CAN-CBX-AI814

8 A/D-Converter-Inputs, 14 Bit

Manual

to Product C.3020.02

esd electronic system design gmbh
Vahrenwalder Str. 207 • 30165 Hannover • Germany
http://www.esd.eu
Phone: +49 (0) 511 3 72 98-0 • Fax: +49 (0) 511 3 72 98-68

N O T E

The information in this document has been carefully checked and is believed to be entirely reliable. **esd** makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors that may appear in this document. In particular descriptions and technical data specified in this document may not be constituted to be guaranteed product features in any legal sense.

esd reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance or design.

esd electronic system design gmbh
Vahrenwalder Str. 207
30165 Hannover
Germany

Phone: +49-511-372 98-0
Fax: +49-511-372 98-68
E-mail: info@esd.eu
Internet: www.esd.eu

Trademark Notices

CiA® and CANopen® are registered community trademarks of CAN in Automation e.V.
All other trademarks, product names, company names or company logos used in this manual are reserved by their respective owners.

Document-File:	I:\Texte\Doku\MANUALS\CAN\CBX\AI814\Englisch\CAN-CBX-AI814_Manual_en_15.wpd
Date of print:	2014-07-18

PCB version:	from CAN-CBX-AI814 Rev. 1.0
Firmware version:	from Rev. 2.6

Changes in the chapters

The changes in the document listed below affect changes in the hardware and firmware as well as changes in the description of facts only.

Chapter	Chapter in previous version	Changes versus previous version
-	-	Safety Instructions and Conformity
2.3	2.3	Description of electrical isolation changed
2.4	2.4	Accuracy of the analog inputs inserted
	2.6	Order information moved and revised
3.4	3.4	Chapter updated
4.	4. and 5.	Chapters combined and revised
4.4		Chapter 'Conductor Connection / Conductor Cross Section' inserted
5.	6.	Chapter moved and updated
6.	7.	Chapter moved and updated
7.	8.	Chapter moved, note to installation and wiring inserted
8.	9.	Chapter 'Software' renamed to 'CANopen-Firmware', general part (chapter 8.1 -8.4) updated and restructured
8.6	9.5	Description of the parameters inserted
8.8	-	Chapter 'Used Names and Abbreviations' inserted
8.9.1	9.7	Product-specific properties of CAN-CBX-AI814 described in table, new objects $1F80_h$, $1F91_h$
8.9.2 - 8.9.21	9.71-9.7.20	Chapter revised, description of the product-specific properties removed to table in'Overview of Communication Profile-Specific Objects with Product-Specific Properties (chapter 8.9.1), objects 1010_h and 1011_h inserted
8.9.22	-	New chapter 'NMT Startup ($1F80_h$)'
8.9.23	-	New chapter 'Self Starting Nodes Timing Parameters ($1F91_h$)'
9.	10.	Chapter revised
10.	-	EU Declaration of Conformity updated
11.	2.6	Chapter moved

Technical details are subject to change without further notice.

Safety Instructions

- When working with CAN-CBX modules follow the instructions below and read the manual carefully to protect yourself and the CAN-CBX module from damage.
- The permitted operating position is specified as shown (Fig. 7). Other operating positions are not allowed.
- Do not open the housing of the CAN-CBX module
- Never let liquids get inside the CAN-CBX module. Otherwise, electric shocks or short circuits may result.
- Protect the CAN-CBX module from dust, moisture and steam.
- Protect the CAN-CBX module from shocks and vibrations.
- The CAN-CBX module may become warm during normal use. Always allow adequate ventilation around the CAN-CBX module and use care when handling.
- Do not operate the CAN-CBX module adjacent to heat sources and do not expose it to unnecessary thermal radiation. Ensure an ambient temperature as specified in the technical data.
- Do not use damaged or defective cables to connect the CAN-CBX module and follow the CAN wiring hints in chapter: 'Correct Wiring of Electrically Isolated CAN Networks' .
- In case of damages to the device, which might affect safety, appropriate and immediate measures must be taken, that exclude an endangerment of persons and domestic animals, or property.
- Current circuits which are connected to the device have to be sufficiently protected against hazardous voltage (SELV according to EN 60950-1).
- The CAN-CBX module may only be driven by power supply current circuits, that are contact protected. A power supply, that provides a safety extra-low voltage (SELV or PELV) according to EN 60950-1, complies with this conditions.

Qualified Personal

This documentation is directed exclusively towards qualified personal in control and automation engineering.
The installation and commissioning of the product may only be carried out by qualified personal, which is authorized to put devices, systems and electric circuits into operation according to the applicable national standards of safety engineering.

Conformity

The CAN-CBX module is an industrial product and meets the demands of the EU regulations and EMC standards printed in the conformity declaration at the end of this manual.

Warning:	In a residential, commercial or light industrial environment the CBX-module may cause radio interferences in which case the user may be required to take adequate measures.
Note:	To ensure EU Conformity a cable with a maximum wire length of 3 m has to be used for the analog inputs.

Intended Use

The intended use of the CAN-CBX module is the operation as a CANopen module with analog inputs.
The esd guarantee does not cover damages which result from improper use, usage not in accordance with regulations or disregard of safety instructions and warnings.

- The CAN-CBX module is intended for indoor installation only.
- The operation of the CAN-CBX module in hazardous areas, or areas exposed to potentially explosive materials is not permitted.
- The operation of the CAN-CBX module for medical purposes is prohibited.

Service Note

The CAN-CBX module does not contain any parts that require maintenance by the user. The CAN-CBX module does not require any manual configuration of the hardware. Unauthorized intervention in the device voids warranty claims.

Disposal

Devices which have become defective in the long run have to be disposed in an appropriate way or have to be returned to the manufacturer for proper disposal. Please, make a contribution to environmental protection.

Contents

Typographical Conventions

Throughout this manual the following typographical conventions are used to distinguish technical terms.

Convention	Example
File and path names	`/dev/null or <stdio.h>`
Function names	***open()***
Programming constants	`NULL`
Programming data types	`uint32_t`
Variable names	*Count*

The following indicators are used to highlight noticeable descriptions.

Attention:
Warnings or cautions to tell you about operations which might have unwanted side effects.

Note:
Notes to point out something important or useful.

Number Representation
All numbers in this document are base 10 unless designated otherwise. For hexadecimal numbers $_h$ is appended . For example, 42 is represented as $2A_h$ in hexadecimal format.

1. Overview

1.1 Description of the Module

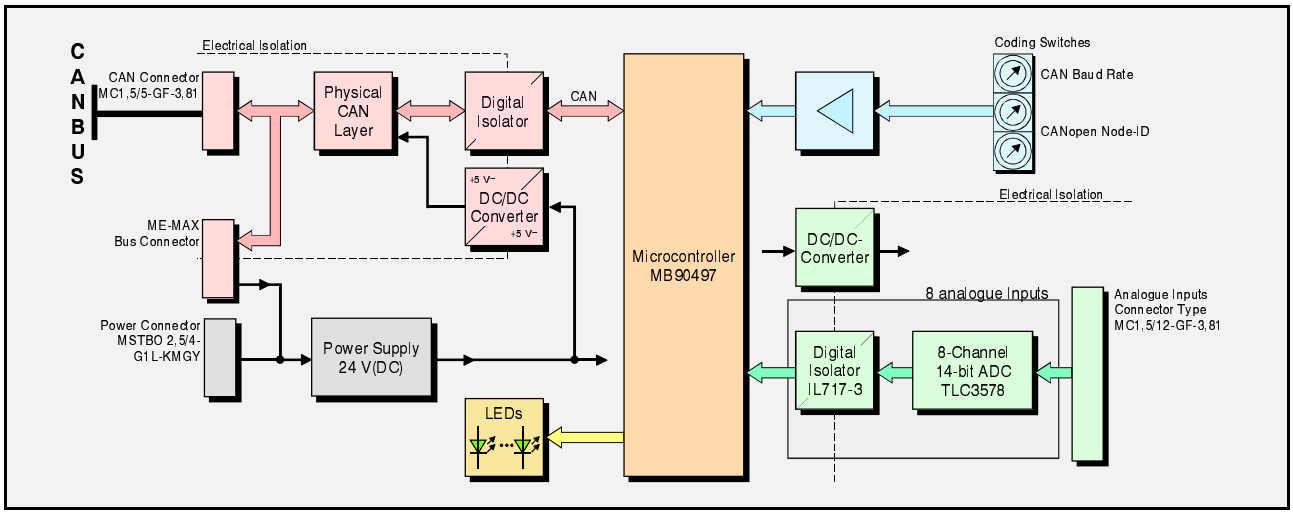


Fig. 1: Block circuit diagram of the CAN-CBX-AI814 module

The CAN-CBX-AI814 module is a CANopen module with eight A/D-converter inputs and InRailBus. It is equipped with a MB90F497 microcontroller. The firmware is held in the flash. Parameters are stored in a serial EEPROM.

The 14-bit A/D-converter, a TLC3578, converts the eight analog inputs.
The inputs are connected via a 12-pole plug with spring-cage connections. The analog inputs are electrically isolated by digital isolators.

The power supply voltage and the CAN-bus connection can be fed via the InRailBus connector, integrated in the top hat rail, or via separate plugs.

The ISO 11898 compliant CAN interface allows a maximum data transfer rate of 1 Mbit/s. The CAN interface is electrically isolated by a digital isolator and a DC/DC converter.

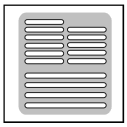
The CANopen node number and the CAN bit rate can be configured via three coding switches.

2. Technical Data

2.1 General technical Data

Power supply voltage	nominal voltage: 24 V/DC input voltage range: 24 V ±20% current consumption (24 V, 20 ˚C): typically: 40 mA maximum: 45 mA
Connectors	24V (4-pin printed circuit board connector with spring-cage connection, X400) - 24V-power supply voltage InRailBus (5-pin ME-MAX-TBUS-connector, Phoenix Contact, X101) - CAN interface and power supply voltage via InRailBus 1G ... 8P (12-pin printed circuit board connector, X500) - analog inputs CAN (5-pin printed circuit board connector, X400) - CAN interface Only for test and programming purposes (internal): X200 (6-pin printed circuit board connector connector)
Temperature range	-20 ˚C ... +70 ˚C ambient temperature
Humidity	max. 90%, non-condensing
Protection class	IP20
Pollution degree	maximum permissible according to DIN EN 61131-2: Pollution Degree 2
Housing	Plastic housing for carrier rail mounting NS35/7,5 DIN EN 60715
Dimensions	width: 2.2 cm, height: 11.2 cm, depth: 11.3 cm (including mounting rail fitting and connector projection)
Weight	approx. 125 g

Table 1: General technical data



2.2 CPU-Unit

CPU	16 bit µC MB90F497
RAM	2 Kbyte integrated
Flash	64 Kbyte integrated
EEPROM	minimum 256 byte

Table 2: Microcontroller

2.3 CAN Interface

Number	1
Connection	5-pin COMBICON with spring-cage connection or via Phoenix Contact TBUS-connector (InRailBus)
CAN Controller	MB90F497, ISO11898-1 (CAN 2.0) (CANopen software supports only 11-bit CAN identifier)
Electrical isolation of CAN interfaces against other units	via digital Isolator and DC/DC-converter
Physical layer CAN	physical layer according to ISO 11898-2, transfer rate programmable from 10 Kbit/s up to 1 Mbit/s
Bus termination	has to be set externally if required

Table 3: Data of the CAN interface

2.4 Analog Inputs

Number	8 A/D-converter channels
Converter-Type	TLC3578
Resolution	14 bit via A/D converter
Input voltage rage	±10.24 V
Accuracy input voltage ±9.2 V	error < 5 mV
Drift temperature range	±50 ppm/°C
Conversion time	≥ 100 µs / 8 channels simultaneously
Electrical isolation against other units	via dual digital isolator
Input impedance	typically 10 kΩ

Table 4: Data of analog inputs

2.5 Software Support

The firmware of the module supports CANopen® according to CiA ® CANopen specifications CiA 301 [1] and CiA DS-401 [2].

3. Hardware Installation

3.1 Connecting Diagram

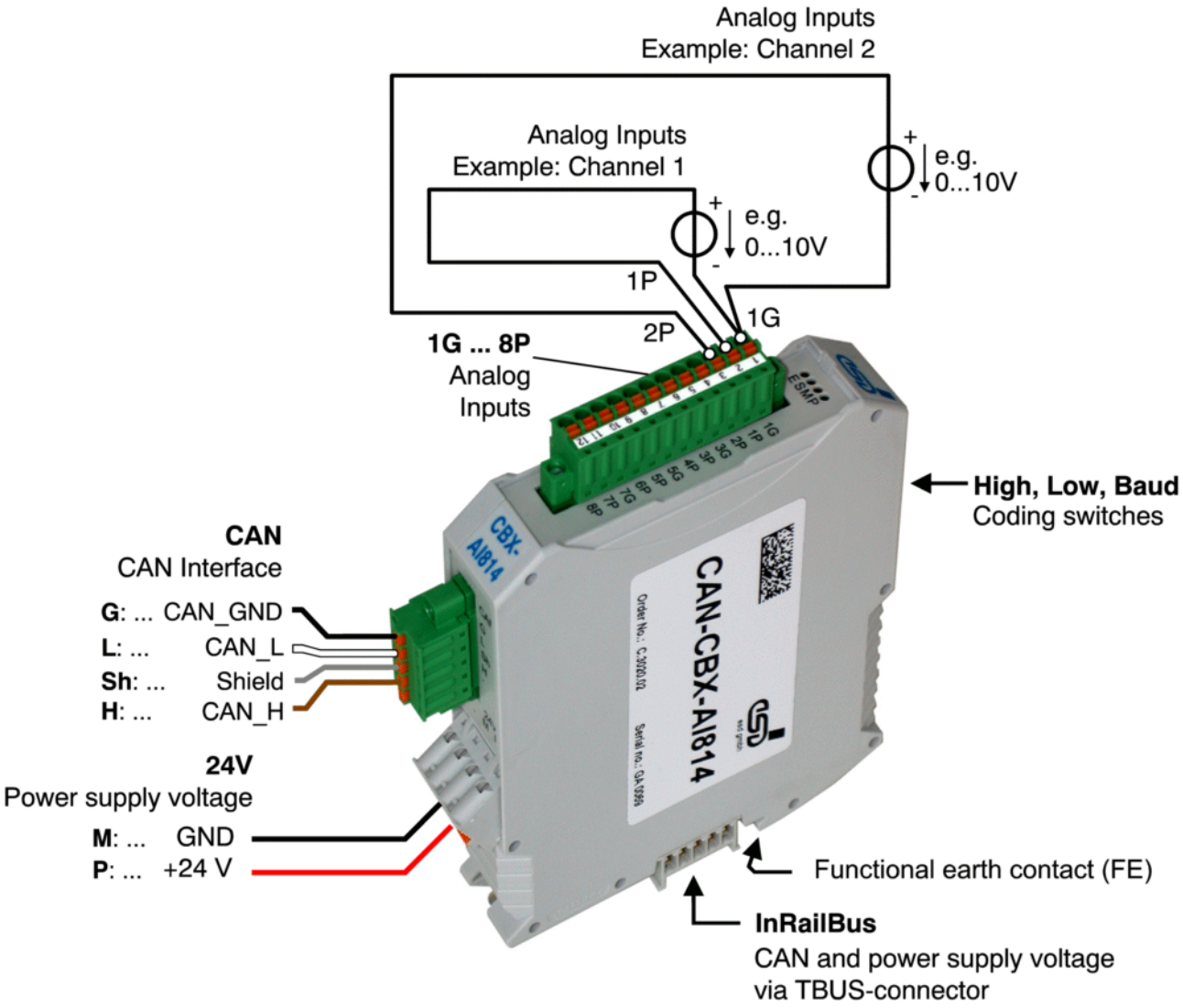


Fig. 2: Connections of the CAN-CBX-AI814 module

Note:
The connector pin assignment can be found on page 24 and following.
For conductor connection and conductor cross section see page 30.

3.2 LED Display

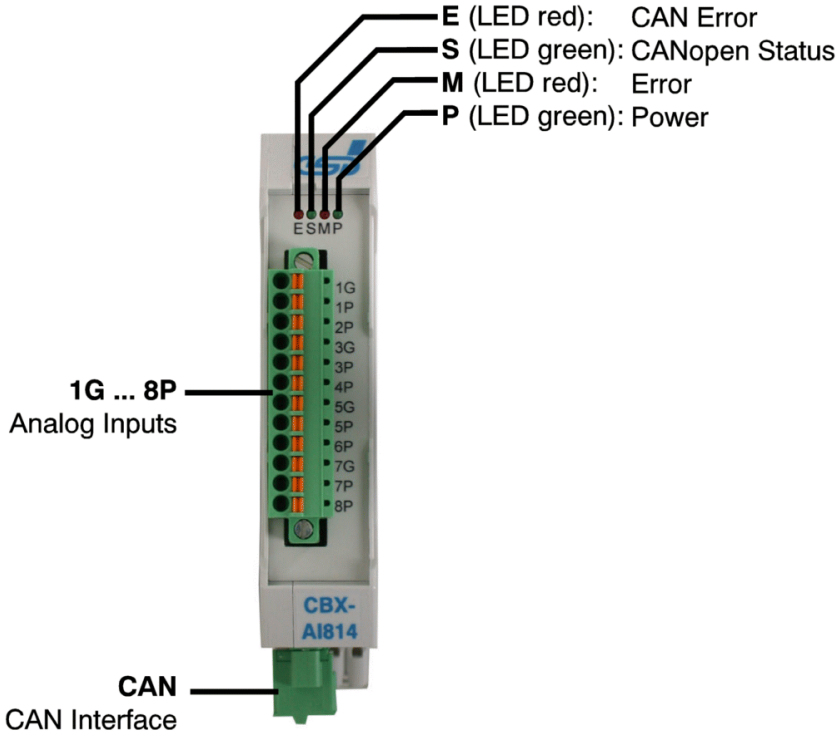


Fig. 3: Position of the LEDs in the front panel

The CAN-CBX-AI814 module is equipped with 4 status LEDs.
The terms of the indicator states of the LEDs are chosen in accordance with the terms recommended by the CiA [3]. The indicator states are described in the following chapters.

3.2.1 Indicator States

In principle there are 8 indicator states distinguished:

Indicator state	Display
on	LED constantly on
off	LED constantly off
blinking	LED blinking with a frequency of approx. 2.5 Hz
flickering	LED flickering with a frequency of approx. 10 Hz
1 flash	LED 200 ms on, 1400 ms off
2 flashes	LED 200 ms on, 200 ms off, 200 ms on 1000 ms off
3 flashes	LED 2x (200 ms on, 200 ms off) + 1x (200 ms on, 1000 ms off)
4 flashes	LED 3x (200 ms on, 200 ms off) + 1x (200 ms on, 1000 ms off)

Table 5: Indicator states

Note:
Red and green LEDs are strictly switched in phase opposition according to the CANopen Specification [3].
For certain indicator states viewing all LEDs together might lead to a misinterpretation of the indicator states of adjacent LEDs. It is therefore recommended to look at the indicator state of an LED individually, in covering the adjacent LEDs.

