refer to 4.12, Table 1.*

The device will now answer with a message containing potential *error codes*.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x7E4	8	13	**EE**	00	00	00	00	00	00

*Step 3: Response message (where **EE** is the error code; see below)*

There are several *error code* definitions:

- *0x00* … No error occurred. Bit rate accepted.
- *0x01* … Bit rate not supported.
- *0xFF* … Implementation specific error occurred. At this moment, not used yet.

4) With the next message the changes will to be written to EEPROM. It is strongly recommended to wait 500ms after this command to avoid memory corruptions.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x7E5	8	17	00	00	00	00	00	00	00

Step 4: Transmission of "Store Configuration"

The device will now answer with a message containing potential *error codes*.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x7E4	8	17	**EE**	00	00	00	00	00	00

*Step 4: Response message (where **EE** is the error code; 00 means "no error")*

5) The new settings are not active, yet. There are two ways to activate them. The user may choose one of them.
   a. *Unpower and repower* the device. The new configuration will be loaded at boot-up.
   b. Perform a *soft-reboot* of the device firmware. This is done using the *NMT command* "*Node Reset*".

CAN-ID (hex)	DLC	Data bytes (hex)							
0	2	81	**XX**						

*Step 5.b: Perform a "soft-reboot" of the device firmware to enable configuration. **XX** is the Node-ID of the device.*

8.5 Enabling/disabling the automatic transitioning to Operational Mode (Autostart)

After finishing boot-up, a CANopen device typically settles its *state machine* in Preoperational Mode. In this state, the PDO service is not enabled (refer to 4.3). To enable/disable the *automatic transitioning* to Operational Mode while boot-up, the object 0x1F80 (see 6.1.20) shall be used.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	23	80	1F	00	**XX**	00	00	00

*SDO message for changing the start-up behaviour (where **XX** is the desired behaviour; see below)*

For *enabling the automatic transitioning* to Operational Mode, **XX** must be substituted with the hexadecimal value 0x08. For *disabling the feature* instead, the value 0x00 must be entered.

In return, the *SMC pressure/temperature transducer* answers with the response below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x580 + Node-ID	8	60	80	1F	00	00	00	00	00

SDO response after changing the start-up behaviour.

Be aware that the changes are only stored in RAM. For writing them to EEPROM, the procedure described in 8.9 must be performed.

8.6 Reading and interpreting measurement values using SDO

General

Measurement values can be obtained via PDO and SDO. The following subsections show how to read and interpret measurements using SDO access.

For interpreting the values, the ***Measurements Endianness*** is of high importance. The *Endianness* decides how *multi-byte-values* will be transferred:

- Least significant byte first (*CANopen standard*; known as *Little Endian or Intel Format*),
- Most significant byte first (known as *Big Endian* or *Motorola Format*).

The *Measurements Endianness* of *SMC CANopen transducers* can be chosen by the user (refer to section 6.3.6). Because most likely *Little Endian* will be preferred over *Big Endian*, the following examples describe how to interpret measurements arranged as *Little Endian*.

Each ***digital value range*** (which is typically $[0 \dots 20000]$) corresponds to its ***physical measuring range***. In case of a $5\ bar$ *SMC pressure transducer*, the digital value range corresponds to $[0 \dots 5\ bar]$; in case of a *SMC temperature transducer*, the digital value range corresponds to its nominal temperature range.

SMC pressure transducers additionally provide a coarse temperature value (± 5K), accessible as *secondary measuring channel*. The secondary measuring channel's digital range does always correspond to $[-55 \dots 125\ °C]$.

Reading and interpreting *float32* Process Values (*Primary Measuring Channel*)

When reading *Process Values*, their ***digital value range*** must be known. This range is usually set to $[0 \dots 20000]$, but it can differ according to the settings made to *Input Scaling* (refer to section 6.2.6). For reading the *float32* Process Value, the object 0x6130 must be accessed as shown below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	40	30	61	01	00	00	00	00

SDO message for reading the Primary Measuring Channel Process Value (float32)

In return, the *SMC pressure/temperature transducer* answers with the response below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x580 + Node-ID	8	43	30	61	01	**DD**	**CC**	**BB**	**AA**

SDO response after requesting the Primary Measuring Channel Process Value.

