CAN-CBX-DIO8

Compact I/O-Module
with InRailBus

Manual

to Product C.3010.02

esd electronic system design gmbh
Vahrenwalder Str. 207 • 30165 Hannover • Germany
www.esd-electronics.com • Fax: 0511/37 29 8-68
Phone: 0511/37 29 80 • International: +49-5 11-37 29 80

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

esd assumes no responsibility for the use of any circuitry other than circuitry which is part of a product of **esd** gmbh.

esd does not convey to the purchaser of the product described herein any license under the patent rights of **esd** gmbh nor the rights of others.

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-electronics.com
Internet: www.esd-electronics.com

USA / Canada:
esd electronics Inc.
525 Bernardston Road
Suite 1
Greenfield, MA 01301
USA

Phone: +1-800-732-8006
Fax: +1-800-732-8093
E-mail: us-sales@esd-electronics.com
Internet: www.esd-electronics.us

Document-File:	I:\Texte\Doku\MANUALS\CAN\CBX\DIO8\Englisch\CBX-DIO8_27.en9
Date of print:	2008-03-25

PCB versions:	Rev. 1.0
Firmware version:	2.0

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	Changes versus previous version
3.2.2	Description of the LED indication corrected
-	Current Declaration of CE Conformity inserted.

Technical details are subject to change without further notice.

This page is intentionally left blank.

This page is intentionally left blank.

1. Overview

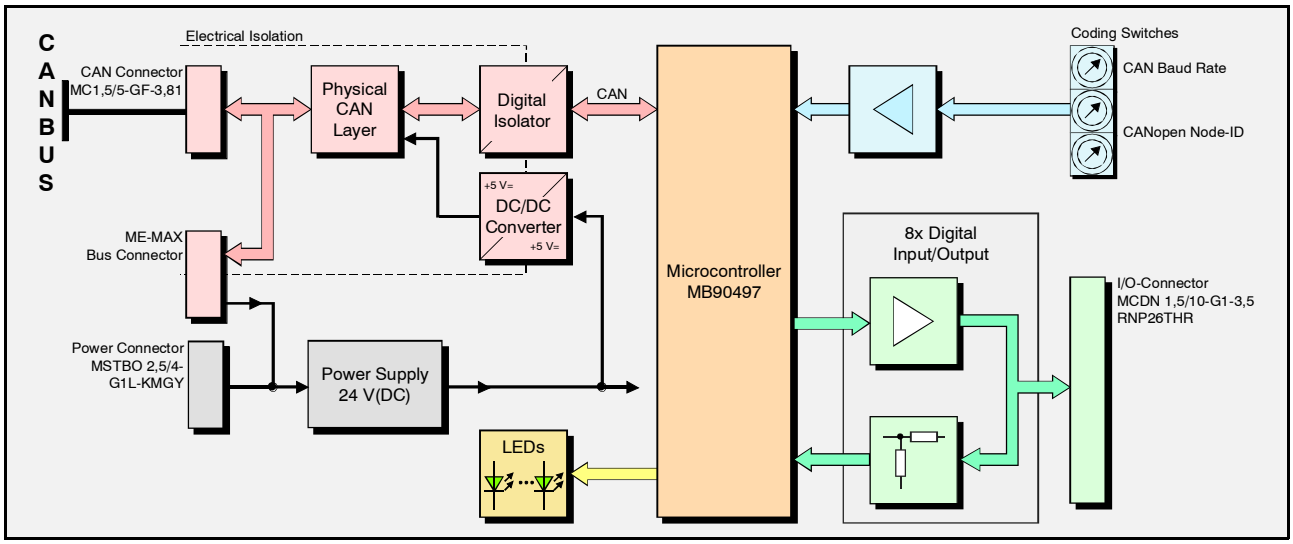


Figure 1: Block circuit diagram

The CAN-CBX-DIO8 module is equipped with eight digital in-/outputs which can be individually set as outputs or inputs.
The power supply voltage and the CAN bus signals can be fed via the InRailBus or they can be fed via plugs with spring-cage connection.

The CAN-CBX-DIO8 module operates with a MB90F497 microcontroller which buffers CAN data into a local SRAM. The firmware is contained in the flash. Parameters are stored in a serial EEPROM.

The ISO 11898-compliant CAN interface allows a maximum data transfer rate of 1 Mbit/s. The CAN interface is electrically isolated by a dual 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 input voltage range: 18...30 V/DC current consumption (24 V, 20 ˚C): typ.: 30 mA, (without VIO) max.: 40 mA
Connectors	X100 (4-pin COMBICON plug with spring-cage connection) - 24 V-power supply voltage X101 (5-pin ME-MAX-TBUS connector, Phoenix Contact) - CAN interface and power supply voltage via InRailBus X400 (20-pin Mini-COMBICON connector) - digital input/output X600 (5-pin Mini-COMBICON connector) - CAN interface only for test- and programming purposes: X200 (6-pin COMBICON connector) the connector is placed inside the case
Temperature range	-20 ˚C ... +85 ˚C ambient temperature
Humidity	max. 90%, non-condensing
Dimensions	22 x 112 x 113 mm (W, H, D) (including mounting rail fitting and connector projection)
Weight	approx. 125 g