3.2.2 Operation of the CAN-Error LED

LED indication			Display function	
Label	Name	Colour	Indicator state	Description
E	CAN Error	red	off	no error
			1 flash	CAN controller is in *Error Active* state
			on	CAN controller state is *Bus Off* (or coding switch position ID-node > $7F_h$ when switching on; see 'Special Indicator States'on page 16)
			2 flashes	Heartbeat or Nodeguard error occurred. The LED automatically turns off, if Nodeguard/Heartbeat-messages are received again.

Table 6: Indicator states of the red CAN Error-LED

3.2.3 Operation of the CANopen-Status LED

LED indication			Display function	
Label	Name	Colour	Indicator state	Description
S	CANopen Status	green	blinking	*Pre-operational*
			on	*Operational*
			1 flash	*Stopped*
			3 flashes	Module is in bootloader mode, the power LED is off, (or coding switch position ID-node > $7F_h$ when switching on; see page 16)

Table 7: Indicator states of the CANopen Status-LED

3.2.4 Operation of the Error-LED

LED indication			Display function	
Label	Name	Colour	Indicator state	Description
M	Error	red	off	no error
			on	CAN Overrun Error The sample rate is set so high, that the firmware is not able to transmit all data on the CAN bus.
			2 flashes	Internal software error e.g.: - stored data have an invalid checksum therefore default values are loaded - internal watchdog has triggered - indicator state is continued until the module resets or an error occurs at the outputs.

Table 8: Indicator state of the Error-LED

3.2.5 Operation of the Power-LED

LED indication				Display function	
Label	Name	Colour	Component No.	Indicator state	Description
P	Power	green	200D	off	no power supply voltage; or the module is in Bootloader-Mode, this state is indicated by the CANopen status-LED (3 Flashes)
				on	power supply voltage is on and application software is running

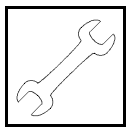
Table 9: Indicator state of the Power-LED

3.2.6 Special Indicator States

The special indicator state described in the following table is indicated by the CANopen-Status-LED and the CAN-Error-LED together:

LED indication	Description
CANopen-Status LED: 3 flashes CAN-Error LED: on	The coding switches for the Node-ID are set to an invalid ID-value, when switching on. The firmware application will not be started.

Table 10: Special Indicator States



3.2.7 Assignment of LED Labelling to Name in Schematic Diagram

Labelling on CAN-CBX-AI814	Name of the LED in Schematic Diagram*
E	LED200A
S	LED200B
M	LED200C
P	LED200D

* The Schematic Diagram is not part of this manual.

3.3 Coding Switches

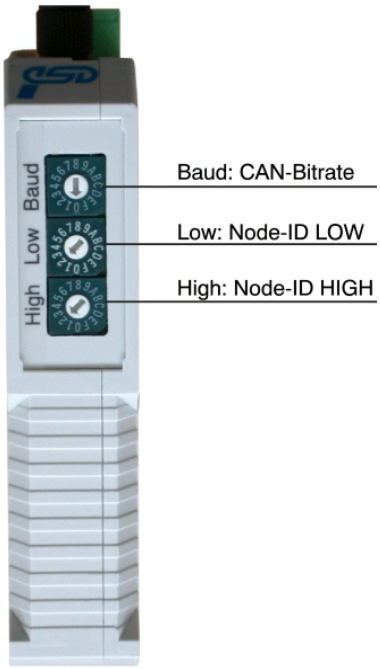


Fig. 4: Position of the coding switches

Attention:
At the moment the module is switched 'on', the state of the coding switches is determined. Changes of the settings therefore have to be made **before switching on** the module, because changes of the settings are not determined during operation.

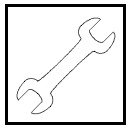
After a reset (e.g. NMT reset) the settings are read again.

3.3.1 Setting the Node-ID via Coding Switch

The address range of the CAN-CBX-module can be set *decimal* from 1 to 127 or *hexadecimal* from 01_h to $7F_h$.

The three higher-order bits (higher-order nibble) can be set with coding switch **HIGH**, the four lower-order bits can be set with coding switch **LOW**.

Note:
Avoid the following settings:
Setting the address range of the coding switches to values higher than $7F_h$ causes error messages, the red CAN-Error LED turns on.
If the coding switches are set to 00_h, the CAN-CBX-AI814 changes into Bootloader mode.



3.3.2 Setting the Baud Rate

The baud rate can be set with the coding switch **Baud**.

Values from 0_h to F_h can be set via the coding switch. The values of the baud rate can be taken from the following table:

Setting [Hex]	Bit rate [Kbit/s]
0	1000
1	$666,\overline{6}$
2	500
3	$333,\overline{3}$
4	250
5	166
6	125
7	100
8	$66,\overline{6}$
9	50
A	$33,\overline{3}$
B	20
C	12,5
D	10
E *1)	800
F *1)	$83,\overline{3}$ *2

*1) implemented since firmware version 2.03

Table 11: Index of the baud rate

3.3.3 Assignment of Coding-Switch Labelling to Name in Schematic Diagram

Labelling on the CAN-CBX-AI814	Name in the Schematic Diagram *2)
Baud	SW301
Low	SW300
High	SW302

*2) The Schematic Diagram is not part of this manual.

3.4 Installation of the Module Using InRailBus Connector

If the CAN bus signals and the power supply voltage shall be fed via the InRailBus, please proceed as follows:

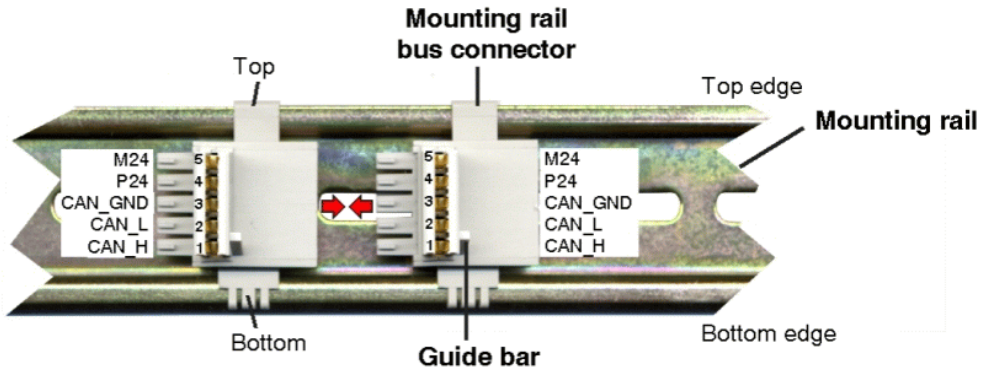


Figure 5: Mounting rail with bus connector

1. Position the InRailBus connector on the mounting rail and snap it onto the mounting rail using slight pressure. Plug the bus connectors together to contact the communication and power signals (in parallel with one). The bus connectors can be plugged together before or after mounting the CAN-CBX modules.

2. Place the CAN-CBX module with the DIN rail guideway on the top edge of the mounting rail.

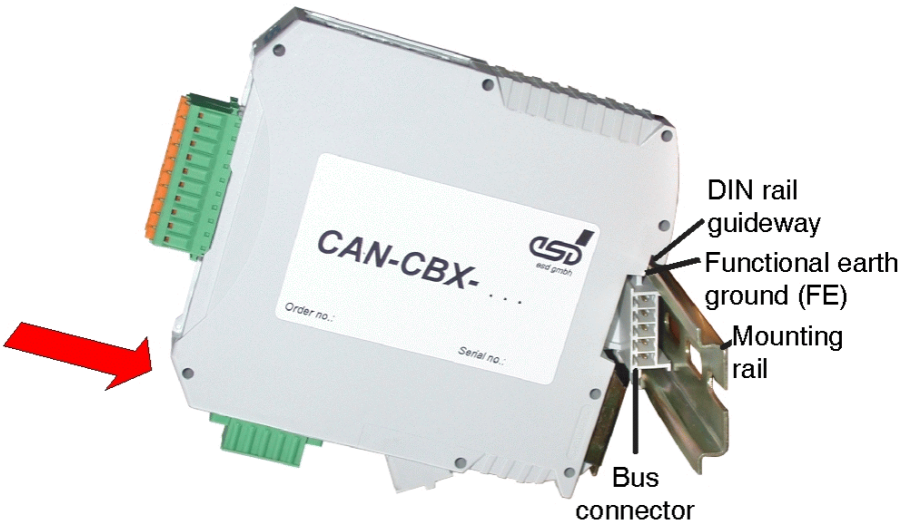


Figure 6 : Mounting CAN-CBX modules

3. Swivel the CAN-CBX module onto the mounting rail in pressing the module downwards according to the arrow as shown in figure 6. The housing is mechanically guided by the DIN rail bus connector.

4. When mounting the CAN-CBX module the metal foot catch snaps on the bottom edge of the mounting rail. Now the module is mounted on the mounting rail and connected to the InRailBus via the bus connector. Connect the bus connectors and the InRailBus if not already done.

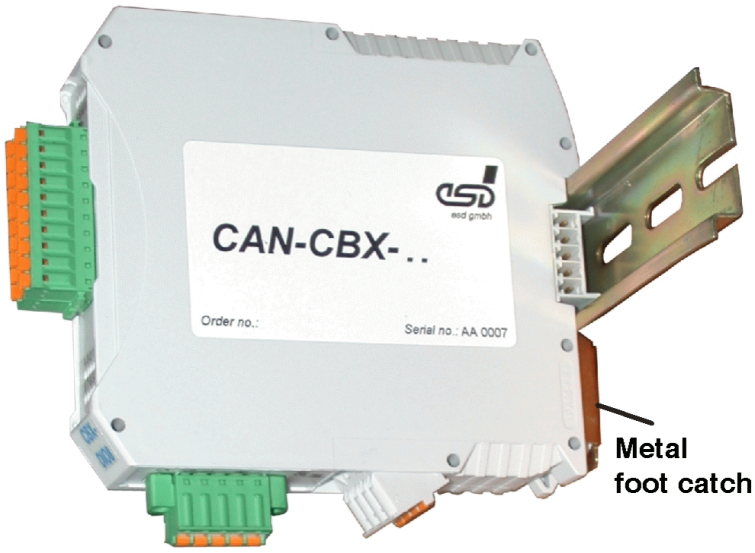


Figure 7: Mounted CAN-CBX module

3.4.1 Connecting Power Supply and CAN-Signals to CBX-InRailBus

To connect the power supply and the CAN-signals via the InRailBus, a terminal plug is needed. The terminal plug is not included in delivery and must be ordered separately (order no.: C.3000.02, see order information).

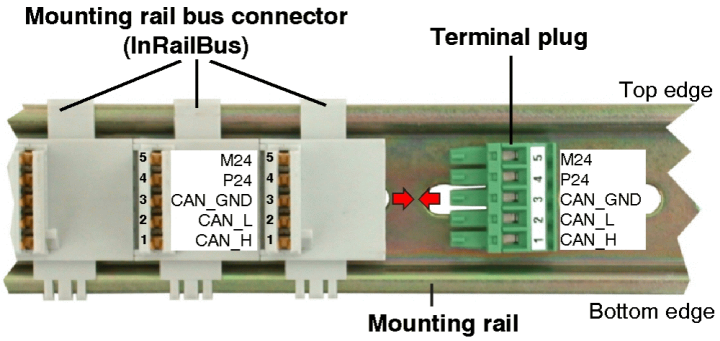


Fig. 8: Mounting rail with InRailBus and terminal plug

Plug the terminal plug into the socket on the right of the mounting-rail bus connector of the InRailBus, as described in Fig. 8. Then connect the CAN interface and the power supply voltage via the terminal plug.

3.4.2 Connection of the Power Supply Voltage

The power supply voltage can be supplied via the 24V connector or via the InRailBus.

Attention!
Please note the safety instructions containing the requirements on power supply current circuits (see page 4)!

Attention!
The connections for the 24 V power supply are internally connected and must **not** be supplied by two independent power sources at the same time!

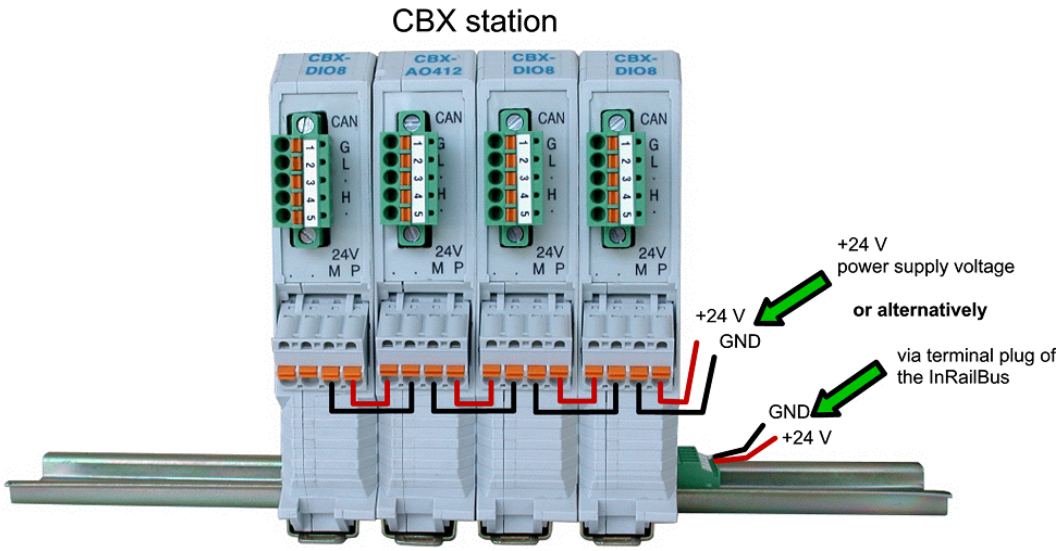


Fig. 9: Connecting the power supply voltage to the CAN-CBX station

3.4.3 Connection of CAN

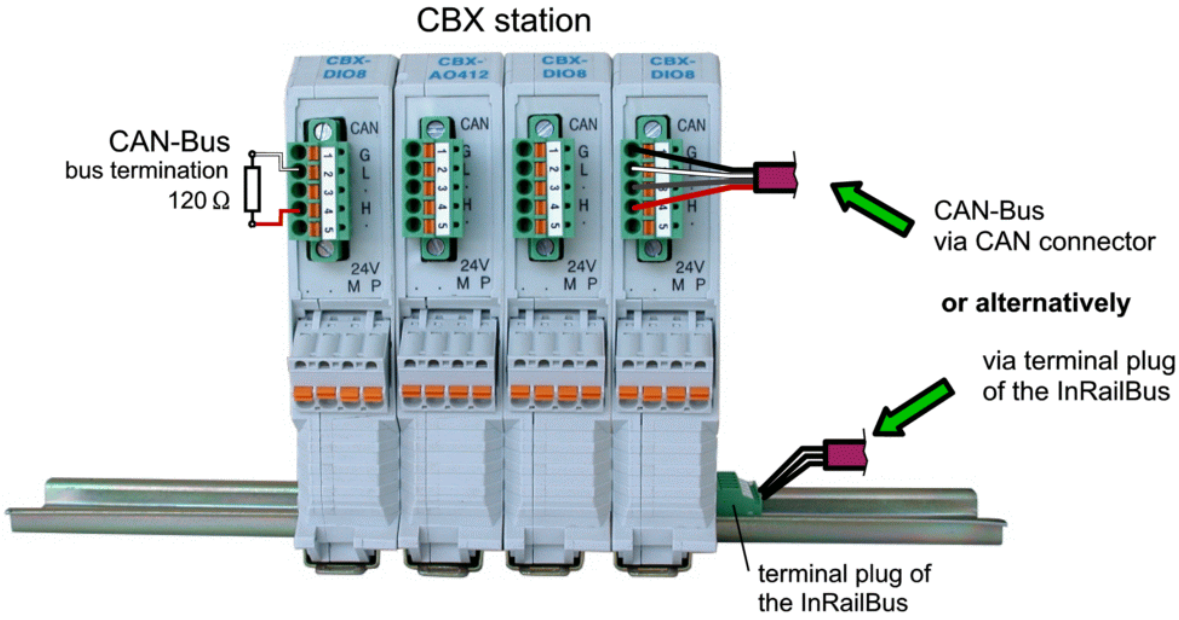


Fig. 10: Connecting the CAN signals to the CAN-CBX station

Generally the CAN signals can be fed via the CAN connector of the first CAN-CBX module of the CBX station. The signals are then connected through the CAN-CBX station via the InRailBus. To loop the CAN signals through the CBX station the CAN bus connector of the last CAN-CBX module of the CAN-CBX station has to be used. The CAN connectors of the CAN-CBX modules which are not at the ends of the CAN-CBX station must not be connected to the CAN bus, because this would cause incorrect branching.
A bus termination must be connected to the CAN connector of the CAN-CBX module at the end of the CBX-InRailBus (see Fig. 10), if the CAN bus ends there.

3.5 Remove the CAN-CBX Module from the InRailBus

If the CAN-CBX module is connected to the InRailBus please proceed as follows:

Release the module from the mounting rail in moving the foot catch (see Fig. 7) downwards (e.g. with a screwdriver). Now the module is detached from the bottom edge of the mounting rail and can be removed.

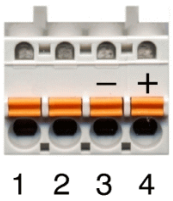
Note:
It is possible to remove individual devices from the CBX station without interrupting the InRailBus connection, because the contact chain will not be disrupted.

4. Connector Assignment

4.1 Power Supply Voltage 24 V (X100)

Device connector: Phoenix-Contact MSTBO 2,5/4-G1L-KMGY
Line connector: Phoenix-Contact FKCT 2,5/4-ST, 5.0 mm pitch, spring-cage connection,
Phoenix-Contact order no.: 19 21 90 0 (included in the scope of delivery)
For conductor connection and conductor cross section see page 30.

Pin Position:



Pin Assignment:

Labelling on Housing	24V			
	•	•	M	P
Labelling on connector	(free)	(free)	-	+

Pin No.	1	2	3	4
Signal	P24 (+ 24 V)	M24 (GND)	M24 (GND)	P24 (+ 24 V)

Please refer also to the connecting diagram on page 13.

Note:
The pins 1 and 4 are connected internally.
The pins 2 and 3 are connected internally.

Signal Description:

P24... power supply voltage +24 V
M24... reference potential

4.2 CAN Bus (X400)

4.2.1 CAN Interface

The physical layer is designed according to ISO 11898-2. The CAN bus signals are electrically isolated from the other signals via a digital isolator and a DC/DC converter.