The last four bytes of the SDO response contain the recent measurement, typically arranged as *Little Endian*. This means that the bytes need to be assembled as follows: **AA BB CC DD**.

Afterwards, the value can be decoded according to the IEEE 754 standard. For manual evaluation, the user should use an online tool (e.g. [7]) in order to convert the readouts to human-readable numbers.

Reading and interpreting *int16/int32* Process Values (*Primary Measuring Channel*)

When reading *Process Values*, their ***digital value range*** must be known. This range is usually set to $[0 \dots 20000]$, but it can differ according to the settings made to *Input Scaling* (refer to section 6.2.6). Additionally, the *Decimal Digits Value* (object 0x6132; refer to 6.2.4) is important. This value contains the places the comma needs to be shifted; it is typically set to zero, which means there is no shifting necessary.

For reading the *int16/int32* Process Value, the object 0x7130/0x9130 must be accessed as shown below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	40	30	91	01	00	00	00	00

SDO message for reading the Primary Measuring Channel Process Value (int32)

In return, the *SMC pressure/temperature transducer* answers with the response below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x580 + Node-ID	8	43	30	91	01	**DD**	**CC**	**BB**	**AA**

SDO response after requesting the Primary Measuring Channel Process Value.

The last four bytes of the SDO response contain the recent measurement, typically arranged as *Little Endian*. This means that the bytes need to be assembled as follows: **CC DD** (int16) or **AA BB CC DD** (int32).

Be aware that all int16/int32 readings are signed. Negative numbers will be outputted using the *two's complement*. Thus, for example, 0xFFF3 (*int16*) represents the decimal number -13.

In order to avoid potential negative output values, the *SMC pressure/temperature transducer* features the ability to clamp *Process Values* to a minimum and a maximum value. For further information, please refer to 6.3.4.

Reading and interpreting *int16* Field Values

When reading *Field Values*, the digital value range is always $[0 \dots 20000]$; it cannot be altered by the user. In addition, neither scaling, clamping nor comma-shifting will be applied to them. The *Field Values* are signed; their data size and type is *int16*.

For reading the *int16* Field Value, the object 0x7100 must be accessed as shown below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	40	00	71	01	00	00	00	00

SDO message for reading the Primary Measuring Channel Field Value (int16)

In return, the *SMC pressure/temperature transducer* answers with the response below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x580 + Node-ID	8	43	00	71	01	**BB**	**AA**	00	00

SDO response after requesting the Primary Measuring Channel Field Value.

The SDO response contains the recent measurement, typically arranged as *Little Endian*. This means that the bytes need to be assembled as follows: **AA BB** (int16).

Be aware that all int16 readings are signed. Negative numbers are possible and will be outputted using the *two's complement*. Thus, for example, 0xFFF3 (*int16*) represents the decimal number -13.

Pressure transducers only: Reading and interpreting *float32* Process Values (*Secondary Channel*)
As previously described, *SMC pressure transducers* allow the user to obtain coarse temperature measurements (± 5K). These measurements are available on the *Secondary Measuring Channel* and can be read as *int16, int32* and *float32* (Process Value) or *int16* (Field Value).

For reading the *float32* Process Value, the object 0x6130 must be accessed as shown below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	40	30	61	02	00	00	00	00

SDO message for reading the Secondary Measuring Channel Process Value (float32)

In return, the *SMC pressure/temperature transducer* answers with the response below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x580 + Node-ID	8	43	30	61	02	**DD**	**CC**	**BB**	**AA**

SDO response after requesting the Secondary Measuring Channel Process Value.

The last four bytes of the SDO response contain the recent measurement, typically arranged as *Little Endian*. This means that the bytes need to be assembled as follows: **AA BB CC DD**.

Afterwards, the value can be decoded according to the IEEE-754 standard. For manual evaluation, the user should use an online tool (e.g. [7]) in order to convert the readouts to human-readable numbers.

The measurements' *digital value range* does correspond to the following temperature range: $[-55\ °C \dots 125\ °C]$. Provided that the *digital value range* is $[0 \dots 20000]$, the temperature can be calculated according to the following formula:

$$\vartheta[°C] = \left(readout \cdot \frac{180\ °C}{20000}\right) - 55\ °C$$

Otherwise, if no further calculating is desired, the user may manipulate the *Input Scaling* (refer to sections 6.2.6 and 8.7) to any value range. In turn, the transducer will do all necessary linear transformations.