Table 1: General data of the module

2.2 CPU Unit

CPU	16 bit µC MB90F497
RAM	2 Kbyte integrated
Flash	64 Kbyte integrated
EEPROM	128 byte

Table 2: Microcontroller

2.3 CAN Interface

Number of CAN interfaces	1 x CAN
Connection	5-pin Phoenix Contact Combicon with spring-cage connection or via Phoenix Contact TBUS connector (InRailBus)
CAN controller	MB90F497, CAN 2.0A/B, (only 11-bit CAN identifier are supported by software)
Electrical isolation of CAN interfaces from other units	via dual digital isolator (ADUM120BR) and DC/DC converters
Physical layer CAN	physical layer according to ISO 11898, transfer rate programmable from 10 Kbit/s up to 1 Mbit/s

Table 3: CAN interface data

2.4 Digital In-/Outputs

Number	8 I/O channels, in- or output individually selectable, over voltage protection (max. 30 V)
Digital output circuit	
Design	high side switch
Nominal voltage	24 V
Nominal load current	up to 1 A
Digital input circuit	
Nominal voltage	24 V
Protective circuit	20-pin Mini-Combicon

Table 4: Data of digital in/outputs

2.5 Order Information

Type	Features	Order No.
CAN-CBX-DIO8	CAN-CBX-DIO8 8 I/O-ports, 24 V input voltage, output current 1 A, 1x CAN-CBX-TBUS (C.3000.01)	C.3010.02
Manuals:		
CAN-CBX-DIO8-ME	Manual in English	C.3010.21
Accessories:		
CAN-CBX-TBUS	Mounting-rail bus connector of the CBX-InRailBus for CAN-CBX modules, (a 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 +24V power supply voltage and the CAN interface	C.3000.02

Table 5: Order information

3. Hardware Installation

3.1 Connecting Diagram

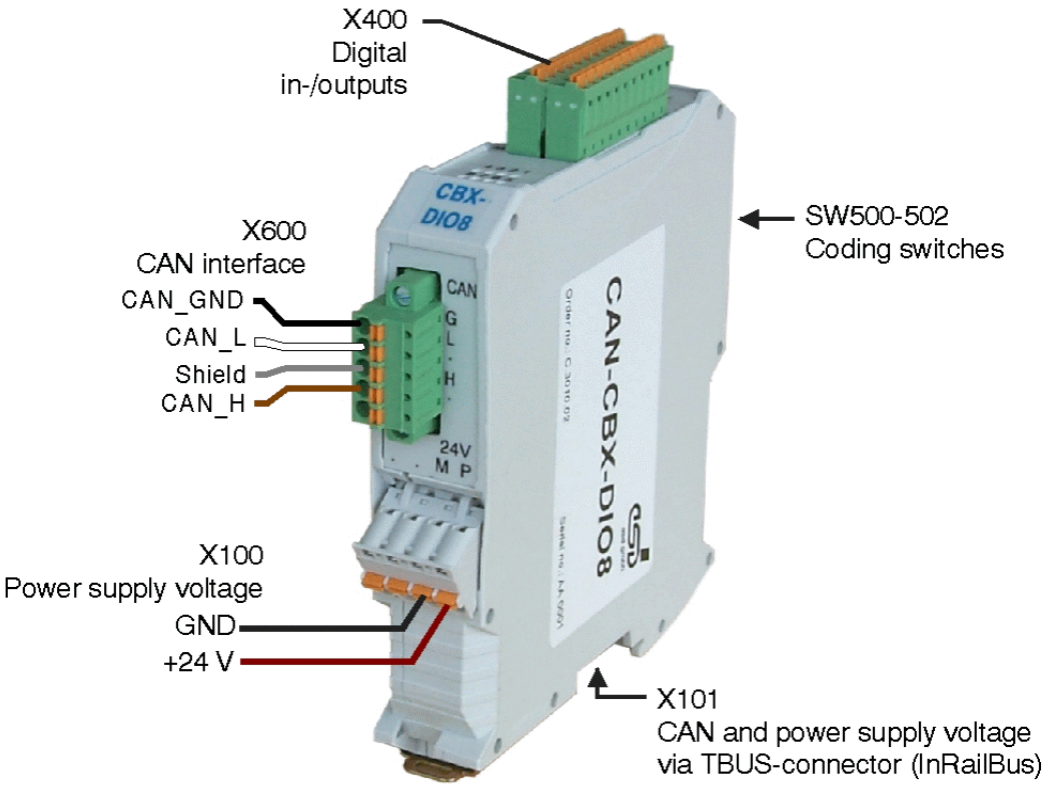


Figure 2: Connections of the CAN-CBX-DIO8 module

The connector pin assignments can be found on page 27 and following.