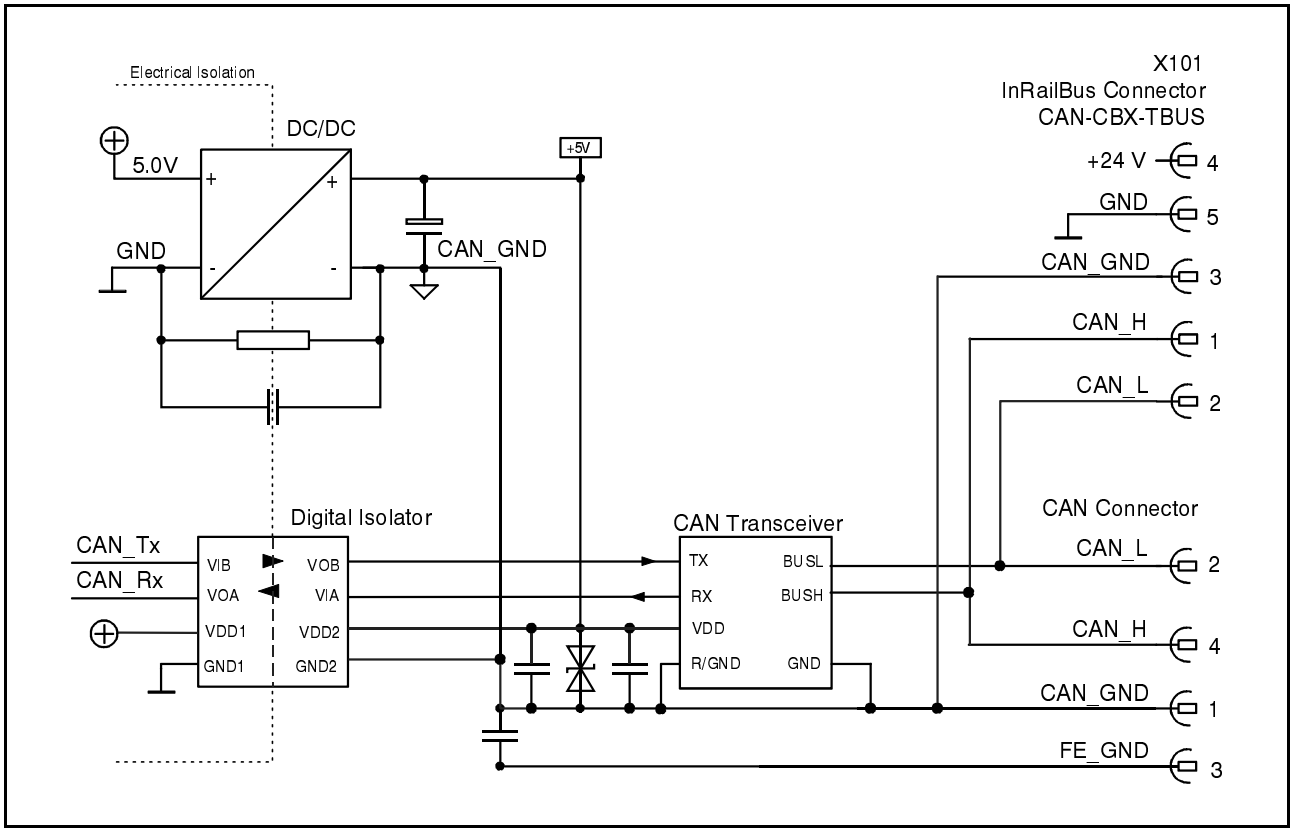
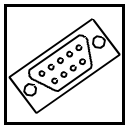


Fig. 11: CAN Interface

The CAN interface can be connected via the CAN connector or optionally via the InRailBus. Use the mounting-rail bus connector of the CBX-InRailBus (CAN-CBX-TBUS), see order information (page 107).



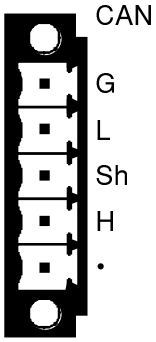
Connector Pin Assignment

4.2.2 CAN Connector (X400)

Device Connector: Phoenix-Contact MC 1,5/5-GF-3,81
Line Connector: Phoenix-Contact FK-MCP 1,5/5-STF-3,81, spring-cage connection,
Phoenix-Contact order no.:1851261 (included in the scope of delivery)
For conductor connection and conductor cross section see page 30.

Pin Position:

(Illustration of device connector)



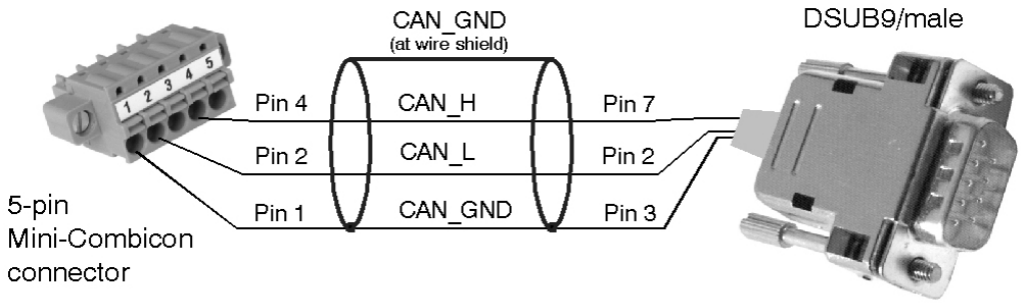
Pin-Assignment:

Labelling	Signal	Pin
G	CAN_GND	1
L	CAN_L	2
Sh	Shield	3
H	CAN_H	4
•	-	5

Signal description:

CAN_L, CAN_H ...	CAN signals
CAN_GND ...	reference potential of the local CAN physical layer
Shield ...	pin for line shield connection (using hat rail mounting direct contact to the mounting rail potential)
- ...	not connected

Recommendation of an adapter cable from 5-pin Combicon (here line connector FK-MCP1,5/5-STF-3,81 with spring-cage-connection) to 9-pin DSUB:

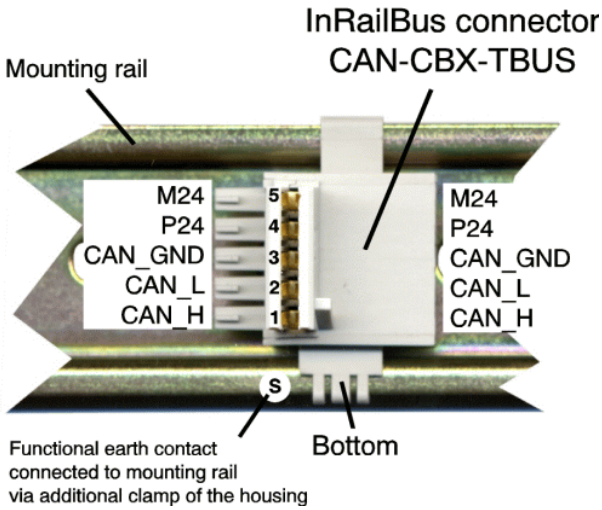


The 9-pin DSUB connector is assigned according to CiA 102

4.2.3 CAN and Power Supply Voltage via InRailBus Connector

Connector type: Mounting rail bus connector CAN-CBX-TBUS
(Phoenix-Contact ME 22,5 TBUS 1,5/5-ST-3,81 KMGY)

Pin Position:



Pin Assignment:

Pin	Signal
5	M24 (GND)
4	P24 (+24 V)
3	CAN_GND
2	CAN_L
1	CAN_H

S	FE (PE_GND)

Signal Description:

CAN_L,
CAN_H ... CAN signals
CAN_GND ... reference potential of the local CAN-Physical layers
P24... power supply voltage +24 V
M24... reference potential
FE... functional earth contact (EMC)(connected to mounting rail potential)

4.3 Analog Inputs (X500)

4.3.1 Analog Input Circuit

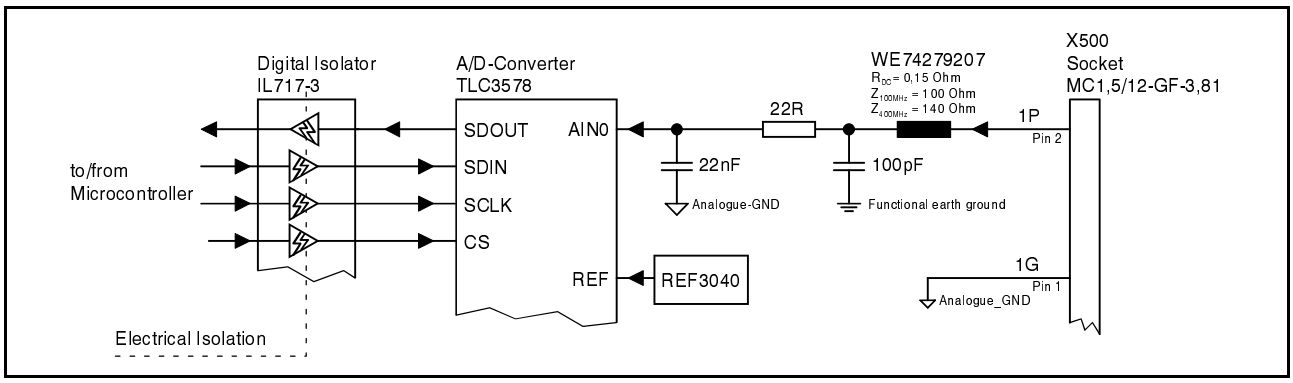
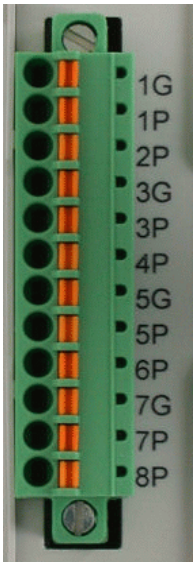


Fig. 12: Analog input circuit (example: channel 1)

4.3.2 Connector Assignmant Analog Inputs (X500)

Device's socket: Phoenix MC 1,5/12-GF-3,81
Line connector: Phoenix FK-MCP 1,5/12-STF-3,81 (spring-cage connection)
(included in delivery)
For conductor connection and conductor cross section see page 30.

Pin Position:



Pin-Assignment:

Pin	Signal
1G	GND
1P	Input Channel1
2P	Input Channel2
3G	GND
3P	Input Channel3
4P	Input Channel4
5G	GND
5P	Input Channel5
6P	Input Channel6
7G	GND
7P	Input Channel7
8P	Input Channel8

Signal description:

*x*P... Positive input pin of the analog input (*x* = 1, 2, ..., 8)
*x*G ... Reference potential analog ground (connected to functional earth contact of the module) (*x* = 1, 2, ..., 8)

Note:
The *x*G-pins are electrically connected.
The *x*P-inputs can be connected to any *x*G-contact.
To achieve short conductor loops it is recommended to use an adjacent GND-Pin.

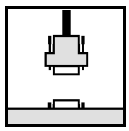
Note:
To ensure EU Conformity a cable with a maximum wire length of 3 m has to be used for the analog inputs.

4.4 Conductor Connection/Conductor Cross Sections

The following table contains an extract of the technical data of the line connectors.

Interface	**Power Supply Voltage 24 V** [6]	**Analog Inputs, CAN** [7]
Connector type plug component (Range of articles)	FKCT 2,5/..-ST KMGY	FK-MCP 1,5/..-STF-3,81
Connection method	spring-cage connection	spring-cage connection
Stripping length	10 mm	9 mm
Conductor cross section solid min.	0.2 mm²	0.14 mm²
Conductor cross section solid max.	2.5 mm²	1.5 mm²
Conductor cross section stranded min.	0.2 mm²	0.14 mm²
Conductor cross section stranded max.	2.5 mm²	1.5 mm²
Conductor cross section stranded, with ferrule without plastic sleeve min.	0.25 mm²	0.25 mm²
Conductor cross section stranded, with ferrule without plastic sleeve max.	2.5 mm²	1.5 mm²
Conductor cross section stranded, with ferrule with plastic sleeve min.	0.25 mm²	0.25 mm²
Conductor cross section stranded, with ferrule with plastic sleeve max.	2.5 mm²	0.5 mm²
Conductor cross section AWG/kcmil min.	24	26
Conductor cross section AWG/kcmil max	12	16
2 conductors with same cross section, solid min.	n.a.	n.a.
2 conductors with same cross section, solid max.	n.a.	n.a.
2 conductors with same cross section, stranded min.	n.a.	n.a.
2 conductors with same cross section, stranded max.	n.a.	n.a.
2 conductors with same cross section, stranded, ferrules without plastic sleeve, min.	n.a.	n.a.
2 conductors with same cross section, stranded, ferrules without plastic sleeve, max.	n.a.	n.a.
2 conductors with same cross section, stranded, TWIN ferrules with plastic sleeve, min.	0.5 mm²	n.a.
2 conductors with same cross section, stranded, TWIN ferrules with plastic sleeve, max.	1 mm²	n.a.
Minimum AWG according to UL/CUL	26	28
Maximum AWG according to UL/CUL	12	16

n.a. ... not allowed



5. Correct Wiring of Electrically Isolated CAN Networks

For the CAN wiring all applicable rules and regulations (EC, DIN), e.g. regarding electromagnetic compatibility, security distances, cable cross-section or material, have to be met.

5.1 Light Industrial Environment (Single Twisted Pair Cable)

5.1.1 General Rules

Note:
esd grants the EU Conformity of the product, if the CAN wiring is carried out with at least single shielded single twisted pair cables that match the requirements of ISO 118982-2. Single shielded double twisted pair cable wiring as described in chapter 5.2 ensures the EU Conformity as well.

The following general rules for CAN wiring with single shielded single twisted pair cable must be followed:

1	A cable type with a wave impedance of about 120 Ω ±10% with an adequate wire cross-section (0.22 mm²) has to be used. The voltage drop over the wire has to be considered!
2	For light industrial environment use at least a **two-wire** CAN cable. Connect ● the two twisted wires to the data signals (CAN_H, CAN_L) and ● the cable shield to the reference potential (CAN_GND)!
3	The reference potential CAN_GND has to be connected to the functional earth (FE) at exactly one point.
4	A CAN net must not branch (exception: short cable stubs) and has to be terminated with the characteristic impedance of the line (generally 120 Ω ±10%) at both ends (between the signals CAN_L and CAN_H and not at GND)!
5	Keep cable stubs as short as possible (l < 0.3 m)!
6	Select a working combination of bit rate and cable length.
7	Keep away cables from disturbing sources. If this cannot be avoided, double shielded wires are recommended.

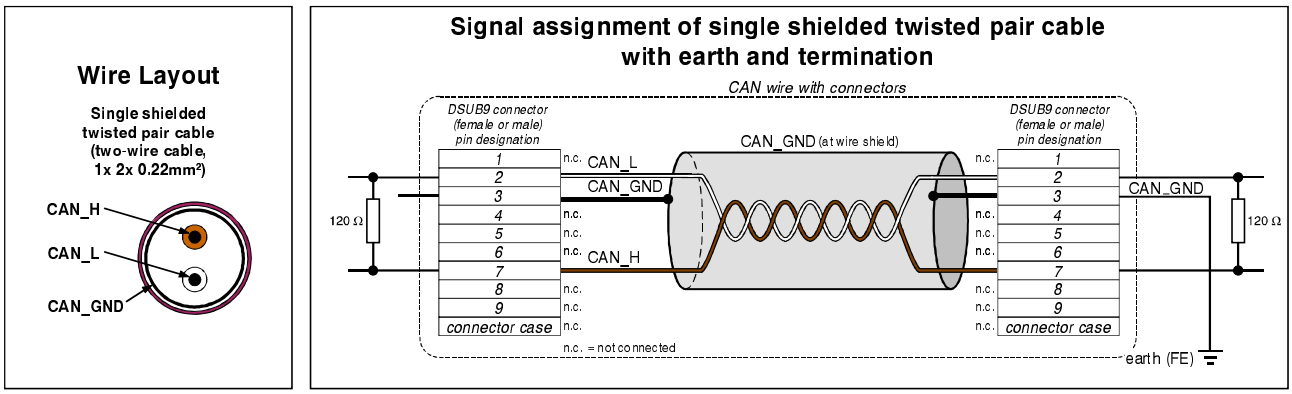


Figure. 13: CAN wiring for light industrial environment

5.1.2 Cabling

- for devices which have only one CAN connector per net use T-connectors and cable stubs (shorter than 0.3 m) (available as accessory)

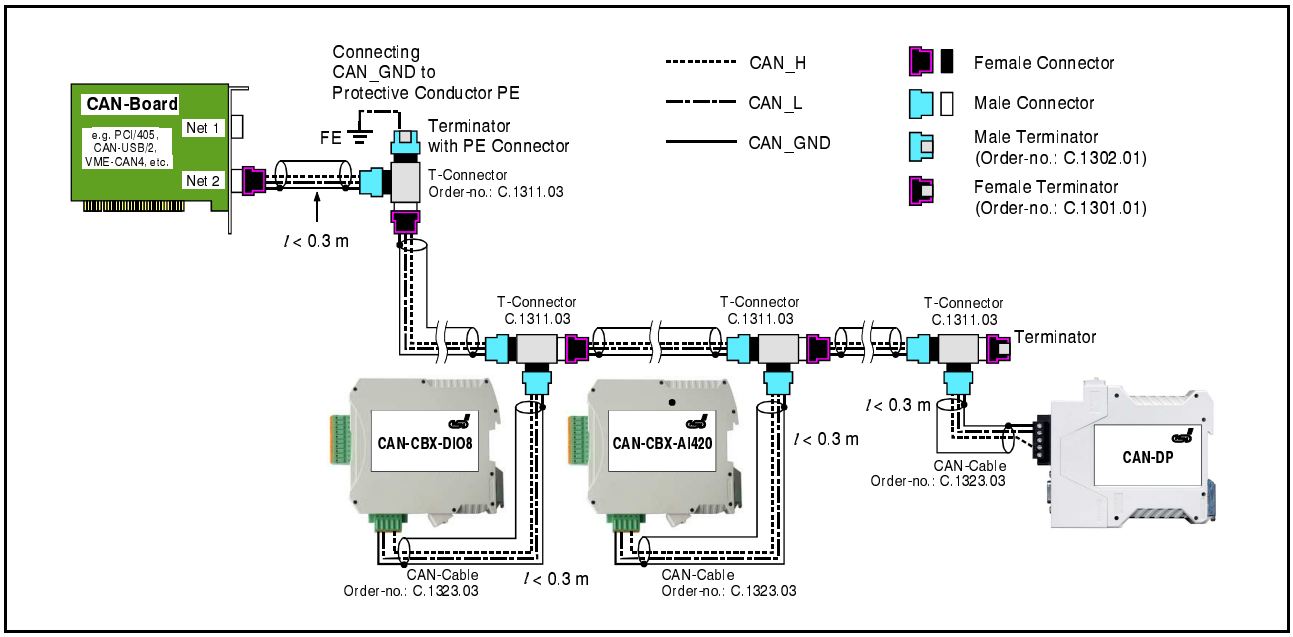


Figure. 14: Example for proper wiring with single shielded single twisted pair wires

5.1.3 Termination

- If the used CAN interface is not equipped with an integrated CAN termination and it is at an end of the bus, use external termination plugs.

- 9-pin DSUB-termination connectors with male and female contacts and earth terminal are available as accessories

5.2 Heavy Industrial Environment (Double Twisted Pair Cable)

5.2.1 General Rules

The following general rules for CAN wiring with single shielded single twisted pair cable must be followed:

1	A cable type with a wave impedance of about 120 Ω ±10% with an adequate wire cross-section (0.22 mm²) has to be used. The voltage drop over the wire has to be considered!
2	For heavy industrial environment use a **four-wire** CAN cable. Connect ● the two twisted wires to the data signals (CAN_H, CAN_L) and ● the cable shield to the reference potential (CAN_GND)! ● the cable shield to functional earth (FE) at least at one point!
3	The reference potential CAN_GND has to be connected to the functional earth (FE) at exactly **one** point.
4	A CAN net must not branch (exception: short cable stubs) and has to be terminated with the characteristic impedance of the line (generally 120 Ω ±10%) at both ends (between the signals CAN_L and CAN_H and **not** at GND)!
5	Keep cable stubs as short as possible (l < 0.3 m)!
6	Select a working combination of bit rate and cable length.
7	Keep away cables from disturbing sources. If this cannot be avoided, double shielded wires are recommended.