8.7 Configuration of the measurements' scaling (*Process Values* only)

Each measuring channels' *Process Values* are (in opposite to the *Field Values*) independently linearly scalable: Their typical value range $[0 \dots 20000]$ may be overridden with a user-defined value range; all the necessary linear transformations will be handled by the *SMC pressure/temperature transducer*. The following sub-sections contain two example scenarios.

Be aware that all changes are only stored in RAM. For writing them to EEPROM, additionally the procedure described in 8.9 must be performed.

Example 1: Manipulating the Primary Measuring Channel's *Process Value* range

Imagine using a $[0 \dots 5000\ PSI]$ *SMC pressure transducer*. For obtaining the recent *Process Value* there are two options:

- Reading the *Process Value*, which lies in the typical range $[0 \dots 20000]$ and ***manually transitioning*** it to its *physical unit* by applying the following formula: $p[PSI] = \frac{value}{20000} \cdot 5000 = \frac{1}{4} \cdot value$
- Once assigning the desired *Process Value* range $[0 \dots 5000]$ to the transducer so that ***no formula needs to be applied*** to future *Process Value* readings.

The following example shows how to apply any desired *Process Value* range to the *Primary Measuring Channel*. At first, the desired range's minimum and maximum limits need to be converted to *hexadecimal float32* values (e.g. using the online tool [7]):

$$0 = 0x00\ 00\ 00\ 00$$
$$5000 = 0x45\ 9C\ 40\ 00$$

Now, these values will be assigned to the objects "*Input Scaling – Process Value 1/2*" (0x6121 and 0x6123; sub-index 2). Be careful as the values need to be transferred as ***Little Endian***. The following two messages will be used to do the transfer.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	22	21	61	01	**00**	**00**	**00**	**00**

*SDO message for writing the Primary Channel's **lower scaling limit** (float32)*

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	22	23	61	01	**00**	**40**	**9C**	**45**

*SDO message for writing the Primary Channel's **upper scaling limit** (float32)*

In return, the *SMC pressure/temperature transducer* acknowledges each writing access with the response below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x580 + Node-ID	8	60	21/23	61	02	00	00	00	00

SDO response after writing the Secondary Measuring Channel's scaling limits.

Example 2: Manipulating the Secondary Channel's *Process Value* range (*SMC pressure transducers* only)
As previously described, *SMC pressure transducers* allow the user to obtain coarse temperature measurements (± 5K). These measurements are available on the *Secondary Measuring Channel*; their typical value range $[0 \dots 20000]$ corresponds to $[-55\ °C \dots 125\ °C]$.

Wouldn't it be useful if the output values' range was $[-55 \dots 125]$? To make the pressure transducer automatically transfer its measurements to any range, at first, the desired minimum and maximum limits need to be converted to *hexadecimal float32* values (e.g. using the online tool [7]):

$$-55 = 0xC2\ 5C\ 00\ 00$$
$$125 = 0x42\ FA\ 00\ 00$$

Now, these values will be assigned to the objects "*Input Scaling – Process Value 1/2*" (0x6121 and 0x6123; sub-index 2). Be careful as the values need to be transferred as ***Little Endian***. The following two messages will be used to do the transfer.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	22	21	61	02	**00**	**00**	**5C**	**C2**

*SDO message for writing the Secondary Channel's **lower scaling limit** (float32)*

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	22	23	61	02	**00**	**00**	**FA**	**42**

*SDO message for writing the Secondary Channel's **upper scaling limit** (float32)*

In return, the *SMC pressure transducer* acknowledges each writing access with the response below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x580 + Node-ID	8	60	21/23	61	02	00	00	00	00

SDO response after writing the Secondary Measuring Channel's scaling limits.

8.8 Configuring transmission parameters of measurement PDOs

The *SMC pressure/temperature transducers* feature three kinds of triggers to initiate TPDO transmission:

- Milliseconds-based timer,
- Every nth SYNC message reception,
- Measuring events (Interrupt Delta, Interrupt Limits).