3.2 LED Display

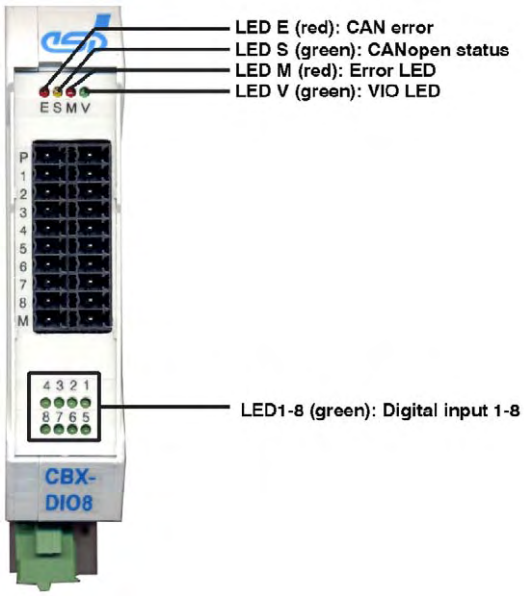


Figure 3: Position of the LEDs

The CAN-CBX-DIO8 module is equipped with 4 status LEDs and 8 LEDs for the I/O-channels. The terms of the indicator states of the LEDs are chosen in accordance with the CANopen Standard DS 303-3, V 1.2 (chapter 3.1). They are described in the following chapters.

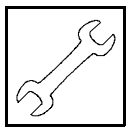
3.2.1 Indicator States

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 6: Indicator states

The red LED flashes opposite in phase with the green LED.



3.2.2 Operation of the CAN-Error LED

LED indication				Display function	
Label	Name	Colour	Circuit no.	Indicator state	Description
E	CAN Error	red	200A	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 page 17)
				2 flashes	Heartbeat or Nodeguard error occurred. The LED switches automatically to 'off', if Nodeguard/Heartbeat-messages are received again.

Table 7: Indicator states of the red CAN Error-LED

3.2.3 Operation of the CANopen-Status LED

LED indication				Display function	
Label	Name	Colour	Circuit no.	Indicator state	Description
S	CANopen Status	green	200B	blinking	*Pre-operational*
				on	*Operational*
				1 flash	*Stopped*
				3 flashes	Module is in bootloader mode (or coding switch position ID-node > $7F_h$ when switching on; see page 17)

Table 8: Indicator states of the CANopen Status-LED

3.2.4 Operation of the Error-LED

LED indication				Display function	
Label	Name	Colour	Circuit no.	Indicator state	Description
M	Error	red	200C	off	no error
				on	error occurred on one of the monitored outputs - if the module has switched to *Stopped* state when the error occurred, the LED remains 'on', even if the error does no longer exist - errors that occur after switching to *Stopped* state, will not be indicated
				2 flashes	EEPROM error - stored data have an invalid checksum therefore default values are loaded - indicator state lasts until the module resets or an error occurs at the outputs

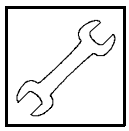
Table 9: Indicator state of the Error-LED

3.2.5 Operation of the VIO-LED

The display functions described in the following table are only valid if the module is in the *Operational* state and Output Port and Output Port Config are set.
If these conditions do not apply, the LED is on.

LED indication				Display function if the module is in *Operational* state and Output Port and Output Port Config are set	
Label	Name	Colour	Circuit no.	Indicator state	Description
V	VIO_Sense	green	200D	off	read value of the operating voltage of the digital outputs (object 2300_h ,sub-index 1, see page 96) is lower than 5 V
				blinking	read value of the operating voltage of the digital outputs is lower than the value set in object 2300_h sub-index 2 (see page 96)
				flickering	read value of the operating voltage of the digital outputs is higher than the value set in object 2300_h sub-index 3 (see page 96)
				on	read value of the operating voltage of the digital outputs is within the limits (object 2300_h , sub-index 2 and 3, see page 96)

Table 10: Indicator state of the VIO-LED



3.2.6 Special Indicator States

The following status 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 11: Special indicator states

3.2.7 I/O-LED 1-8

LED	Name	Indication function = I/O channel state
1	LED400D	IO1
2	LED400C	IO2
3	LED400B	IO3
4	LED400A	IO4
5	LED410D	IO5
6	LED410C	IO6
7	LED410B	IO7
8	LED410A	IO8

Table 12: Indication of LEDs 1-8

LED	State of channels IO1...IO8
off	input voltage level is below the lower switching threshold, or output state is ‘off’
on	input voltage level is higher than the upper switching threshold, or output state is ‘on’