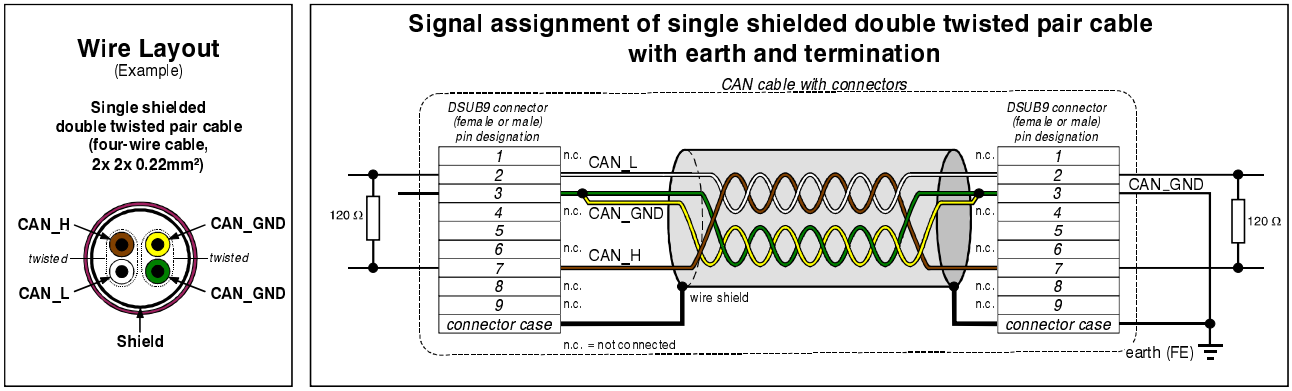


Fig. 15: CAN wiring for heavy industrial environment

5.2.2 Device Cabling

Attention:
If single shielded double twisted pair cables are used, realize the T-connections by means of connectors that support connection of two CAN cables at one connector where the cable's shield is looped through e.g. DSUB9-connector from ERNI (ERBIC CAN BUS MAX, order no.:154039).

The usage of esd's T-connector type C.1311.03 is not recommended for single shielded double twisted pair cables because the shield potential of the conductive DSUB housing is not looped through this T-connector type.

Furthermore, mixed use of single twisted and double twisted cables should be avoided!

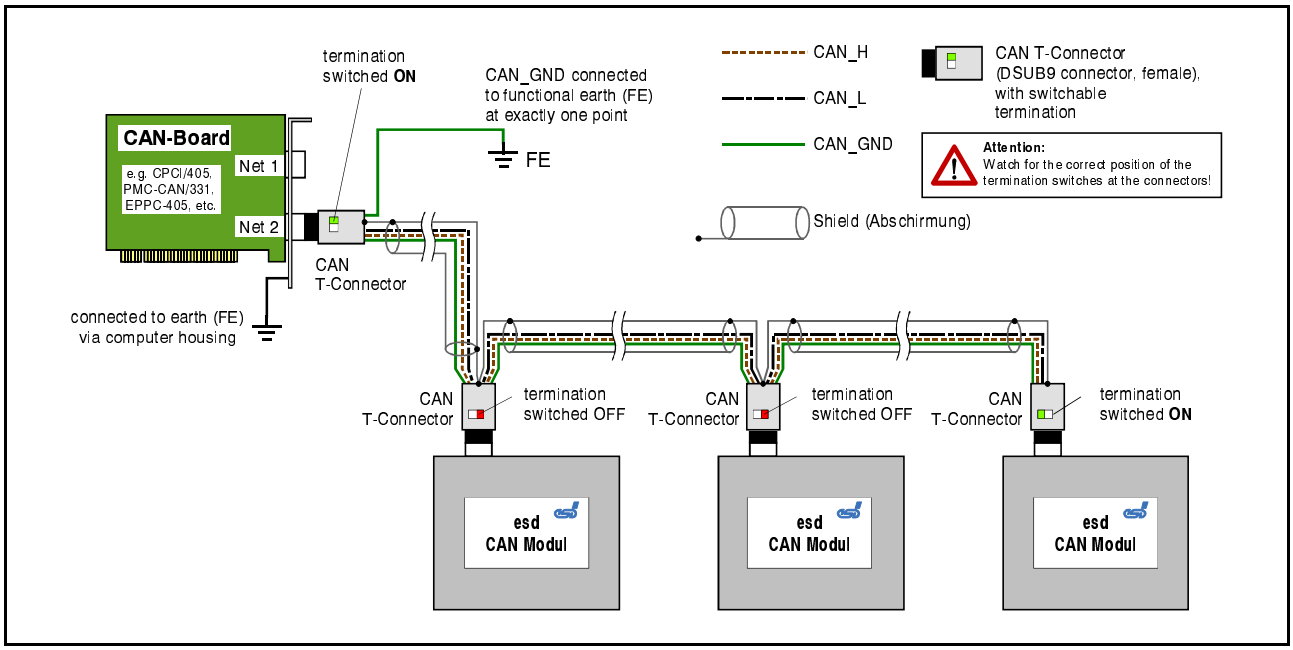
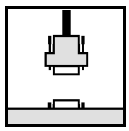


Fig. 16: Example for proper wiring with single shielded double twisted pair cables

5.2.3 Termination

- If the used CAN interface is not equipped with an integrated CAN termination and it is at an end of the bus, use external termination plugs.

- A 9-pin DSUB-connector with integrated switchable termination resistor can be ordered e.g. from ERNI (ERBIC CAN BUS MAX, female contacts, order no.:154039).



5.3 Electrical Grounding

- For CAN devices with electrical isolation the CAN_GND must be connected between the CAN devices!
- CAN_GND has to be connected to the earth potential (FE) at **exactly one** point in the net!
- Each *CAN interface with electrical connection to earth potential* acts as an earthing point. For this reason do not connect more than one *CAN device with electrical connection to earth potential*!
- Earthing can e.g. be made at a connector

5.4 Bus Length

- Optical couplers are delaying the CAN signals. By using fast digital isolators and testing each board at 1 Mbit/s, esd modules typically reach a wire length of 37 m at 1 Mbit/s within a closed net without impedance disturbances like e.g. longer stub.

Bit-Rate [kBit/s]	Typical values of reachable wire length **with esd interface** l_{max} [m]	**CiA recommendations** (07/95) for reachable wire lengths l_{min} [m]
1000	37	25
800	59	50
666.6̄	80	-
500	130	100
333.3̄	180	-
250	270	250
166	420	-
125	570	500
100	710	650
83.3̄	850	-
66.6̄	1000	-
50	1400	1000
33.3̄	2000	-
20	3600	2500
12.5	5400	-
10	7300	5000

Table 12: Reachable wire lengths depending on the bit rate when using esd-CAN interfaces

Note:
Please note the recommendations according to ISO 11898 for the selection of the cross section of the wire depending of the wire length.

5.5 Examples for CAN Cables

5.5.1 Cable for Light Industrial Environment Applications (Two-Wire)

Manufacturer	Cable Type
U.I. LAPP GmbH Schulze-Delitzsch-Straße 25 70565 Stuttgart Germany www.lappkabel.de	e.g. UNITRONIC ®-BUS CAN UL/CSA (1x 2x 0.22) (UL/CSA approved) Part No.: 2170260 UNITRONIC ®-BUS-FD P CAN UL/CSA (1x 2x 0.25) (UL/CSA approved) Part No.: 2170272
ConCab GmbH Äußerer Eichwald 74535 Mainhardt Germany www.concab.de	e.g. BUS-PVC-C (1x 2x 0.22 mm²) Part No.: 93 022 016 (UL appr.) BUS-Schleppflex-PUR-C (1x 2x 0.25 mm²) Part No.: 94 025 016 (UL appr.)

5.5.2 Cable for Heavy Industrial Environment Applications (Four-Wire)

Manufacturer	Cable Type
U.I. LAPP GmbH Schulze-Delitzsch-Straße 25 70565 Stuttgart Germany www.lappkabel.de	e.g. UNITRONIC ®-BUS CAN UL/CSA (2x 2x 0.22) (UL/CSA approved) Part No: 2170261 UNITRONIC ®-BUS-FD P CAN UL/CSA (2x 2x 0.25) (UL/CSA approved) Part No.: 2170273
ConCab GmbH Äußerer Eichwald 74535 Mainhardt Germany www.concab.de	e.g. BUS-PVC-C (2x 2x 0.22 mm²) Part No.: 93 022 026 (UL appr.) BUS-Schleppflex-PUR-C (2x 2x 0.25 mm²) Part No.: 94 025 026 (UL appr.)

Note:
Completely configured CAN cables can be ordered from **esd**.

6. CAN-Bus Troubleshooting Guide

The CAN-Bus Troubleshooting Guide is a guide to find and eliminate the most frequent hardware-error causes in the wiring of CAN-networks.

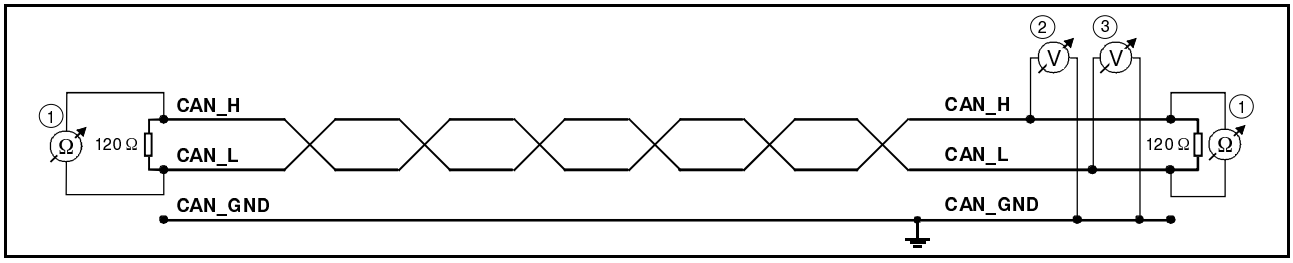


Figure. 17: Simplified diagram of a CAN network

6.1 Termination

The termination is used to match impedance of a node to the impedance of the transmission line being used. When impedance is mismatched, the transmitted signal is not completely absorbed by the load and a portion is reflected back into the transmission line. If the source, transmission line and load impedance are equal these reflections are eliminated. This test measures the series resistance of the CAN data pair conductors and the attached terminating resistors.

To test it, please

1. Turn off all power supplies of the attached CAN nodes.
2. Measure the DC resistance between CAN_H and CAN_L at the ends of the network (see figure above) and the middle (if the network cable consists of more than one line section). (1)

The measured value should be between 50 Ω and 70 Ω. The measured value should be nearly the same at each point of the network.

If the value is below 50 Ω, please make sure that:
- there is no **short circuit** between CAN_H and CAN_L wiring
- there are **not more than two** terminating resistors connected
- the nodes do not have faulty transceivers.

If the value is higher than 70 Ω, please make sure that:
- there are no open circuits in CAN_H or CAN_L wiring
- your bus system has two terminating resistors (one at each end) and that they are 120 Ω each.

6.2 Ground

CAN_GND of the CAN network has to be connected to Functional earth potential at only one location. This test will indicate if the shielding is grounded in several places. To test it, please

1. Disconnect the CAN_GND from the earth potential (FE).
2. Measure the DC resistance between CAN_GND and earth potential (see picture on the right hand).
3. Connect CAN_GND to earth potential.

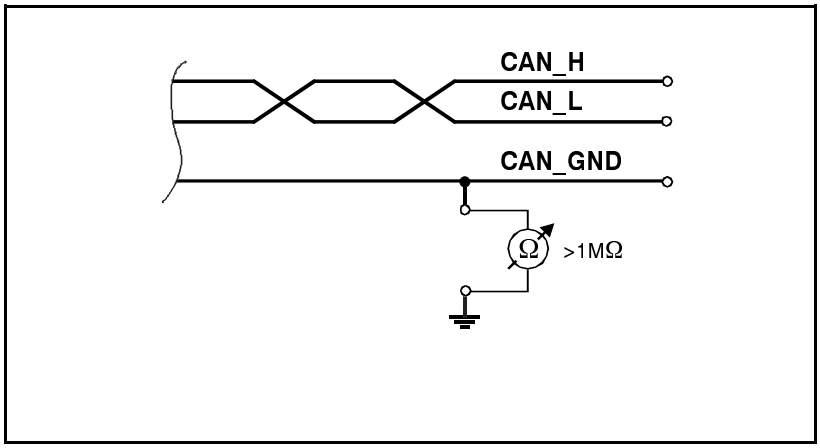


Fig. 18: Simplified schematic diagram of ground test measurement

The resistance should be higher than 1 M Ω. If it is lower, please search for additional grounding of the CAN-GND wires.

6.3 Short Circuit in CAN Wiring

A CAN bus might possibly still be able to transmit data, if there is a short circuit between CAN_GND and CAN_L, but the error rate will increase strongly. Make sure that there is no short circuit between CAN_GND and CAN_L!

6.4 CAN_H/CAN_L Voltage

Each node contains a CAN transceiver that outputs differential signals. When the network communication is idle the CAN_H and CAN_L voltages are approximately 2.5 volts. Faulty transceivers can cause the idle voltages to vary and disrupt network communication.

To test for faulty transceivers, please
1. Turn on all supplies.
2. Stop all network communication.
3. Measure the DC voltage between CAN_H and GND (2) (see figure above).
4. Measure the DC voltage between CAN_L and GND (3) (see figure above).

Normally the voltage should be between 2.0 V and 4.0 V.

If it is lower than 2.0 V or higher than 4.0 V, it is possible that one or more nodes have faulty transceivers.
For a voltage lower than 2.0 V please check CAN_H and CAN_L conductors for continuity. For a voltage higher than 4.0 V, please check for excessive voltage.

To find the node with a faulty transceiver please test the CAN transceiver resistance (see next page).

6.5 CAN Transceiver Resistance Test

CAN transceivers have one circuit that controls CAN_H and another circuit that controls CAN_L. Experience has shown that electrical damage to one or both of the circuits may increase the leakage current in these circuits.

To measure the current leakage through the CAN circuits, please use an resistance measuring device and:

1. Switch off the (4) node and disconnect it from the network (see figure below).
2. Measure the DC resistance between CAN_H and CAN_GND (5) (see figure below).
3. Measure the DC resistance between CAN_L and CAN_GND (6) (see figure below).

Normally the resistance should be between 1 M Ω and 4 M Ω or higher. If it is lower than this range, the CAN transceiver is probably faulty.
Another sign for a faulty transceiver is a very high deviation between the two measured input resistance (>> 200%).

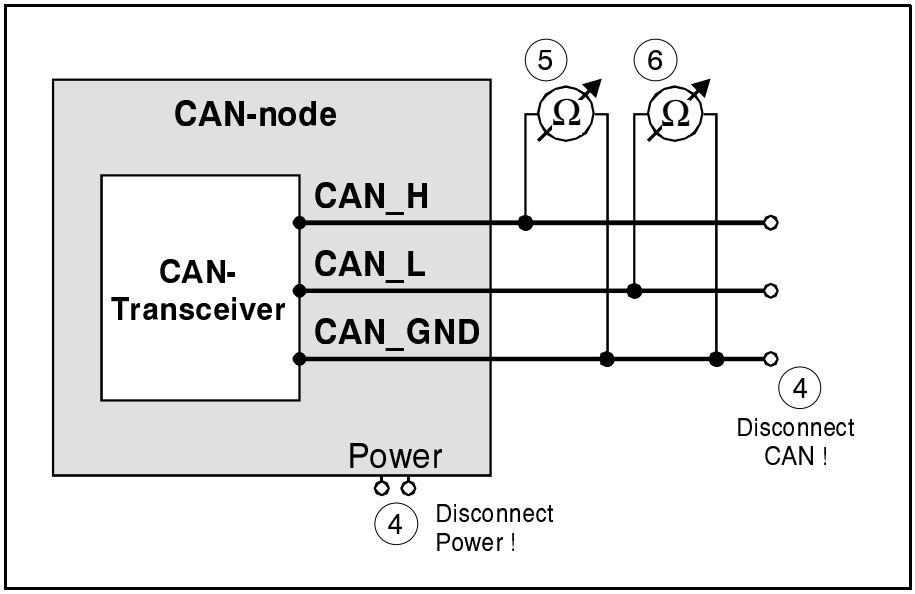


Figure 19: Simplified diagram of a CAN node

7. Quick Start Guide

For a quick start with a simple configuration for the event-triggered transmission of data the following steps are necessary:

Step		see page
	Read the safety notes at the beginning of the manual carefully before you start with the installation!	4
	Please also note the chapters ‘Installation of the Module Using InRailBus Connector’ and ‘Correct Wiring of Electrically Isolated CAN Networks’!	20, 31
1	Mount the CAN-CBX-AI814 module and connect the interfaces (power supply voltage, CAN, analog inputs)	13
2	Please note that the CAN bus has to be terminated at both ends! **esd** offers special T-connectors and terminator connectors. Additionally the CAN_GND signal has to be connected to earth at **exactly one** point. For easier wiring the termination connectors are equipped with an earth connector. A CAN participant without an electrically isolated interface acts as an earth connection.	-
3	Setting the baud rate (only if another baud rate than the default baud rate is requested.) The default baud rate is 1 MBit/s. The baud rate can be configured via the coding switch BAUD, as described in chapter: ‘Setting the Baud Rate’.	19
4	Setting the module number (Node-ID). The node-ID can be configured via the coding switches LOW und HIGH. It can be set to values between 1 and 127 (01-7F$_h$). Example: For node- ID 1 the coding switch LOW has to be set to ‘1' and the coding switch HIGH has to be set to '0'.	18

<table>
<tr>
<td>5</td>
<td>Set Interrupt_Enable
- Switch on the module. Connect power supply voltage
- transmit the following CAN message:
Set Global_Interrupt_Enable to true ('1'), the module will then transmit the measured values every 2,5 ms.

<table>
<tr><th>CAN-Identifier</th><th>Len</th><th colspan="8">Data</th></tr>
<tr><td>600_h + Node-ID</td><td>8 Byte</td><td>2F</td><td>23</td><td>64</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr>
</table>
</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>Send the start telegram for all CANopen modules (NMT-start command)

<table>
<tr><th>CAN-Identifier</th><th>Len</th><th colspan="2">Data</th></tr>
<tr><td>0</td><td>2 Byte</td><td>1</td><td>00 or
Node-ID
(00 = start all modules)</td></tr>
</table>
Now the CAN-CBX-AI814 should be cyclically transmitting data on the 2. and 3. PDO.
</td>
<td>-</td>
</tr>
</table>

8. CANopen Firmware

Apart from basic descriptions of the CANopen, this chapter contains the most significant information about the implemented functions.

A complete CANopen description is too extensive for the purpose of this manual.
Further information can therefore be taken from the CANopen documentation [1] and [4].

8.1 Definition of Terms

COB ...	Communication Object
Emergency-Id...	Emergency Data Object
NMT...	Network Management (Master)
SDO...	Service Data Object
Sync...	Sync(frame) Telegram

PDOs (Process Data Objects)
PDOs are used to transmit process data.
In the ‘Transmit’-PDO (TPDO) the CAN-CBX-module transmits data to the CANopen network.
In the ‘Receive’-PDO (RPDO) the CAN-CBX-module receives data from the CANopen network.