The sub-sections below show how to configure each of the transmission triggers.

Milliseconds-based timer

Configuring PDO transmission parameters to "timer-driven" consists of two steps: Setting the *transmission type* to 0xFE and assigning the desired milliseconds value to the *event timer* register. The next CAN messages contain all necessary SDO accesses in order to configure TPDO1 to a cyclic, 100ms-based, transmission.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	22	00	18	02	**FE**	00	00	00

*SDO message for setting the TPDO1 **transmission type** to 0xFE ("timer-driven")*

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	22	00	18	05	**64**	**00**	00	00

*SDO message for setting the TPDO1 two-byte **event timer** to 100 (0x64).*

In return, the *SMC pressure transducer* acknowledges each writing access with the response below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x580 + Node-ID	8	60	00	18	02/05	00	00	00	00

SDO response to setting up TPDO1 transmission parameters.

Every nth SYNC message reception

In order to configure PDO transmission to "SYNC driven", its *transmission type* must be assigned to a value between 1 and 240 (0x1 … 0xF0). The CAN message below contains the necessary SDO access in order to configure TPDO1 to be sent out every 10th SYNC message.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	22	00	18	02	**0A**	00	00	00

*SDO message for setting the TPDO1 **transmission type** to 10 (0x0A; "SYNC driven")*

In return, the *SMC pressure transducer* acknowledges the writing access with the response below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x580 + Node-ID	8	60	00	18	02	00	00	00	00

SDO response to setting up TPDO1 transmission parameters.

Measuring events

Configuring PDO transmission parameters to "measuring event driven" consists of three steps: Setting the *transmission type* to 0xFF, disabling timer-based transmission and assigning the desired configuration to the *Process Value Interrupt* registers (refer to sections 6.2.8 and 5).

The CAN messages below demonstrate how to configure TPDO1 to *measuring event driven* transmission, triggered as soon as the ***Process Value*** *(Primary Measuring Channel)* exceeds the range $[1495.2 \dots 5010.9]$.

Therefore, at first, the range limits need to be converted to *hexadecimal float32* (e.g. using the online tool [7]):

$$1495.2 = 0x44\ BA\ E6\ 66$$
$$5010.9 = 0x45\ 9C\ 97\ 33$$

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	22	00	18	02	**FF**	00	00	00

*SDO message for setting the TPDO1 **transmission type** to 0xFF ("measuring event driven")*

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	22	00	18	05	**00**	**00**	00	00

*SDO message for disabling the TPDO1 **event timer**, which still might be enabled*

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	22	34	61	01	**66**	**E6**	**BA**	**44**

*SDO message for writing the **Interrupt Lower Limit** (Primary Measuring Channel)*

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	22	35	61	01	**33**	**97**	**9C**	**45**

*SDO message for writing the **Interrupt Upper Limit** (Primary Measuring Channel)*

Now, each time the *Process Value* leaves the specified range, TPDO1 automatically will be transmitted. Be aware of the 1%-hysteresis for re-entering the range; for more information, refer to section 6.2.9.

8.9 Storing configuration changes to EEPROM

When changing values via SDO, the changes will not automatically be written to EEPROM. It is strongly recommended to wait 500ms after this command to avoid memory corruptions. To do so, the following SDO message must be sent to the *SMC pressure/temperature transducer*:

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	22	10	10	01	73	61	76	65

SDO message for storing the configuration to EEPROM

In return, the *transducer* acknowledges the procedure with the response below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x580 + Node-ID	8	60	10	10	01	00	00	00	00

SDO response to storing the parameter set.

8.10 Resetting the transducer to factory settings

For restoring the factory settings, the message below must be sent to the *SMC pressure/temperature transducer*. Once restored, the parameters will automatically be assigned to the boot-up parameters section; no further storing is necessary. It is strongly recommended to wait 500ms after this command to avoid memory corruptions.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x600 + Node-ID	8	22	11	10	01	6C	6F	61	64

SDO message for restoring the factory configuration

In return, the *transducer* acknowledges the procedure with the response below.

CAN-ID (hex)	DLC	Data bytes (hex)							
0x580 + Node-ID	8	60	11	10	01	00	00	00	00

SDO response to storing the parameter set.