Table 13: State of channels IO1...IO8

3.3 Coding Switch

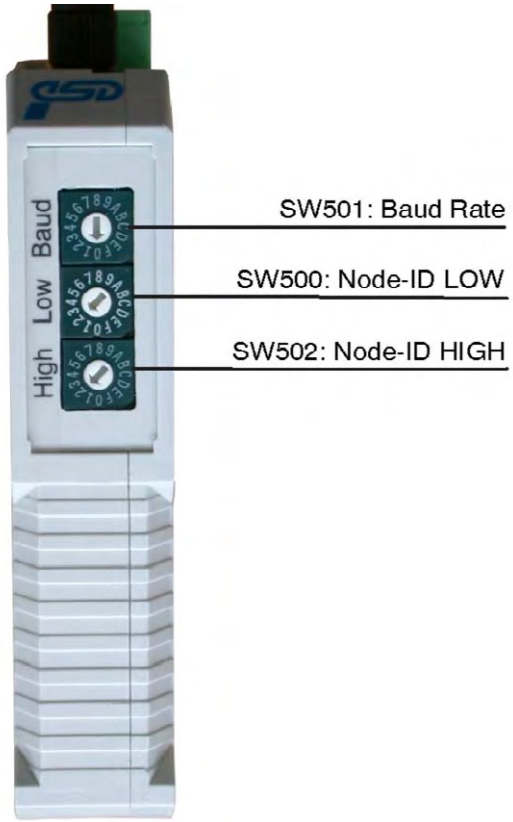


Figure 4: Position of the coding switches

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-DIO8 modules can be set *hexadecimal* from 01_h to $7F_h$ (*decimal* from 1 to 127).

The four 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.

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]	Baud rate [Kbit/s]
0	1000
1	666.6
2	500
3	333.3
4	250
5	166
6	125
7	100
8	66.6
9	50
A	33.3
B	20
C	12.5
D	10
E	reserved
F	reserved

Table 14: Index of the baud rate

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:

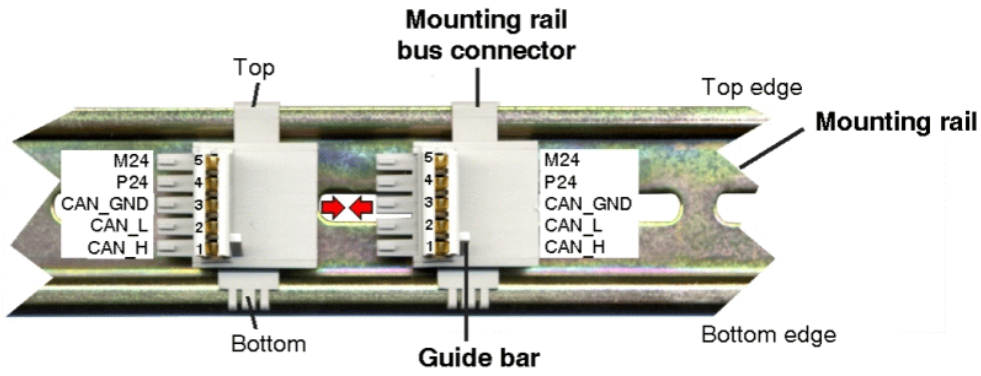


Figure 5: Mounting rail with bus connector

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.

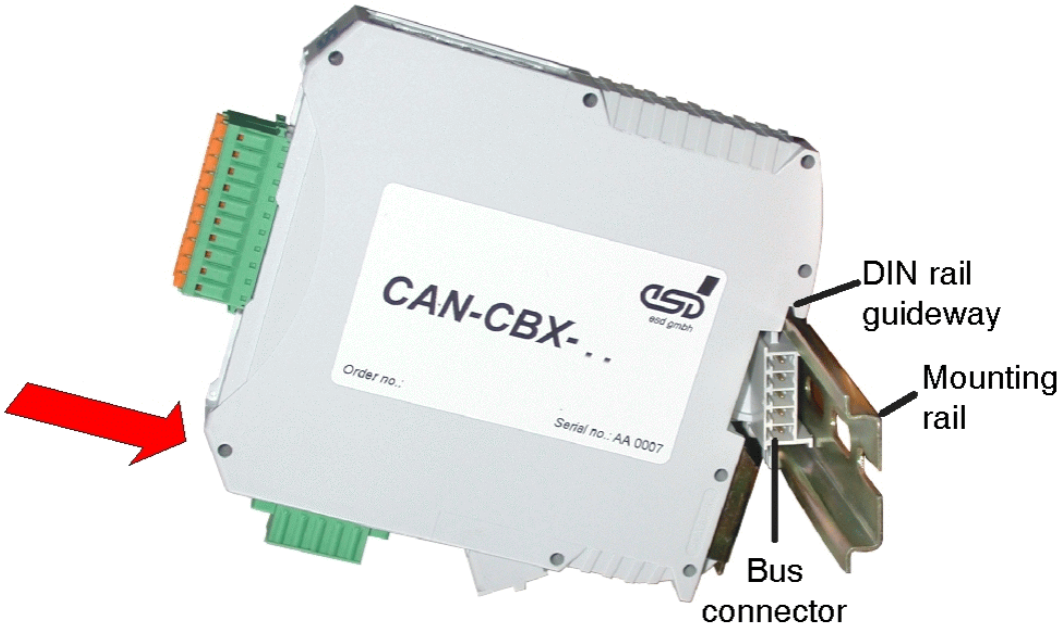


Figure 6 : Mounting CAN-CBX modules

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.