SDOs (Service Data Objects)
SDOs are used to transmit module internal configuration- and parameter data. In opposition to the PDOs SDO-messages are confirmed. A write or read request on a data object is always answered by a response telegram with an error index.

8.2 NMT-Boot-up

The CAN-CBX module can be initialized with the 'Minimum Capability Device' boot-up as described in [1].

Usually a telegram to switch from *Pre-Operational* status to *Operational* status after boot-up is sufficient. For this the 2-byte telegram '01_h', '00_h', for example, has to be transmitted with CAN-identifier '0000_h' (= Start Remote Node all Devices).

8.3 The CANopen-Object Directory

The object directory is basically a (sorted) group of objects which can be accessed via the CAN network. Each object in this directory is addressed with a 16-bit index. The index in the object directories is represented in hexadecimal format.
The index can be a 16-bit parameter in accordance with the CANopen specification [1] or a manufacturer-specific code. By means of the MSBs of the index the object class of the parameter is defined.

Part of the object directory are among others:

Index	Object
0001_h ... $009F_h$	definition of data types
1000_h ... $1FFF_h$	Communication Profile Area
2000_h ... $5FFF_h$	Manufacturer Specific Profile Area
6000_h ... $9FFF_h$	Standardized Device Profile Area
$A000_h$ ... $FFFF_h$	reserved

8.4 Communication Parameters of the PDOs

The communication parameters of the PDOs (according to [1]) are transmitted as SDO (Service Data Objects) on ID '**600_h + Node-ID**' (Request). The receiver acknowledges the parameters on ID '**580_h + Node-ID**' (Response).

The **Node-ID** (module No.) is configured via coding switches Low and High. Please refer to chapter "Coding Switches" (page 18) for a detailed description of possible configurations.

8.4.1 Access on the Object Directory

The SDOs (Service Data Objects) are used to access the object directory of a device.
An SDO is therefore a 'channel' to access the parameters of the device. Access via this channel is possible in *operational* and *pre-operational* status.

The SDOs (Service Data Objects) are transmitted on ID '**600_h + Node-ID**' (request).
The server acknowledges the parameters on ID '**580_h + Node-ID**' (response).

An SDO is structured as follows:

Identifier	Command code	Index		Sub-index	LSB	Data field	MSB
		(low)	(high)				

Example:

600_h+ Node-ID	23_h (write)	00_h ($Index=1400_h$) (Receive-PDO-Comm-Para)	14_h	01_h (COB-def.)	$7F_h$	04_h	00_h	00_h COB Node ID = 0000 $047F_h$

Identifier
The parameters are transmitted with ID '600_h + NodeID' (request).
The receiver acknowledges the parameters with ID '580_h + NodeID' (response).

Command code
The command code transmitted consists among other things of the Command Specifier and the length. Frequently required combinations are, for instance:

$40_h = 64_{dec}$: Read Request, i.e. a parameter is to be read
$23_h = 35_{dec}$: Write Request with 32-bit data, i.e. a parameter is to be set

The CAN-CBX-module responds to every received telegram with a response telegram. This can contain the following command codes:

$43_h = 67_{dec}$: Read Response with 32 bit data, this telegram contains the parameter requested
$60_h = 96_{dec}$: Write Response, i.e. a parameter has been set successfully
$80_h = 128_{dec}$: Error Response, i.e. the CAN-CBX-module reports a communication error

Frequently Used Command Codes
The following table summarizes frequently used command codes. The command frames must always contain 8 data bytes. Notes on the syntax and further command codes can be found in [1].

Command	Number of data bytes	Command code
Write Request (Initiate Domain Download)	1 2 3 4	$2F_h$ $2B_h$ 27_h 23_h
Write Response (Initiate Domain Download)	-	60_h
Read Request (Initiate Domain Upload)	-	40_h
Read Response (Initiate Domain Upload)	1 2 3 4	$4F_h$ $4B_h$ 47_h 43_h
Error Response (Abort Domain Transfer)	-	80_h

Index, Sub-Index
Index and sub-index will be described in the chapters “Device Profile Area” and “Manufacturer Specific Objects” of this manual.

Data Field
The data field has got a size of a maximum of 4 bytes and is always structured ‘LSB first, MSB last’. The least significant byte is always in ‘Data 1’. With 16-bit values the most significant byte (bits 8...15) is always in ‘Data 2’, and with 32-bit values the MSB (bits 24...31) is always in ‘Data 4’.

Error Codes of the SDO Domain Transfer

The following error codes might occur (according to [1]):

Abort code	Description
05040001_h	wrong command specifier
06010002_h	wrong write access
06020000_h	wrong index
06040041_h	object can not be mapped to PDO
06060000_h	access failed due to an hardware error
06070010_h	wrong number of data bytes
06070012_h	service parameter too long
06070013_h	service parameter too small
06090011_h	wrong sub-index
06090030_h	transmitted parameter is outside the accepted value range
08000000_h	undefined cause of error
08000020_h	data cannot be transferred or stored in the application
08000022_h	data cannot be transferred or stored in the application because of the present device state
08000024_h	access to flash failed

8.5 Overview of used CANopen-Identifiers

Function	Identifier	Description
Network management	0	NMT
SYNC	80_h	Sync to all, (configurable via object 1005_h)
Emergency Message	80_h + *Node-ID*	configurable via object 1014_h
TPDO2	280_h + *Node-ID*	PDO2 from CAN-CBX-AI814 (object 1801_h)
TPDO3	380_h+ *Node-ID*	PDO3 from CAN-CBX-AI814 (object 1802_h)
Client-SDO	580_h + *Node-ID*	SDO from CAN-CBX-AI814 (transmit SDO)
Server-SDO	600_h + *Node-ID*	SDO to CAN-CBX-AI814 (receive SDO)
Node Guarding	700_h + *Node-ID*	configurable via object $100E_h$

NodeID: CANopen address [1_h...$7F_h$]

8.5.1 Setting the COB-ID

The COB-IDs which can be set (except the one of SYNC), are deduced initially from the setting of the Node-ID via the coding switches (see page 18). If the COB-IDs are set via SDO, this setting is valid even if the coding switches are set to another Node-ID after that.
To accept the Node-ID from the coding switches again, the *Comm defaults* or all defaults have to be restored (object 1011_h)

8.6 Default PDO-Assignment

PDOs (Process Data Objects) are used to transmit process data. The PDO mapping can be changed. The following tables show the default mapping at delivery of the module:

PDO	CAN Identifier	Length	Transmission direction	Assignment
TPDO1	n.a.	n.a.	n.a.	TPDO1 is not used
TPDO2	280_h + Node-ID	8 byte	from CAN-CBX-AI814 (Transmit PDO)	A/D values channel 1 to 4 as 16 bit-values
TPDO3	380_h + Node-ID	8 byte	from CAN-CBX-AI814 (Transmit PDO)	A/D values channel 5 to 8 as 16 bit values

TPDO2 (CAN-CBX-AI814 ->)

CAN Identifier: 280_h + Node-ID

Byte	0	1	2	3	4	5	6	7
Parameter	*Analog_Input_16_1*		*Analog_Input_16_2*		*Analog_Input_16_3*		*Analog_Input_16_4*	

TPDO3 (CAN-CBX-AI814 ->)

CAN-Identifier: 380_h + Node-ID

Byte	0	1	2	3	4	5	6	7
Parameter	*Analog_Input_16_5*		*Analog_Input_16_6*		*Analog_Input_16_7*		*Analog_Input_16_8*	

Parameter description:

Name	Description	Data type	see page
Analog_Input_ 16_x	Read the analog inputs x (x = 1-8)	integer 16	83

8.7 Reading the Analog Values

8.7.1 Messages of the Analog Inputs

The transmission types for the analog inputs are described in the following:

- *acyclic, synchronous*: The transmission is initiated if a SYNC-message has been received (PDO-transmission type 0) and data has changed.
- *cyclic, synchronous*: The transmission is initiated if a defined number of SYNC-messages have been received (PDO-transmission type 1...240).
- *synchronous, remote request*: The state of the inputs is latched with each SYNC-message and is transmitted after the reception of a RTR-frame (PDO-transmission type 252).
- *asynchronous, remote request*: After the reception of a RTR-frame the last latched state of the inputs is transmitted (PDO-transmission type 253).
- *event controlled, asynchronous*: The transmission is initiated if the state of selected inputs has changed (PDO-transmission type 254, 255).

8.7.2 Supported Transmission Types Based on DS-301

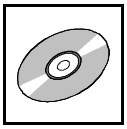
Transmission Type	PDO-Transmission					supported by CAN-CBX-AI814
	cyclic	acyclic	synchro-nous	asynchro-nous	RTR	
0		X	X			x
1...240	X		X			x
241...251	reserved					-
252			X		X	x
253				X	X	x
254				X	X	x
255				X	X	x

8.8 Communication Profile Area

8.8.1 Used Names and Abbreviations

The following names are used in the tables for the description of the communication parameters:

PDO-Mappable	PDO-Mapping is possible for this Sub-index of the PDO
Save to EEPROM	the value of this parameter is stored in the local EEPROM, if the command 'save' is called (see page 63)
Data type	data type (e.g. unsigned 8, unsigned 32)
Access mode	allowed access modes to this parameter ro... read_only This parameter can only be read. Write accesses will cause an error message. const.... constant This parameter can not be set by the user. It is readable. Write accesses will cause an error message. rw... read&write This parameter can be read or written.
Value range	value range of the parameter
Default value	default setting of the parameter
Name/Description	name and short description of the parameter

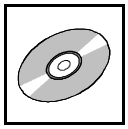


8.9 Implemented CANopen-Objects

A detailed description of the objects can be taken from CiA DS-301.

8.9.1 Overview of Communication Profile Objects with Product-Specific Properties

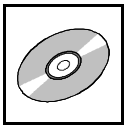
Index	Sub-index (max.)	Description	Data type	Access mode	Product-specific properties
1000_h	-	Device Type	unsigned 32	ro	00040191_h
1001_h	-	Error Register	unsigned 8	ro	supported error-bits: 0: generic 2: voltage 4: communication error
1003_h	10	Pre-Defined-Error-Field	unsigned 32	rw	00_h
1005_h	-	COB-ID-Sync	unsigned 32	rw	80_h
1006_h	-	Communication Cycle Period	unsigned 32	rw	00_h
1008_h	-	Manufacturer Device Name	visible string	ro	'CAN-CBX-AI814'
1009_h	-	Manufacturer Hardware Version	visible string	ro	x.yy (depending on version)
$100A_h$	-	Manufacturer Software Version	visible string	ro	x.yy (depending on version)
$100C_h$	-	Guard Time	unsigned 16	rw	0000_h
$100D_h$	-	Life Time Factor	unsigned 8	rw	00_h
$100E_h$	-	Node Guarding Identifier	unsigned 32	rw	Node-ID + 700_h
1010_h	4	Store Parameter	unsigned 32	rw	
1011_h	4	Restore Parameter	unsigned 32	rw	
1014_h	-	COB-ID Emergency Object	unsigned 32	rw	80_h + Node-ID
1015_h	-	Inhibit Time EMCY	unsigned 16	rw	00_h
1016_h	1	Consumer Heartbeat Time	unsigned 32	rw	00_h
1017_h	-	Producer Heartbeat Time	unsigned 16	rw	00_h
1018_h	4	Identity Object	unsigned 32	ro	Vendor Id: 00000017_h Prod. Code: 23020002_h
1019_h	-	Synchronous Counter Overflow	unsigned 8	rw	00_h
1029_h	3	Error Behaviour	unsigned 8	rw	00_h



Implemented CANopen Objects

Index	Sub-index	Description	Data type	Access mode
1801_h	5	2. Transmit PDO-Parameter	PDO CommPar (20_h)	rw
1802_h	5	3. Transmit PDO-Parameter	PDO CommPar (20_h)	rw
$1A01_h$	4	2. Transmit PDO-Mapping	PDO Mappping (21_h)	rw
$1A02_h$	4	3. Transmit PDO-Mapping	PDO Mappping (21_h)	rw

Index	Sub-index (max.)	Description	Data type	Access mode	Product-specific properties
$1F80_h$	-	NMT startup	unsigned 32	rw	default: 2 (autostart disabled)
$1F91_h$	1	Self starting nodes timing parameters	unsigned 16	rw	default: 64_h (= 100 ms)



8.9.2 Device Type (1000_h)

INDEX	**1000_h**
Name	*device type*
Data type	unsigned 32
Access mode	ro
Default value	see chapter 8.9.1 (page 51)

Example: Reading the Device Type

The CANopen master transmits the read request with identifier '603_h' (600_h + Node-ID) to the CAN-CBX module with the module no. 3 (Node-ID=3_h):

ID	RTR	LEN	DATA							
			1	2	3	4	5	6	7	8
603_h	0_h	8_h	40_h Read Request	00_h	10_h Index=1000_h	00_h Sub Index	00_h	00_h	00_h	00_h

The CAN-CBX module no. 3 responds to the client by means of read response with identifier '583_h' (580_h + Node-ID) with the value of the device type:

ID	RTR	LEN	DATA							
			1	2	3	4	5	6	7	8
583_h	0_h	8_h	43_h Read Response	00_h	10_h Index=1000_h	00_h Sub Index	**94_h** Example here: Device Profile Nr.0191_h	**01_h**	**02_h** Example here: Input	**00_h**

value of device type: 0002.0194_h.
The value of the device type of this CAN-CBX module is printed in chapter 8.9.1 (page 51)

The data field is always structured following the rule 'LSB first, MSB last' (see page 45, data field).

8.9.3 Error Register (1001_h)

The CAN-CBX module uses the error register to indicate error messages.

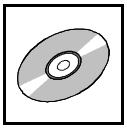
INDEX	**1001_h**
Name	*error register*
Data type	unsigned 8
Access type	ro
Default value	0

The following bits of the error register are being supported at present:

Bit	Meaning
0	*generic*
1	*current*
2	*voltage*
3	*temperature*
4	*communication error* (overrun, error state)
5	*device profile*
6	reserved
7	*manufacturer*

For a list of the error bits supported by this CAN-CBX module see chapter 8.9.1 (page 51).

Bits which are not supported are always returned as ‘0’.
If an error is active, the according bit is set to ‘1’.



8.9.4 Pre-defined Error Field (1003_h)

INDEX	**1003_h**
Name	*pre-defined error field*
Data type	unsigned 32
Access mode	ro
Default value	No

The *pre-defined error field* provides an error history of the errors that have occurred on the device and have been signalled via the Emergency Object.
Sub-index 0 contains the current number of errors stored in the list.
Under sub-index 1 the last error which occurred is stored. If a new error occurs, the previous error is stored under sub-index 2 and the new error under sub-index 1, etc. In this way a list of the error history is created.
The error buffer is structured like a ring buffer. If it is full, the oldest entry is deleted for the latest entry.

This module supports a maximum of 10 error entries. When the 11th error occurs the oldest error entry is deleted. In order to delete the entire error list, sub-index '0' has to be set to '0'. This is the only permissible write access to the object.

With every new entry to the list the module transmits an **Emergency Frame** to report the error.

Index	Sub-index	Description	Value range	Default	Data type	Access mode
1003_h	0	*no_of_errors_in_list*	0, 1...A_h	-	unsigned 8	rw
	1	*error-code n*	0...$FFFFFFFF_h$	-	unsigned 32	ro
	2	*error-code (n-1)*	0...$FFFFFFFF_h$	-	unsigned 32	ro
	:	*:*	:	:	:	ro
	A_h	*error-code (n-9)*	0...$FFFFFFFF_h$	-	unsigned 32	ro

Meaning of the variables:

no_of_errors_in_list - contains the number of error codes currently on the list
n = number of error which occurred last
- in order to delete the error list this variable has to be set to '0'
- if *no_of_errors_in_list* ≠0, the error register (Object 1001_h) is set

error-code x The 32-bit long error code consists of the CANopen-emergency error code described in [1] and the error code defined by esd (manufacturer-specific error field).

Bit:	31 16	15 0
Contents:	*manufacturer-specific error field*	*emergency-error-code*

manufacturer-specific error field: always ‘00’, unless
emergency-error-code = 2300_h (see below)

emergency-error-code: The following error-codes are supported:

8110_h - CAN overrun error
- Sample rate is set too high, thus the firmware is not able to transmit all data to the CAN bus.

8120_h - CAN in error passive mode
8130_h - Lifeguard error / heartbeat error
8140_h - Recovered from “Bus Off”
8240_h - Unexpected SYNC data length
6000_h - Software error:
-EEPROM checksum error (no transmission of this error message as emergency message)
6110_h - Internal Software error
e.g.:
- saved data had invalid checksum and default data is loaded

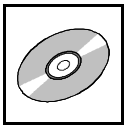
$FF10_h$ - Data loss (A/D data overflow)
5000_h - Hardware error (e.g. A/D-converter defective)
5030_h - Sensor error

Emergency Message

The data otf the emergency frame transmitted by the CAN-CBX-module have the following structure:

Byte:	0	1	2	3	4	5	6	7
Contents:	*emergency-error-code* (siehe oben)		*error-register* 1001_h	*no_of_errors _ in_list* $1003,00_h$	-			

An emergency message is transmitted, if an error occurs. If this error occurs again, no further emergency message is generated.
If the last error message is cancelled, again an emergency message is transmitted to indicate the error disappearance.



8.9.5 COB-ID of SYNC-Message (1005_h)

INDEX	**1005_h**
Name	*COB-ID SYNC message*
Data type	unsigned 32
Access mode	rw
Default value	see chapter 8.9.1 (page 51)

Structure of the parameter:

Bit-No.	Value	Meaning
31 (MSB)	-	do not care
30	0/1	0: Device does not generate SYNC message 1: Device generates SYNC message
29	0	always 0 (11-bit ID)
28...11	0	always 0 (29-bit IDs are not supported)
10...0 (LSB)	x	Bit 0...10 of the SYNC-COB-ID

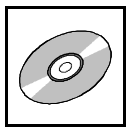
The identifier can take values between $0...7FF_h$.

8.9.6 Communication Cycle Period (1006_h)

INDEX	**1006_h**
Name	*Communication Cylcle Period*
Data type	unsigned 32
Access mode	rw
Default value	0 µs

Value range of the parameter:

Value	Meaning
0	No transmission of SYNC messages
1...$FFFFFFFF_h$	Cycle time in microseconds



8.9.7 Manufacturer Device Name (1008_h)

INDEX	**1008_h**
Name	*manufacturer device name*
Data type	visible string
Default value	see chapter 8.9.1 (page 51)

For detailed description of the SDO Uploads, please refer to [1].

8.9.8 Manufacturer Hardware Version (1009_h)

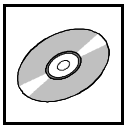
INDEX	**1009_h**
Name	*manufacturer hardware version*
Data type	visible string
Default value	string: e.g. ‘1.00’ (depending on version)

The hardware version is read similarly to reading the manufacturer’s device name via the domain upload protocol. Please refer to [1] for a detailed description of the upload.

8.9.9 Manufacturer Software Version ($100A_h$)

INDEX	**$100A_h$**
Name	*manufacturer software version*
Data type	visible string
Default value	string: e.g.: ‘1.2’ (depending on version)

Reading the software version is similar to reading the manufacturer’s device name via the domain upload protocol. Please refer to [1] for a detailed description of the upload.



8.9.10 Guard Time ($100C_h$) und Life Time Factor ($100D_h$)

The CAN-CBX module supports the node guarding or alternatively the heartbeat function (see page 71).