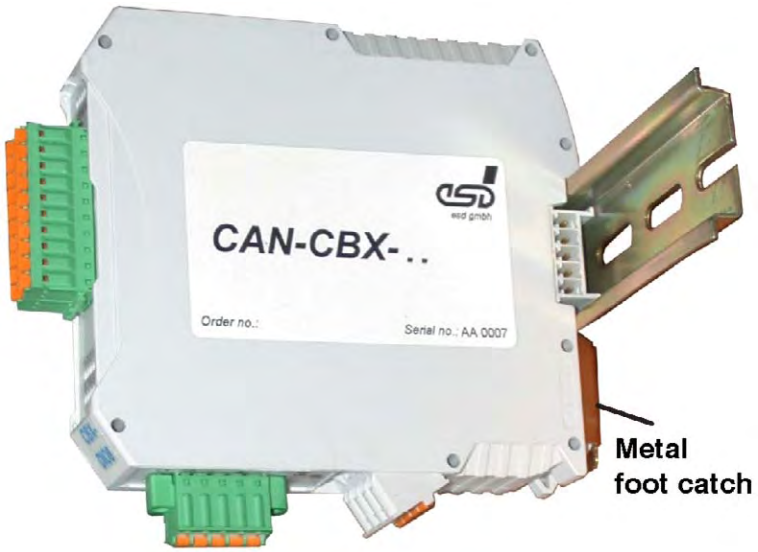


Figure 7: Mounted CAN-CBX module

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 (order no.: C.3000.02) is needed. The terminal plug is not included in delivery and must be ordered separately (see order information).

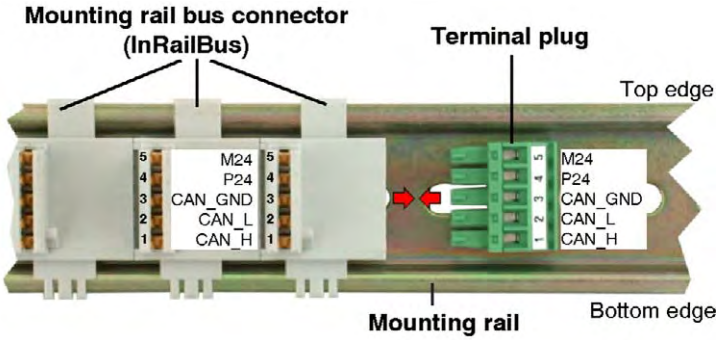


Fig. 8: Mounting rail with InRailBus and terminal plug

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

Attention!
It is **not permissible** to feed through the power supply voltage through the CBX station and to supply it to another CBX station via connector X100! A feed through of the +24 V power supply voltage can cause damage on the CBX modules.