Note:
By the recommendation of the CiA, the heartbeat-function shall be used preferentially. Use the node-guarding only for existing systems and not for new developments!

Guard time and life time factors are evaluated together. Multiplying both values will give you the life time. The guard time is represented in milliseconds.

INDEX	**$100C_h$**
Name	*guard time*
Data type	unsigned 16
Access mode	rw
Default value	0 [ms]
Minimum value	0
Maximum value	$FFFF_h$ (65.535 s)

INDEX	**$100D_h$**
Name	*life time factor*
Data type	unsigned 8
Access mode	rw
Default value	0
Minimum value	0
Maximum value	FF_h

8.9.11 Node Guarding Identifier ($100E_h$)

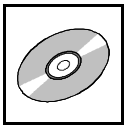
The module only supports 11-bit identifiers.

INDEX	**$100E_h$**
Name	*node guarding identifier*
Data type	unsigned 32
Access mode	rw
Default value	700_h + Node-ID

Structure of the parameter *node guarding identifier* :

Bit-No.	Meaning
31 (MSB) 30	reserved
29...11	always 0, because 29-bit-IDs are not supported
10...0 (LSB)	bit 0...10 of the node guarding identifier

The identifier can take values between $1...7FF_h$.



8.9.12 Store Parameters (1010_h)

This object supports saving of parameters to a non-volatile memory, the EEPROM here.
Therefore the parameter groups shown below are distinguished.

After they are transferred, the parameters are immediately active.
The non-volatile storage of the parameters however is not carried out automatically. It must be initiated with a write access to object 1010_h and should only be carried out if the module is in the state *pre-operational*.
In order to avoid storage of parameters by mistake, storage is only executed when the specific signature as shown below is transmitted.

Reading the index returns information about the implemented storage functionality (refer to [1] for more information).

INDEX	**1010_h**
Name	*store parameters*
Data type	unsigned 32

Index	Sub-index	Description	Value range	Data type	Access mode
1010_h	0	*number_of_entries*	4	unsigned 8	ro
	1	*save_all_parameters* (objects 1000_h ... $9FFF_h$)	no default, write: 65 76 61 73_h (= ASCII: ‘e’ ‘v’ ‘a’ ‘s’)	unsigned 32	rw
	2	*save_communication_parameter* (objects 1000_h ... $1FFF_h$)		unsigned 32	rw
	3	*save_application_parameter* (objects 6000_h ... $9FFF_h$)		unsigned 32	rw
	4	*save_manufacturer_parameter* (objects 2000_h ... $5FFF_h$)		unsigned 32	rw

Assignment of the variables

save all parameters

saves the parameters of all objects (if available), which have a read/write (rw) right of access.

save_communication_parameter

saves all communication parameters of those objects (objects 1000_h ... $1FFF_h$, if available), which have a read/write (rw) right of access (here e.g. 1005_h ... 1029_h).

save_application_parameter

saves all application parameters of those objects (objekte 6000_h ... $9FFF_h$, if available), which have a read/write (rw) right of access (here e.g. $6xxx_h$).

save_manufacturer_parameter

saves all manufacturer parameters of those objects (objects 2000_h ... $5FFF_h$, if available), which have a read/write (rw) right of access (here e.g. $2xxx_h$).

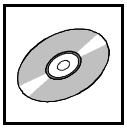
The storage mode is shown in the content of this object:
Bit 1 of object 1010_h, sub-index 1 is not set, i.e the CAN-CBX-module does not save the configuration automatically. The storage must be initiated by writing the character string 'save' (73_h 61_h 76_h 65_h, order from CAN telegram) to object 1010_h, sub-index 1-4.

On read access to the appropriate sub-index, the CAN-CBX module provides information about its storage functionality with the format described in the following:

Bit:	31 2	1	0
Inhalt:	reserved	auto	cmd
	0	0	1
	MSB		LSB

Bit	Value	Description
auto	10	CAN-CBX module does **not** save the parameters autonomously CAN-CBX module **saves** the parameters autonomously
cmd	10	CAN-CBX module does **not** save the parameters on command CAN-CBX module **saves** the parameters on command

Autonomous saving means that the CAN-CBX module stores the storable parameters non-volatile and without a user request.



8.9.13 Restore Default Parameters (1011_h)

Via this command the default parameters, valid when leaving the manufacturer, are restored.
Therefor the parameter groups described below are distinguished.
Every individual setting stored in the EEPROM will be lost.
After a reset the default parameters will be active. The reset of the parameters however must be initiated with a write access to object 1011_h. To write the index a specific signature as shown below has to be transmitted.
Reading the index provides information about its parameter restoring capability (refer to [1] for more information).

INDEX	**1011_h**
Name	*restore default parameters*
Data Type	unsigned 32

Index	Sub-index	Description	Value range	Data type	Access mode
1011_h	0	*number_of_entries*	4	unsigned 8	ro
	1	*restore_all_default_parameters* (objects 1000_h ... $9FFF_h$)	no default, write: 64 61 6F $6C_h$ (= ASCII: ‘d’ ‘a’ ‘o’ ‘l’)	unsigned 32	rw
	2	*restore_communication_parameter* (objects 1000_h ... $1FFF_h$)		unsigned 32	rw
	3	*restore_application_parameter* (objects 6000_h ... $9FFF_h$)		unsigned 32	rw
	4	*restore_manufacturer_parameter* (objects 2000_h ... $5FFF_h$)		unsigned 32	rw

Assignment of the variables

restore all parameters

restores the default parameters of all objects (if available), which have a read/write (rw) right of access.

restore_communication_parameter

restores all communication default parameters of those objects (objects 1000_h ... $1FFF_h$, if available, here e.g. 1005_h ... 1029_h).

restore_application_parameter

restoers all application default parameters of those objects (objekts 6000_h ... $9FFF_h$, if available, here e.g. $6xxx_h$).

restore_manufacturer_parameter

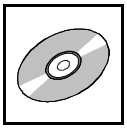
loads all manufacturer default parameters of those objects (objects 2000_h ... $5FFF_h$, if available, here e.g. $2xxx_h$).

Bit 0 of object 1011_h, sub-index 1 is set, i.e. the CAN-CBX module restores the default values initiated by writing the signature 'load' (64_h 61_h $6F_h$ $6C_h$, sequence in CAN telegram) in object 1011_h, sub-index 1-4.

On read access to the appropriate sub-index, the CANopen device provides information about its default parameter restoring capability with the following format:

Bit:	31 ... 1	0
Content:	reserved	cmd
	0	1
	MSB	LSB

Bit	Value	Description
cmd	1	the CAN-CBX-module does **not** restore default parameters the CAN-CBX-module **restores** the default parameters



8.9.14 COB_ID Emergency Message (1014_h)

INDEX	**1014_h**
Name	*COB-ID emergency object*
Data type	unsigned 32
Default value	80_h + Node-ID

This object defines the COB-ID of the emergency object (EMCY).

The structure of this object is shown in the following table:

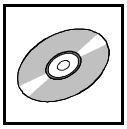
Bit-No.	Value	Meaning
31 (MSB)	0/1	0: EMCY exists / is valid 1: EMCY does not exist / EMCY is not valid
30	0	reserved (always 0)
29	0	always 0 (11-bit ID)
28...11	0	always 0 (29-bit IDs are not supported)
10...0 (LSB)	x	bits 0...10 of COB-ID

The identifier can take values between $0...7FF_h$.

8.9.15 Inhibit Time EMCY (1015_h)

INDEX	**1015_h**
Name	*inhibit_time_emergency*
Data type	unsigned 16
Access mode	rw
Value range	0...$FFFF_h$
Default value	0

The *Inhibit Time* for the EMCY message can be defined with this entry. The time is determined as a multiple of 100 µs.



8.9.16 Consumer Heartbeat Time (1016_h)

INDEX	**1016_h**
Name	*consumer heartbeat time*
Data type	unsigned 32
Default value	No

The heartbeat function can be used for mutual monitoring of the CANopen modules (especially to detect connection failures). Unlike node guarding/life guarding the heartbeat function does not require RTR-Frames.

Function:
A module, the so-called heartbeat producer, cyclically transmits a heartbeat message on the CAN-bus on the node-guarding identifier (see object $100E_h$). One or more heartbeat consumers receive the message. It has to be received within the heartbeat time stored on the heartbeat consumer, otherwise a heartbeat event is triggered on the heartbeat-consumer module. A heartbeat event generates a heartbeat error on the CAN-CBX module.

Each module can act as a heartbeat producer and a heartbeat consumer. The CAN-CBX module can represent at most one heartbeat consumer per CAN net.

Index	Sub-index	Description	Value range	Default	Data type	Access mode
1016_h	0	*number_of_entries*	1	1	unsigned 8	ro
	1	*consumer_heartbeat_time*	0...$007FFFFF_h$	0	unsigned 32	rw

Meaning of the variable *consumer-heartbeat_time_x*:

	consumer-heartbeat_time_x		
Bit	31 24	23 16	15 0
Assignment	reserved (always '0')	*Node-ID* (unsigned 8)	*heartbeat_time* (unsigned 16)

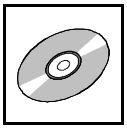
Node-ID Node-Id of the heartbeat producer to be monitored.

heartbeat_time Within this time [ms] the heartbeat producer has to transmit the heartbeat on the node-guarding ID, to avoid the transmission of a heartbeat event.
The consumer-heartbeat time of the monitoring module must always be higher than the producer-heartbeat time of the heartbeat-transmitting module.

Example:

consumer-heartbeat_time = 0031 $03E8_h$

=> *Node-ID* = 31_h = 49_d
=> *heartbeat time* = $3E8_h$ = 1000_d => 1 s



8.9.17 Producer Heartbeat Time (1017_h)

INDEX	**1017_h**
Name	*producer heartbeat time*
Data type	unsigned 16
Default value	0 ms

The producer heartbeat time defines the cycle time with which the CAN-CBX- module transmits a heartbeat-frame to the node-guarding ID.

If the value of the producer heartbeat time is higher than ‘0’, it is active and stops the node-/ life-guarding (see page 61).
If the value of the producer-heartbeat-time is set to ‘0’, transmitting heartbeats by this module is stopped.

Index	Sub-index	Description	Value range	Default	Data type	Access mode
1017_h	0	*producer-heartbeat_time*	0...$FFFF_h$	0 ms	unsigned 16	rw

producer-heartbeat_time Cycle time [ms] of heartbeat producer to transmit the heartbeat on the node-guarding ID (see object $100E_h$).
The consumer-heartbeat time of the monitoring module must always be higher than the producer-heartbeat time of the heartbeat-transmitting module.

8.9.18 Identity Object (1018_h)

INDEX	**1018_h**
Name	*identity object*
Data type	unsigned 32
Default value	No

This object contains general information to the CAN module.

Index	Sub-index	Description	Value range	Default	Data type	Access mode
1018_h	0	*no_of_entries*	4	4	unsigned 8	ro
	1	*vendor_id*	0...FFFFFFFF$_h$	0000 0017$_h$	unsigned 32	ro
	2	*product_code*	0...FFFFFFFF$_h$	see chapter 8.9.1 (page 51)	unsigned 32	ro
	3	*revision_number*	0...FFFFFFFF$_h$	0	unsigned 32	ro
	4	*serial_number*	0...FFFFFFFF$_h$	-	unsigned 32	ro

Description of the variables:

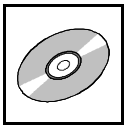
vendor_id This variable contains the esd-vendor-ID. This is always 00000017$_h$.

product_code Here the esd-article number of the product is stored.
The nibbles of the long words have the following meaning:

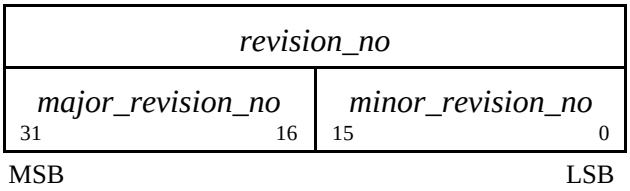
product_code = `abcd efgh`$_h$

`a`: 1... article number beginning with character “K”
2....article number beginning with character “C”
`bcde`: 4-digit hex number, which is interpreted as the integer part of the decimal number (on the left of the decimal point).
`f`: currently not evaluated
`gh`: 2-digit hex number, which is interpreted an fraction part of the decimal number (on the right of the decimal point).

Example: ‘2303 2002$_h$’ corresponds to article number ‘C.3020.02’.



revision_number Here the software version is stored. In accordance with [1] the two MSB represent the revision numbers of the major changes and the two LSB show the revision number of minor corrections or changes.



serial_number Here the serial number of the hardware is read. The first two characters of the serial number are letters which designate the manufacturing lot. The following characters represent the actual serial number.

In the two MSB of *serial_no* the letters of the manufacturing lot are coded. They each contain the ASCII-code of the letter with the MSB set '1' in order to be able to differentiate between letters and numbers:
(ASCII-Code) + 80_h = read_byte

The two last significant bytes contain the number of the module as BCD-value.

Example:
If the value 'C1C2 0105_h' is being read, this corresponds to the hardware-serial number code 'AB 0105'. This value has to correspond to the serial number of the module.

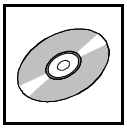
8.9.19 Synchronous Counter Overflow Value (1019_h)

INDEX	**1019_h**
Name	*Synchronous_Counter_Overflow*
Data type	unsigned 8
Default value	0

This object defines whether a counter is mapped into the SYNC message or not and further the highest value the counter can reach.

The value range of the object is described in the following table:

Value	Description
0	The SYNC message shall be transmitted as a CAN message of data length ‘0’.
1	reserved
2...240	The SYNC message shall be transmitted as a CAN message of data length ‘1’. The first data byte contains the counter.
241...255	reserved



8.9.20 Verify Configuration (1020_h)

INDEX	**1020_h**
Name	*verify configuration*
Data type	unsigned 32
Default value	No

In this object the date and the time of the last configuration can be stored to check later whether the configuration complies with the expected configuration or not.
The content of the parameters is not evaluated by the firmware.

Index	Sub-index	Description	Value range	Default	Data type	Access mode
1020_h	0	*no_of_entries*	2	2	unsigned 8	ro
	1	*configuration_date*	0...$FFFFFFFF_h$	0	unsigned 32	rw
	2	*configuration_time*	0...$FFFFFFFF_h$	0	unsigned 32	rw

Parameter Description:

configuration_date Date of the last configuration of the module. The value is defined in number of days since the 01.01.1984.

configuration_time Time in ms since midnight at the day of the last configuration.

8.9.21 Error Behaviour Object (1029_h)

INDEX	**1029_h**
Name	*error behaviour object*
Data type	unsigned 8
Default value	No

If an error event occurs (such as heartbeat error), the module changes into the status which has been defined in variable *communication_error* or *output_error*.

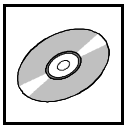
Index	Sub-index	Description	Value range	Default	Data type	Access mode
1029_h	0	*no_of_error_classes*	1	1	unsigned 8	ro
	1	*communication_error*	0...2	0	unsigned 8	rw

Meaning of the variables:

Variable	Meaning
no_of_error_classes	number of error-classes (here always ‘1’)
communication_error	heartbeat/lifeguard error and *Bus off*

The module can enter the following states if an error occurs.

Variable	Module state
0	pre-operational (only if the current state is operational)
1	no state change
2	stopped



8.9.22 NMT Startup ($1F80_h$)

INDEX	**$1F80_h$**
Name	*NMT startup*
Data type	unsigned 32
Default value	2

The NMT startup is implemented to be able to start CANopen nodes in environments without NMT-master.
Via NMT startup the auto startup of a CANopen node can be switched on or off.
Further features of the parameters *NMT startup* are currently not supported.

The value range of the object is described in the following table:

Value [Hex]	Meaning
$0000\ 0002_h$	Auto startup disabled (default)
$0000\ 0008_h$	Auto startup enabled
alle other values	reserved

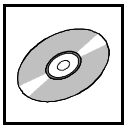
8.9.23 Self Starting Nodes Timing Parameters ($1F91_h$)

INDEX	**$1F91_h$**
Name	*Self starting nodes timing parameters*
Data type	unsigned 16

Index	Sub-index	Description	Value range	Default	Data type	Access mode
$1F91_h$	0	*number_of_entries*	1	1	unsigned 8	ro
	1	*NMT master detection timeout*	0...$FFFF_h$	64_h	unsigned 16	rw

Sub-index 1 of this object contains the timeout in [ms] between the change from “preoperational” > “operational”. In default it is 100 ms.

The sub-indices 2 and 3 of this object are not supported.



8.9.24 Object Transmit PDO Communication Parameter (1801_h, 1802_h)

This objects define the parameters of the transmit-PDOs.

INDEX	**1801_h 1802_h**
Name	*transmit PDO parameter*
Data Type	PDOCommPar

Index	Sub-index	Description	Value range	Default	Data type	Access mode
1801_h	0	*number_of_entries*	0...FF_h	05	unsigned 8	ro
	1	*COB-ID used by PDO*	1...800007FF_h	280_h +Node-ID	unsigned 32	rw
	2	*transmission type*	0...FF_h	FF_h	unsigned 8	rw
	3	*inhibit time*	0...$FFFF_h$	0	unsigned 16	rw
	4	*reserved*	0..FF_h	0	unsigned 8	const
	5	*event timer*	0...$FFFF_h$	0	unsigned 16	rw
1802_h	0	*number_of_entries*	0...FF_h	05	unsigned 8	ro
	1	*COB-ID used by PDO*	1...800007FF_h	380_h +Node-ID	unsigned 32	rw
	2	*transmission type*	0...FF_h	FF_h	unsigned 8	rw
	3	*inhibit time*	0...$FFFF_h$	0	unsigned 16	rw
	4	*reserved*	0..FF_h	0	unsigned 8	const
	5	*event timer*	0...$FFFF_h$	0	unsigned 16	rw

The *transmission types* 0, 1...240 and 252...255 are supported.

8.9.25 Transmit PDO Mapping Parameter ($1A01_h$, $1A02_h$)

This objects define the assignment of the transmit data to the Tx-PDOs.

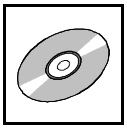
INDEX	**$1A01_h$, $1A02_h$**
Name	*transmit PDO mapping*
Data Type	PDO Mapping

The following table shows the assignment of the transmit PDO mapping parameters:

Index	Sub-index	Description	Value range	Default	Data type	Access mode
$1A01_h$	0	*number of entries*	0...FF_h	4	unsigned 8	rw
	1	*Read_Analog_Input_16_1*	0...$FFFFFFFF_h$	6401 01 10_h	unsigned 32	rw
	2	*Read_Analog_Input_16_2*	0...$FFFFFFFF_h$	6401 02 10_h	unsigned 32	rw
	3	*Read_Analog_Input_16_3*	0...$FFFFFFFF_h$	6401 03 10_h	unsigned 32	rw
	4	*Read_Analog_Input_16_4*	0...$FFFFFFFF_h$	6401 04 10_h	unsigned 32	rw
$1A02_h$	0	*number of entries*	0...FF_h	4	unsigned 8	rw
	1	*Read_Analog_Input_16_5*	0...$FFFFFFFF_h$	6401 05 10_h	unsigned 32	rw
	2	*Read_Analog_Input_16_6*	0...$FFFFFFFF_h$	6401 06 10_h	unsigned 32	rw
	3	*Read_Analog_Input_16_7*	0...$FFFFFFFF_h$	6401 07 10_h	unsigned 32	rw
	4	*Read_Analog_Input_16_8*	0...$FFFFFFFF_h$	6401 08 10_h	unsigned 32	rw

Note:
The local firmware allows every TxPDO-mapping of the objects listed below, i.e. combinations of 16-bit- and 32-bit-A/D-values in one frame are also possible.

The following objects can be mapped: 6401_h,x
6402_h,x
6404_h,x



8.10 Device Profile Area

8.10.1 Overview of the Implemented Objects 6401_h ...6426_h

Index	Name	Data Type
6401_h	*Read Analog Inputs 16-Bit*	integer 16
6402_h	*Read Analog Inputs 32-Bit*	integer 32
6404_h	*Read Analog Inputs Raw Data 16-Bit*	integer 16
6421_h	*Analog Interrupt Trigger Selection*	unsigned 8
6423_h	*Analog Input Global Interrupt Enable*	boolean
6424_h	*Analog Input Interrupt Upper Limit*	integer 32
6425_h	*Analog Input Interrupt Lower Limit*	integer 32
6426_h	*Analog Input Interrupt Delta Unsigned*	unsigned 32

8.10.2 Relationship Between the Implemented Analog Input Objects

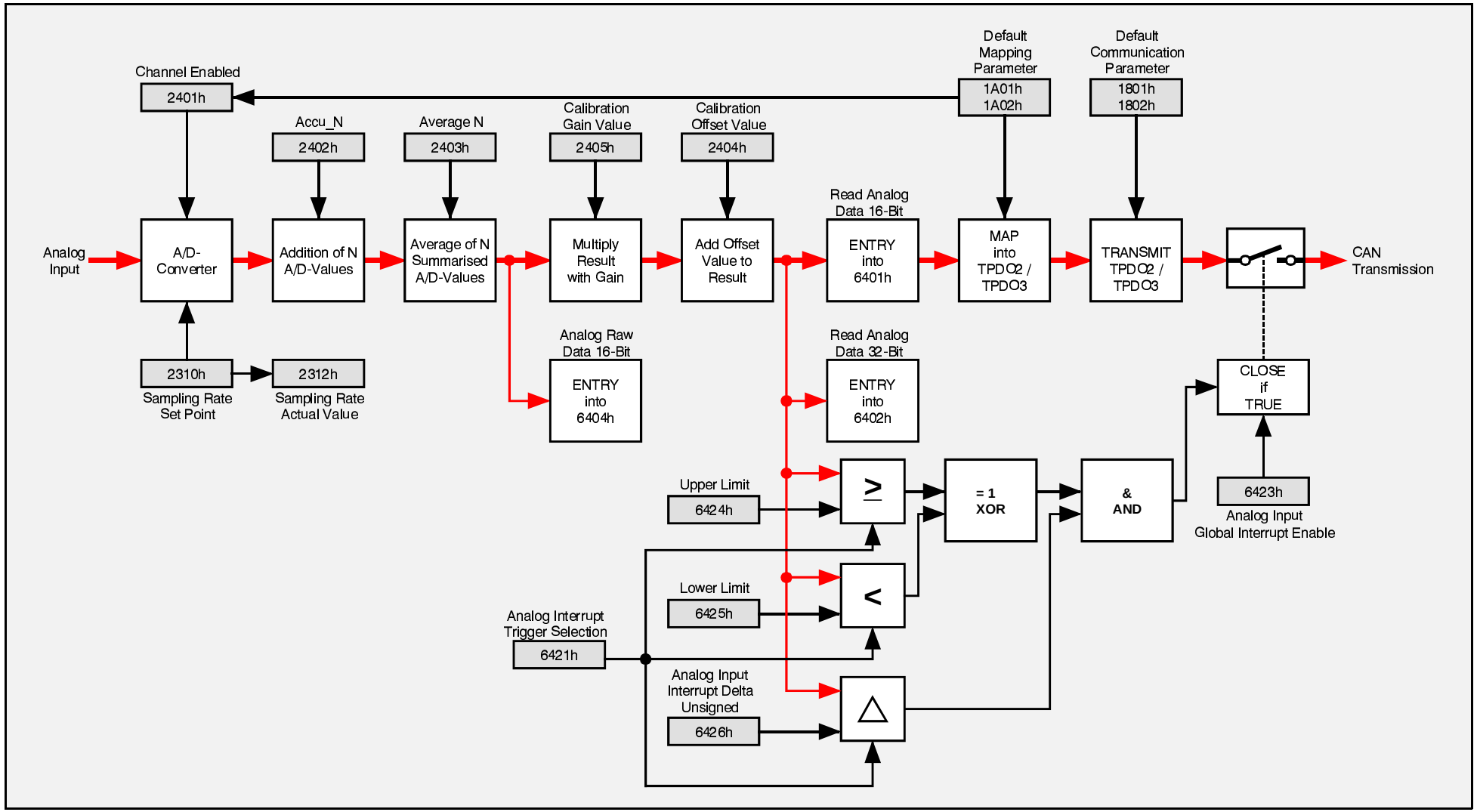
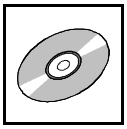


Fig. 20: Relationship between the implemented objects



8.10.3 Read Input 16-Bit (6401_h)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
6401_h	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Read_Analog_Input_16_1*	8000_h...$7FFF_h$	-	integer 16	ro
	2	*Read_Analog_Input_16_2*	8000_h...$7FFF_h$	-	integer 16	ro
	3	*Read_Analog_Input_16_3*	8000_h...$7FFF_h$	-	integer 16	ro
	4	*Read_Analog_Input_16_4*	8000_h...$7FFF_h$	-	integer 16	ro
	5	*Read_Analog_Input_16_5*	8000_h...$7FFF_h$	-	integer 16	ro
	6	*Read_Analog_Input_16_6*	8000_h...$7FFF_h$	-	integer 16	ro
	7	*Read_Analog_Input_16_7*	8000_h...$7FFF_h$	-	integer 16	ro
	8	*Read_Analog_Input_16_8*	8000_h...$7FFF_h$	-	integer 16	ro

Assignment of the variable *Read_Analog_Input_16_x* (x = 1...8):
The last digit of the name of the variable is the number of the respective analog input channel. The data bits are shifted left-aligned in two's complement representation in the 16-bit variable.

Calculation of the measured voltage:
The measured voltage is calculated of the value gained by addition and averaging of the variable *Read_Analog_Input_16_x* (see also diagram on page 82):

Value of the variable *Read_Analog_Input_16_x*																Hexadecimal	Measured voltage
Binary (bit 15...0)																	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000_h	-10.24 V
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$FFFF_h$	-0.3125 mV
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000_h	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001_h	+0.3125 mV
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$7FFF_h$	+10.24 V - (1 $LSB_{VARIABLE}$) = 10.2397 V

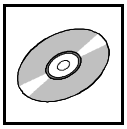
Resolution of the measured value:
1 LSB based on the variable *Read_Analog_Input_16_x* (x = 1...8):
1 $LSB_{VARIABLE}$ => 10.24 V / 8000_h => 0.3125 mV

The returned value has been internally calculated by the following formula:

Read_Analog_Input_16_x = Offset_x + (Read_Analog_In_Raw_16_x · Gain_x)			
from:	Objekt 6401_h/6402_h	Obj. 2404_h	Obj. 6404_h	Obj.2405_h

The *AD-value* is the value, which results from addition and averaging (see figure on page 82). The *AD-value* is then equivalent to the 'Read Analog Input Raw Data 16-Bit'-value (object 6404_h).

Offset is defined in the objects *Calibration Offset Value* (object 2404_h, see page 101) and *Gain* is defined in the objects *Calibration Gain Value* (object 2405_h, see page 102).



8.10.4 Read Input 32-Bit (6402_h)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
6402_h	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Read_Analog_Input_32_1*	80000000_h ... $7FFFFFFF_h$	-	integer 32	ro
	2	*Read_Analog_Input_32_2*	80000000_h ... $7FFFFFFF_h$	-	integer 32	ro
	3	*Read_Analog_Input_32_3*	80000000_h ... $7FFFFFFF_h$	-	integer 32	ro
	4	*Read_Analog_Input_32_4*	80000000_h ... $7FFFFFFF_h$	-	integer 32	ro
	5	*Read_Analog_Input_32_5*	80000000_h ... $7FFFFFFF_h$	-	integer 32	ro
	6	*Read_Analog_Input_32_6*	80000000_h ... $7FFFFFFF_h$	-	integer 32	ro
	7	*Read_Analog_Input_32_7*	80000000_h ... $7FFFFFFF_h$	-	integer 32	ro
	8	*Read_Analog_Input_32_8*	80000000_h ... $7FFFFFFF_h$	-	integer 32	ro

Assignment of the variable *Read_Analog_Input_32_x* (x = 1...8):
The last digit of the name of the variable is the number of the respective analog input channel.
The data bits are shifted left-aligned in two's complement representation in the 32-bit variable.

Calculation of the measured voltage:
The measured voltage is calculated of the value gained by addition and averaging of the variable *Read_Analog_Input_32_x* (see also diagram on page 82):

Value of the variable *Read_Analog_Input_32_x*																Measured voltage (object2402_h=3 object 2403_h=4)
Binary (bit 31...0)																Hexa-decimal
31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0									
1 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	80000000_h	-10.24 V
: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	:	:
1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	$FFFFFFFF_h$	-4.7684 nV
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	00000000_h	0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	00000001_h	+4.7684 nV
: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	:	:
0 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	$7FFFFFFF_h$	+10.24 V -(1 $LSB_{VARIABLE}$)

Resolution of the measured value:
1 LSB based on the variable *Read_Analog_Input_32_x* (x = 1...8):
1 $LSB_{VARIABLE}$ => 10.24 V / 8000 0000_h => 4.7684 nV

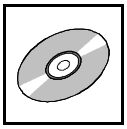
The returned value has been internally calculated by the following formula:

Read_Analog_Input_32_x = Offset_x + (Read_Analog_In_Raw_16_x · Gain_x)

from:	Objekt 6401_h/6402_h	Obj. 2404_h	Obj. 6404_h	Obj.2405_h

The *AD-value* is the value, which results from addition and averaging (see figure on page 82). The *AD-value* is then equivalent to the 'Read Analog Input Raw Data 16-Bit'-value (object 6404_h).

Offset is defined in the objects *Calibration Offset Value* (object 2404_h, see page 101) and *Gain* is defined in the objects *Calibration Gain Value* (object 2405_h, see page 102).



8.10.5 Read Input Raw Data 16-Bit (6404_h)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
6404_h	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Read_Analog_In_Raw_16_1*	8000_h...$7FFF_h$	-	integer 16	ro
	2	*Read_Analog_In_Raw_16_2*	8000_h...$7FFF_h$	-	integer 16	ro
	3	*Read_Analog_In_Raw_16_3*	8000_h...$7FFF_h$	-	integer 16	ro
	4	*Read_Analog_In_Raw_16_4*	8000_h...$7FFF_h$	-	integer 16	ro
	5	*Read_Analog_In_Raw_16_5*	8000_h...$7FFF_h$	-	integer 16	ro
	6	*Read_Analog_In_Raw_16_6*	8000_h...$7FFF_h$	-	integer 16	ro
	7	*Read_Analog_In_Raw_16_7*	8000_h...$7FFF_h$	-	integer 16	ro
	8	*Read_Analog_In_Raw_16_8*	8000_h...$7FFF_h$	-	integer 16	ro

Assignment of the variable *Read_Analog_In_Raw_16_x* (x = 1...8):
The last digit of the name of the variable is the number of the respective analog input channel.
The data bits are shifted left-aligned in the 16-bit variable.

Calculation of the measured voltage:
The measured voltage is calculated of the value gained by addition and averaging of the variable *Read_Analog_In_Raw_16_x* (see also diagram on page 82):

Value of the variable *Read_Analog_In_Raw_16_x*																Hexadecimal	Measured voltage (object 2402_h = 3, object 2403_h = 4)
Binary (bit 15...0)																	
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8000_h	-10.24 V
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$FFFF_h$	-0.3125 mV
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0000_h	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0001_h	+0.3125 mV
:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:	:
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	$7FFF_h$	+10.24 V - (1 $LSB_{VARIABLE}$) = 10.2397 V

Resolution of the measured values:
1 LSB based on the variable *Read_Analog_Input_16_x* (x = 1...8):
1 $LSB_{VARIABLE}$ => 10.24 V / 8000_h => 0.3125 mV

8.10.6 Analog Interrupt Trigger Selection (6421_h)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
6421_h	0	*Number of analog inputs*	8	8	unsigned 8	ro
	1	*Analog_Input_IRQ_Trigger_1*	0 ... 7	7	unsigned 8	rw
	2	*Analog_Input_IRQ_Trigger_2*	0 ... 7	7	unsigned 8	rw
	3	*Analog_Input_IRQ_Trigger_3*	0 ... 7	7	unsigned 8	rw
	4	*Analog_Input_IRQ_Trigger_4*	0 ... 7	7	unsigned 8	rw
	5	*Analog_Input_IRQ_Trigger_5*	0 ... 7	7	unsigned 8	rw
	6	*Analog_Input_IRQ_Trigger_6*	0 ... 7	7	unsigned 8	rw
	7	*Analog_Input_IRQ_Trigger_7*	0 ... 7	7	unsigned 8	rw
	8	*Analog_Input_IRQ_Trigger_8*	0 ... 7	7	unsigned 8	rw

This object defines the interrupt trigger condition.

Assignment of the variable *Analog_Input_IRQ_Trigger_x* (x = 1...8):

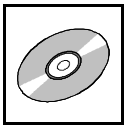
Bit:	7	6	5	4	3	2	1	0
Meaning:	reserved for future applications			not supported		**Change of value higher than Delta (6426_h)**	**Input below Lower Limit (6425_h)**	**Upper Limit exceeded (6424_h)**

Value range:

Analog_Input_IRQ_Trigger_x = 00 => no interrupt trigger

Analog_Input_IRQ_Trigger_x = 03 => Interrupt Upper Limit (object 6424_h) and Interrupt Lower Limit (object 6425_h) enabled.

Analog_Input_IRQ_Trigger_x = 04 => interrupt trigger, if the change of the A/D-value is higher than the value (Delta) defined in object 6426_h.



8.10.7 Global Interrupt Enable (6423_h)

Index	Sub-index	Description	Value range [Boolean]	Default [Boolean]	Data type	Access mode
6423_h	0	*Analog_Input_Global_Interrupt_Enable*	true, false	false	boolean	rw

With the object *Analog Input Global Interrupt Enable* the interrupt-function of the module is enabled or disabled. The values in object 6421_h and 6426_h remain unaffected of this.

In the default-setting the interrupts, triggered by a change of the analog input value, are disabled.

Value range:

Analog_Input_Global_Interrupt_Enable = true => Global Interrupt enabled
Analog_Input_Global_Interrupt_Enable = false => Global Interrupt disabled (default setting)

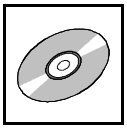
Note:
To enable an event controlled transmission of the TxPDO it is absolutely necessary to set *Analog_Input_Global_Interrupt_Enable* to ‘true’.

8.10.8 Interrupt Upper Limit (6424_h)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
6424_h	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Analog_Input_Interrupt_Upper_Limit_1*	80000000_h ... $7FFF0000_h$	0	integer 32	rw
	2	*Analog_Input_Interrupt_Upper_Limit_2*	80000000_h ... $7FFF0000_h$	0	integer 32	rw
	3	*Analog_Input_Interrupt_Upper_Limit_3*	80000000_h ... $7FFF0000_h$	0	integer 32	rw
	4	*Analog_Input_Interrupt_Upper_Limit_4*	80000000_h ... $7FFF0000_h$	0	integer 32	rw
	5	*Analog_Input_Interrupt_Upper_Limit_5*	80000000_h ... $7FFF0000_h$	0	integer 32	rw
	6	*Analog_Input_Interrupt_Upper_Limit_6*	80000000_h ... $7FFF0000_h$	0	integer 32	rw
	7	*Analog_Input_Interrupt_Upper_Limit_7*	80000000_h ... $7FFF0000_h$	0	integer 32	rw
	8	*Analog_Input_Interrupt_Upper_Limit_8*	80000000_h ... $7FFF0000_h$	0	integer 32	rw

In object *Analog_Input_Interrupt_Upper_Limit* an upper limit is defined for the interrupt function.

Note:
Only the 16 higher-order bits of the 32-bit-variables *Analog_Input_Interrupt_Upper_Limit_x* are evaluated.
Thus the minimum value is $0001\ 0000_h$, i.e. 312,5µV. The lower-order bits are cut off when written.



8.10.9 Interrupt Lower Limit (6425_h)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
6425_h	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Analog_Input_Interrupt_Lower_Limit_1*	80000000_h ... $7FFF0000_h$	0	integer 32	rw
	2	*Analog_Input_Interrupt_Lower_Limit_2*	80000000_h ... $7FFF0000_h$	0	integer 32	rw
	3	*Analog_Input_Interrupt_Lower_Limit_3*	80000000_h ... $7FFF0000_h$	0	integer 32	rw
	4	*Analog_Input_Interrupt_Lower_Limit_4*	80000000_h ... $7FFF0000_h$	0	integer 32	rw
	5	*Analog_Input_Interrupt_Lower_Limit_5*	80000000_h ... $7FFF0000_h$	0	integer 32	rw
	6	*Analog_Input_Interrupt_Lower_Limit_6*	80000000_h ... $7FFF0000_h$	0	integer 32	rw
	7	*Analog_Input_Interrupt_Lower_Limit_7*	80000000_h ... $7FFF0000_h$	0	integer 32	rw
	8	*Analog_Input_Interrupt_Lower_Limit_8*	80000000_h ... $7FFF0000_h$	0	integer 32	rw

In object *Analog_Input_Interrupt_Lower_Limit* a lower limit is defined for the interrupt function.

Note:
Only the 16 higher-order bits of the 32-bit-variables *Analog_Input_Interrupt_Upper_Limit_x* are evaluated. Thus the minimum value is $0001\ 0000_h$, i.e. 312,5µV.
The lower-order bits are cut off when written.

8.10.10 Analog Input Interrupt Delta (6426_h)

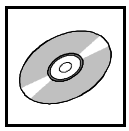
Index	Sub-index	Description	Value range	Default	Data type	Access mode
6426_h	0	*Number_Analog_Inputs*	8	8	unsigned 8	ro
	1	*Analog_Input_IRQ_Delta_1*	0...FFFF0000$_{hh}$	0	unsigned 32	rw
	2	*Analog_Input_IRQ_Delta_2*	0...FFFF0000$_h$	0	unsigned 32	rw
	3	*Analog_Input_IRQ_Delta_3*	0...FFFF0000$_h$	0	unsigned 32	rw
	4	*Analog_Input_IRQ_Delta_4*	0...FFFF0000$_h$	0	unsigned 32	rw
	5	*Analog_Input_IRQ_Delta_5*	0...FFFF0000$_h$	0	unsigned 32	rw
	6	*Analog_Input_IRQ_Delta_6*	0...FFFF0000$_h$	0	unsigned 32	rw
	7	*Analog_Input_IRQ_Delta_7*	0...FFFF0000$_h$	0	unsigned 32	rw
	8	*Analog_Input_IRQ_Delta_8*	0...FFFF0000$_h$	0	unsigned 32	rw

In this object the difference value of the analog input voltage, which can be evaluated for triggering the interrupt, is defined. Via object 6421_h the type of evaluation is selected. The interrupt function is enabled via object 6423_h.