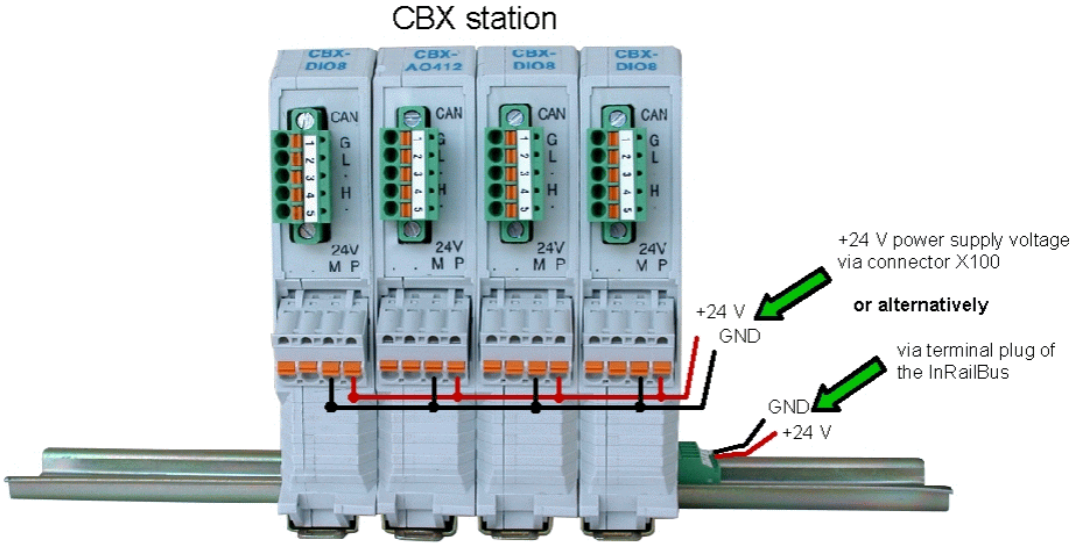


Fig. 9: Connecting the power supply voltage to the CAN-CBX station

3.4.3 Connection of CAN

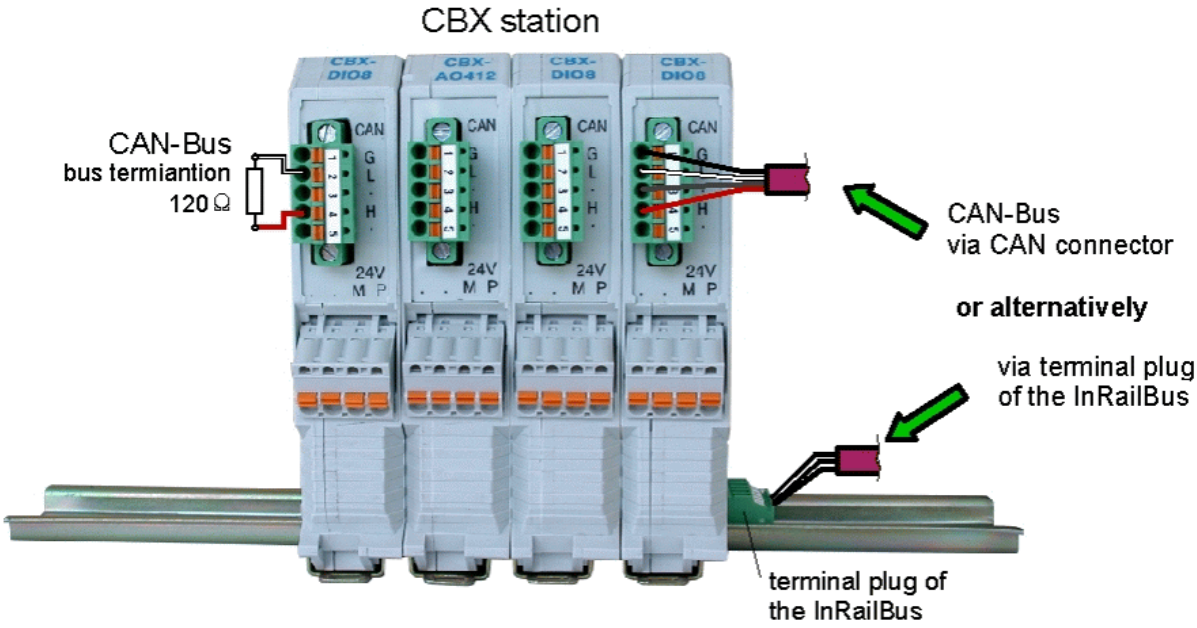


Fig. 10: Connecting the CAN signals to the CAN-CBX station

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 lead through the CAN signals 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 entire individual devices from the whole without interrupting the InRailBus connection, because the contact chain will not be interrupted.

4. Description of the Units

4.1 CAN Interface

An 82C251 is used as driver unit. The differential CAN bus signals are electrically isolated from the other signals via a dual digital converter (ADUM120BR) and a DC/DC converter.