Positive and negative changes of the voltage are evaluated.
The value *Analog_Input_IRQ_Delta_x* is always the absolute value of the difference between the current AD value less the last preceding AD value.
The difference of the AD values is contained in object 6426_h always as a positive value.

Provided that the objects 6421_h and 6423_h are configured accordingly, the interrupt will be triggered if the difference of the AD values equals or exceeds the value *Analog_Input_Interrupt_IRQ_Delta_x*. Per default this value is '0'. Thus the interrupt is always triggered, because the difference is always ≥ 0.

Note:
Only the 16 higher-order bits of the 32-bit-variables *Analog_Input_IRQ_Delta_x* are evaluated. Thus the minimum value is 0001 0000$_h$, i.e. 312.5µV.



Value of the variable *Analog_Input_IRQ_Delta_x*	Change of the voltage	Description
0001 0000_h	312.5 µV	minimum value
:	:	-
FFFF $FFFF_h$	+20.48 V	maximum value
0066 0000_h	31.875 mV	example
3400 0000_h	4.16 V	example

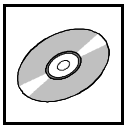
Note:
For the voltage difference calculation the objects 6401_h, or 6402_h are evaluated, which are calculated from gain, offset, Accu_N (object 2402_h) and Average_N (object 2403_h).

It can be expedient, to use all of the upper 16 bits of the 32-bit variable *Analog_Input_IRQ_Delta_x*, because resolutions higher than 14 bit can be achieved by the addition of the measured values.

8.11 Manufacturer Specific Profile Area

8.11.1 Overview of Manufacturer Specifc Objects 2310_h ... 2405_h

Index	Name	Data Type
2310_h	*Sample Time Set Point*	unsigned 16
2312_h	*Sample Time Actual Value*	unsigned 16
2401_h	*Channel Enabled*	unsigned 8
2402_h	*Accu_N*	unsigned 8
2403_h	*Average_N*	unsigned 8
2404_h	*Calibration Offset Value*	integer 32
2405_h	*Calibration Gain Value*	integer 16



8.11.2 Sample Time (2310_h)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
2310_h	0	*Number of entries*	1	1	unsigned 8	ro
	1	*Sample_Time*	$00C8_h$...$FFFF_h$	0271_h => 312.5 µs	unsigned 16	rw

Via this object the sample rate can be defined. Always all eight A/D-channels are converted.

Assignment of the Variable *Sample_Time*:

The sample time is subdivided in 0.5 µs steps:

Value of the variable *Sample_Time_x* [Hex]	Sample time	Sample rate
($00C8_h$)	(100 µs)	(10 000 SPS)
($00C9_h$)	(100,5 µs)	(9 950 SPS)
:	:	:
0271_h	312,5 µs (default)	3200 SPS
:	:	:
$FFFF_h$	32767,5 µs	30 SPS

The sample rate is calculated of the sample time as described below:

$$\text{Sample Rate} = \frac{1}{Sample\_Time}$$

The sample rate is specified in SPS (Samples Per Second).

Note:
The default sample time is 312,5 µs.
Under certain conditions the module might be able to operate error-free with shorter sample times. Furthermore this depends on further parameters as addition (*AccuN*) and averaging (*AverageN*).

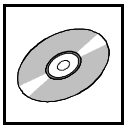
A minimum sample time might be possibly be reached, if the value *AccuN* is set to the highest possible value. In this case *AccuN* = '8'

Example: Settings for minimum sample time		
AccuN	*AverageN*	*Sample Time*
8	0	approx. 125 µs

At disadvantageous settings of *AccuN* and *AverageN* the minimum sample time will be longer. The value *AccuN* is therefore set to the smallest value.
In this case *AccuN* = '0'.
An *AverageN* unequal '0' will increase the sample time too.

Example: Disadvantageous settings for minimum sample time		
AccuN	*AverageN*	*Sample Time*
0	>0	ca. 450µs

Additionally, in practice it has to be taken into account for short sample times, that possibly not all determined analog data can be transmitted, due to the available data rate and the CAN bus load.



8.11.3 Sample Time Actual Value (2312_h)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
2312_h	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Sample_Time_Actual _1*	0...$FFFF_h$	-	unsigned 16	ro
	2	*Sample_Time_Actual _2*	0...$FFFF_h$	-	unsigned 16	ro
	3	*Sample_Time_Actual _3*	0...$FFFF_h$	-	unsigned 16	ro
	4	*Sample_Time_Actual _4*	0...$FFFF_h$	-	unsigned 16	ro
	5	*Sample_Time_Actual _5*	0...$FFFF_h$	-	unsigned 16	ro
	6	*Sample_Time_Actual _6*	0...$FFFF_h$	-	unsigned 16	ro
	7	*Sample_Time_Actual _7*	0...$FFFF_h$	-	unsigned 16	ro
	8	*Sample_Time_Actual _8*	0...$FFFF_h$	-	unsigned 16	ro

Via this object the actual value of the sample time preset can be read for each channel.

Assignment of the variable *Sample_Time_Actual_x* (x = 1...8):

The value of the variable read multiplied by 0.5 results in the sample time [µs] in the default setting of the objects:

Value of variables Sample_Time_Actual_x	Sample time
$00C8_h$	100 µs
$00C9_h$	100.5 µs
:	:
$FFFF_h$	32767.5 µs

Note:
For compatibility reasons the *Sample-Time_Actual_x* equates the *Sample Time* (Objekt: 2310_h).

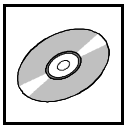
8.11.4 Channel Enabled (2401_h)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
2401_h	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Channel_Enabled_1*	false, true	-	boolean	ro
	2	*Channel_Enabled_2*	false, true	-	boolean	ro
	3	*Channel_Enabled_3*	false, true	-	boolean	ro
	4	*Channel_Enabled_4*	false, true	-	boolean	ro
	5	*Channel_Enabled_5*	false, true	-	boolean	ro
	6	*Channel_Enabled_6*	false, true	-	boolean	ro
	7	*Channel_Enabled_7*	false, true	-	boolean	ro
	8	*Channel_Enabled_8*	false, true	-	boolean	ro

The object *Channel Enabled* returns, which A/D-channels are mapped in a PDO.

Value range:

Channel_Enabled_x = false => A/D-converter channel is not ‘mapped’
Channel_Enabled_x = true => A/D-converter channel is ‘mapped’



8.11.5 Accu N (2402_h)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
2402_h	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Accu_Count_1*	0...8	3	unsigned 8	rw
	2	*Accu_Count_2*	0...8	3	unsigned 8	rw
	3	*Accu_Count_3*	0...8	3	unsigned 8	rw
	4	*Accu_Count_4*	0...8	3	unsigned 8	rw
	5	*Accu_Count_5*	0...8	3	unsigned 8	rw
	6	*Accu_Count_6*	0...8	3	unsigned 8	rw
	7	*Accu_Count_7*	0...8	3	unsigned 8	rw
	8	*Accu_Count_8*	0...8	3	unsigned 8	rw

This object defines, how many analog values are to be added up for the averaging.

By the appropriate pre-processing of the output value of the A/D-converter the result of this addition is always a 16-bit value in the two's complement representation.

The number of additions is calculated as:

$$\text{number of additions} = 2^{(Accu\_Count\_x)}$$

Up to 256 values can be added up (default setting: 8 values are added up).

Advantage: Filter with decimation, improvement of the resolution, limitation of the data rate.
The data rate is calculated as follows:

$$\text{Data rate} = \frac{Sample\_Rate}{Number\_of\_additions} = \frac{1/Sample\_Time}{2^{(Accu\_Count\_x)}}$$

In default (Accu_Count = 03, Sample_Time = 312,5 µs) the data rate is 400 SPS.

Disadvantage: The measured value is not updated until the addition is finished.

8.11.6 Average N (2403_h)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
2403_h	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Average_Count_1*	0...5	4	unsigned 8	rw
	2	*Average_Count_2*	0...5	4	unsigned 8	rw
	3	*Average_Count_3*	0...5	4	unsigned 8	rw
	4	*Average_Count_4*	0...5	4	unsigned 8	rw
	5	*Average_Count_5*	0...5	4	unsigned 8	rw
	6	*Average_Count_6*	0...5	4	unsigned 8	rw
	7	*Average_Count_7*	0...5	4	unsigned 8	rw
	8	*Average_Count_8*	0...5	4	unsigned 8	rw

This object defines, how many buffered analog values are used to calculate the moving average. During determination of the analog values the average of the last 2^n samples is evaluated.
After every conversion a new average is available, because the A/D-values are buffered in a ring buffer.
The number of the averaged values is calculated as:

$$\text{number of averaged values} = 2^{(Average\_Count\_x)}$$

It can be averaged from the last 1, 2, 4, 8, 16 (default) or 32 values.

Note:
For input signals with a frequency << F_{sample} the filter improves the resolution by

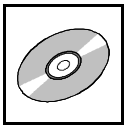
Average_Count /2 bits .

Furthermore a Notchfilter characteristic can be obtained with zero points at

$$F_{notch} = \frac{F_{Sample}}{2^{Average\_Count}} \cdot k \qquad \text{with } k = 1 \ldots 2^{Average\_Count}$$

Example:
At a sample-frequency of 200 Hz and *Average_Count* = 2 a suppression of the frequencies of 150 Hz, 100 Hz and 50 Hz can be additionally obtained.

Advantage: After every conversion a new average is available.
Disadvantage: Higher internal calculating effort as for the addition of the measured values (see object 2402_h)



8.11.7 Calibration Offset Value (2404_h)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
2404_h	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Calibration_Offset_1*	80000000_h ... $7FFFFFFF_h$	-	integer 32	rw
	2	*Calibration_Offset_2*	80000000_h ... $7FFFFFFF_h$	-	integer 32	rw
	3	*Calibration_Offset_3*	80000000_h ... $7FFFFFFF_h$	-	integer 32	rw
	4	*Calibration_Offset_4*	80000000_h ... $7FFFFFFF_h$	-	integer 32	rw
	5	*Calibration_Offset_5*	80000000_h ... $7FFFFFFF_h$	-	integer 32	rw
	6	*Calibration_Offset_6*	80000000_h ... $7FFFFFFF_h$	-	integer 32	rw
	7	*Calibration_Offset_7*	80000000_h ... $7FFFFFFF_h$	-	integer 32	rw
	8	*Calibration_Offset_8*	80000000_h ... $7FFFFFFF_h$	-	integer 32	rw

In this object an offset value for the correction of the A/D-value can be specified. The offset affects the objects 6401_h and 6402_h (see also figure on page 82).

The default values $Offset_{Factory}\_x$ (x = 1 ...8) are determined in the manufacturer's calibration of the CAN-CBX-AI814. With object 1011_h (*load_manufacurer_parameter*) these module-specific default values can be loaded again.

Value range:

Value of the variable *Calibration_Offset_x*								Offset voltage
Binary (bit 31...0)								Hexa-decimal	
31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0		
1 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	80000000_h	-10.24 V
: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	:	:
1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	$FFFFFFFF_h$	-4.7684 nV
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	00000000_h	0
0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 1	00000001_h	+4.7684 nV
: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	: : : :	:	:
0 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1	$7FFFFFFF_h$	+10.24 V -(1 $LSB_{VARIABLE}$)

Resolution of the offset value:
1 LSB based on the variable *Calibration_Offset_x* (x = 1...8):
1 $LSB_{VARIABLE}$ => 10.24 V / 8000 0000_h => 4.7684 nV

8.11.8 Calibration Gain Value (2405_h)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
2405_h	0	*Number of entries*	8	8	unsigned 8	ro
	1	*Calibration_Gain_1*	8000_h...$7FFF_h$	-	integer 16	rw
	2	*Calibration_Gain_2*	8000_h...$7FFF_h$	-	integer 16	rw
	3	*Calibration_Gain_3*	8000_h...$7FFF_h$	-	integer 16	rw
	4	*Calibration_Gain_4*	8000_h...$7FFF_h$	-	integer 16	rw
	5	*Calibration_Gain_5*	8000_h...$7FFF_h$	-	integer 16	rw
	6	*Calibration_Gain_6*	8000_h...$7FFF_h$	-	integer 16	rw
	7	*Calibration_Gain_7*	8000_h...$7FFF_h$	-	integer 16	rw
	8	*Calibration_Gain_8*	8000_h...$7FFF_h$	-	integer 16	rw

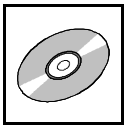
With this object the gain of the A/D-converter channels can be corrected. The gain value, with which the measured A/D-converter value is multiplied, is calculated as:

$$Gain\_x = 1 + \frac{Calibration\_Gain\_x}{2^{18}}$$

with x= 1 ... 8

The resulting value range for the gain factor is: 0.875 ... 1.125

The default values $Gain_{Factory}\_x$ (x = 1 ...8) are determined in the manufacturer's calibration of the CAN-CBX-AI814. With object 1011_h (*load_manufacurer_parameter*) these module-specific default values can be loaded again.



8.12 Firmware Update via DS-302-Objects $1F50_h$...$1F52_h$

The objects described below are used for program updates via the object dictionary.

Attention:
The firmware update must be carried out only by qualified personnel!
Faulty program update can result in deleting of the memory and loss of the firmware.
The module then can not be operated further!

Note:
esd offers the program CANfirmdown for a firmware update.
Please, contact our support for this.

In normal DS 301 mode the object $1F50_h$ can not be accessed.
The objects $1F51_h$ and $1F52_h$ are also available in normal DS 301-mode.
For further information about the objects and the firmware-update please refer to [5].

Index	Sub-index	Description	Data type	Access mode
$1F50_h$	0	Boot-Loader: Firmware download	domain	rw
$1F51_h$	1	Boot-Loader: FLASH command	unsigned 8	rw
$1F52_h$	0,1,2	Boot-Loader: Firmware date	unsigned 32	ro

8.12.1 Download Control via Object ($1F51_h$)

INDEX	**$1F51_h$**
Name	Program Control
Data type	unsigned 8
Access type	rw
Value range	$0...FE_h$
Default value	0

Note:
The value range of this objects in the implementing of the CAN-CBX-AI814 differs from the value range specified in [5].
For further information about object $1F51_h$ and the firmware-update please refer to [5]

8.12.2 Verify Application Software ($1F52_h$)

Index	Sub-index	Description	Value range	Default	Data type	Access mode
$1F52_h$	0	*Number of entries*	2	2	unsigned 8	ro
	1	*Application_Software_Date*	$0...FFFF\ FFFF_h$	-	unsigned 32	rw
	2	*Application_Software_Time*	$0...0526\ 5C00_h$	-	unsigned 32	rw

Description of the variable:

Application_Software_Date — Date of the generation of the firmware used, specified in number of days since 1. January 1984

Application_Software_Time — Time of the generation of the firmware used, specified in milliseconds since midnight.

9. References

[1] CiA 301
CANopen Application layer and communication profile V4.2.0 (12.2011)
CAN in Automation (CiA) e.V., Nürnberg, Germany

[2] CiA Draft Standard Proposal 401 V3.0 (10.2006)
CANopen Device profile for generic IO modules

[3] CiA Draft Recommendation 303 V1.3 (08.2006)
CANopen Additional specification, Part 3: Indicator specification

[4] CAN Application Layer for Industrial Applications CiA/DS202-2 February 1996
CMS Protocol Specification

[5] CiA Draft Standard Proposal 302 V4.1 (04.2010)
Additional Application Layer functions, Part 3: Configuration and program download

[6] Phoenix Contact GmbH & Co. KG, Blomberg.
Technical data is taken from the Phoenix Contact website:
https://www.phoenixcontact.com/online/portal/de;
PCB plug connector - FKCT-2,5/4-ST KMGY - 1921900, downloaded 2013-10-09

[5] Phoenix Contact GmbH & Co. KG, Blomberg.,
Technical data is taken from the Phoenix Contact website:
https://www.phoenixcontact.com/online/portal/de;
PCB plug connector - FK-MCP 1,5/ ...-STF-3,81 - 1851261, downloaded 2013-10-09

10. EU Declaration of Conformity

EU-KONFORMITÄTSERKLÄRUNG
EU DECLARATION OF CONFORMITY

Adresse / *Address*
esd electronic system design gmbh
Vahrenwalder Str. 207
30165 Hannover
Germany

esd erklärt, dass das Produkt *esd declares, that the product*	Typ, Modell, Artikel-Nr. *Type, Model, Article No.*
CAN-CBX-AI814	**C.3020.02**
die Anforderungen der Normen *fulfills the requirements of the standards*	**EN 61000-6-2:2005,** **EN 61000-6-4:2007+A1:2011**
gemäß folgendem Prüfbericht erfüllt. *according to test certificate.*	**H-K00-0357-09**
Das Produkt entspricht damit der EU-Richtlinie „EMV" *Therefore the product corresponds to the EU Directive 'EMC'*	**2014/30/EU**
Das Produkt entspricht der EU-Richtlinie „RoHS" *The product corresponds to the EU Directive 'RoHS'*	**2011/65/EU**

Diese Erklärung verliert ihre Gültigkeit, wenn das Produkt nicht den Herstellerunterlagen entsprechend eingesetzt und betrieben wird, oder das Produkt abweichend modifiziert wird.
This declaration loses its validity if the product is not used or run according to the manufacturer's documentation or if non-compliant modifications are made.

Name / *Name*	T. Ramm
Funktion / *Title*	CE-Koordinator / *CE Coordinator*
Datum / *Date*	Hannover, 2014-07-11

Rechtsgültige Unterschrift / *authorized signature*

I:\Texte\Doku\MANUALS\CAN\CBX\AI814\CE-Konformität\CAN-CBX-AI814_EG_Konformitaetserklaerung_2014-07-11.odt

11. Order Information

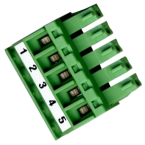
Type	Features	Order No.
CAN-CBX-AI814	CAN-CBX-AI814 8 analog inputs, 14 bit, including 1x CAN-CBX-TBUS (C.3000.01)	C.3020.02
Manuals		
CAN-CBX-AI814-ME	Manual in English	C.3020.21
Accessories		
CAN-CBX-TBUS 	Mounting-rail bus connector of the CBX-InRailBus for CAN-CBX-modules, (one bus connector is included in delivery of the CAN-CBX-module)	C.3000.01
CAN-CBX-TBUS-Connector 	Terminal plug of the CBX-InRailBus for the connection of the +24 V power supply voltage and the CAN interface Female type	C.3000.02
CAN-CBX-TBUS-Connection adapter 	Terminal plug of the CBX-InRailBus for the connection of the +24 V power supply voltage and the CAN interface Male type	C.3000.03

Table 13: Order information

PDF Manuals
Manuals are available in English and usually in German as well. For availability of English manuals see the following table.
Please download the manuals as PDF documents from our esd website www.esd.eu for free.

Order Information

Manuals		Order No.
CAN-CBX-AI814-ME	Manual in English	C.3010.21
CAN-CBX-AI814-MD	Manual in German	C.3010.20

Table 14: Available manuals

Printed Manuals
If you need a printout of the manual additionally, please contact our sales team: sales@esd.eu for a quotation. Printed manuals may be ordered for a fee.