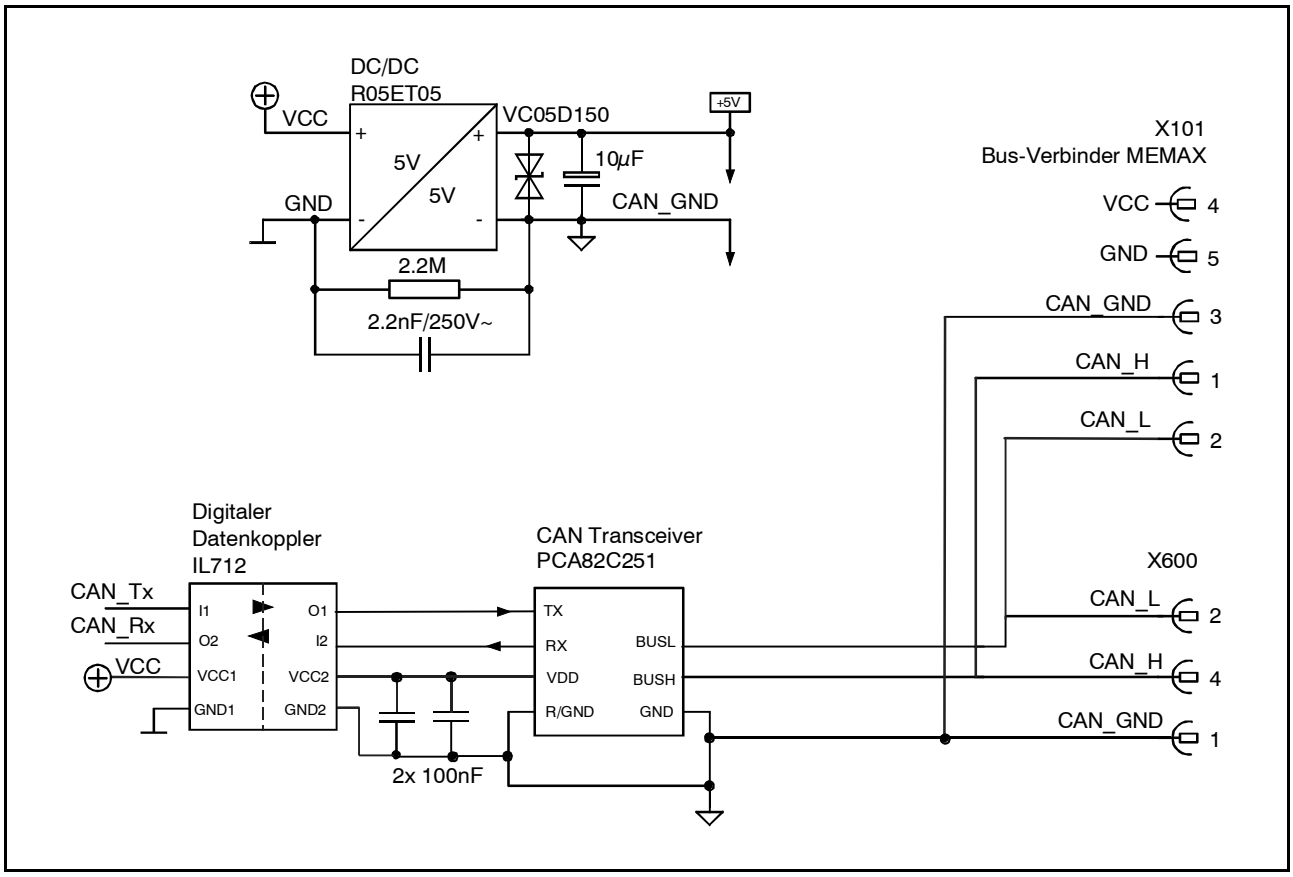


Figure 11: CAN interface

4.2 Digital Inputs

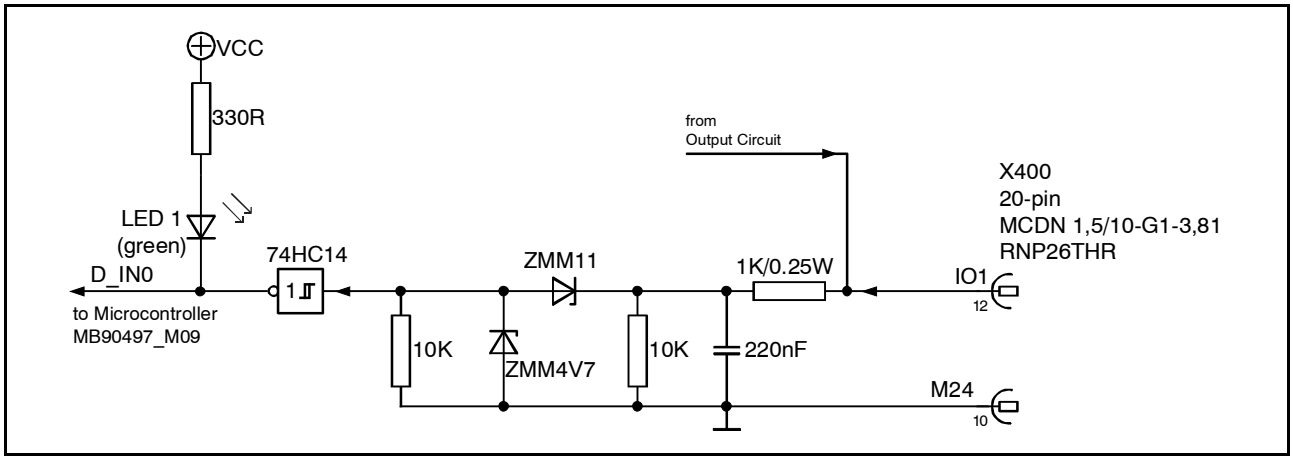


Figure 12: Digital input circuit (Example: IO1)

4.3 Digital Outputs

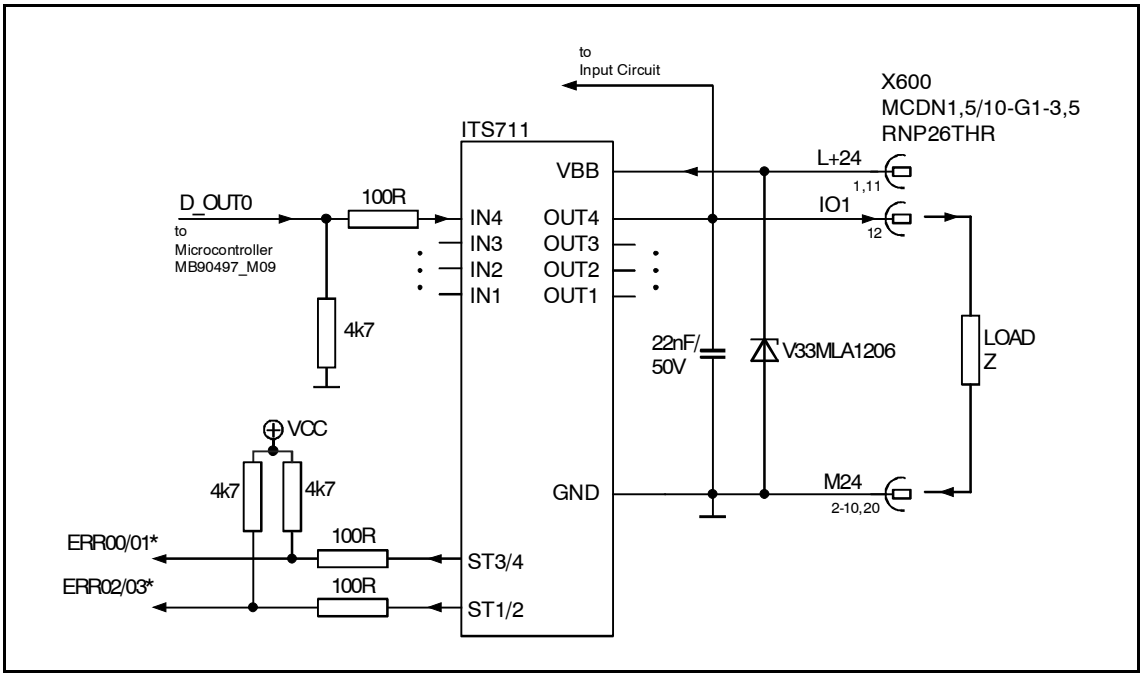


Figure 13: Digital output circuit (Example: IO1)

5. Connector Pin Assignment

5.1 Power Supply Voltage X100

Device connector: COMBICON MSTBO 2,5/4-G1L-KMGY
Line connector: COMBICON 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)

Pin Position:

Pin Assignment:

Pin	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 to each other at the PCB.
The pins 2 and 3 are connected to each other at the PCB.

Signal Description:

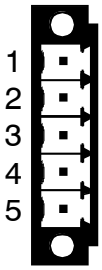
P24... power supply voltage +24 V
M24... reference potential

5.2 CAN-Bus X600

Device Connector: COMBICON MC 1,5/5-GF-3,81
Line Connector: COMBICON FK-MCP 1,5/5-STF-3,81, spring-cage connection (included in the scope of delivery)

Pin Position:

(Illustration of device connector)



Pin-Assignment:

Pin	Signal
1	CAN_GND
2	CAN_L
3	Shield
4	CAN_H
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:

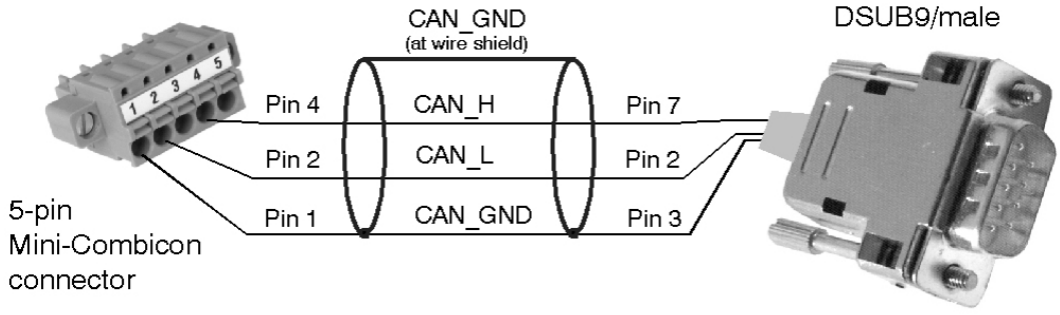
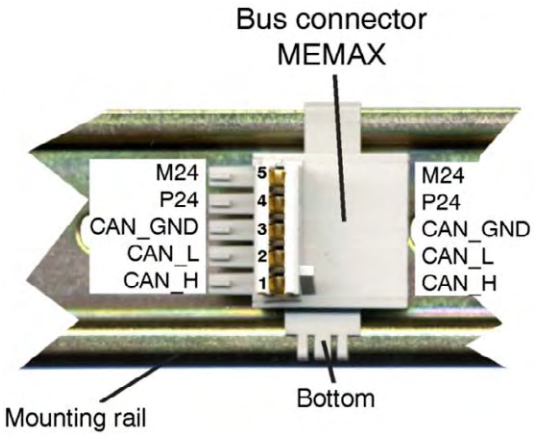


Figure 14: Assignment of the 9-pin DSUB-connector according to CiA DS 102.

5.3 CAN and Power Supply Voltage via InRailBus Connector X101

Connector type: Bus connector MEMAX
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)

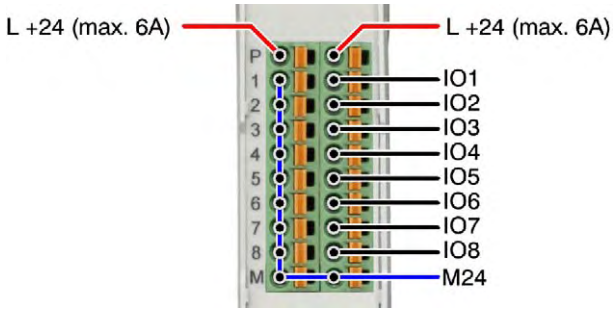
Signal Description:

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)

5.4 Digital In/Outputs X400

Device connector: Phoenix Contact Combicon MCDN 1,5/10-G1-3,5 RNP26THR
Line connector: Phoenix Contact Combicon 2x FMC 1,5/10-ST-3,5 (spring-cage connections)
(contained in the scope of supply)

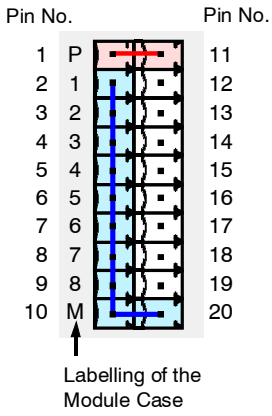
Connector top view:



Pin Assignment:

Signal	Pin
L + 24 V	1
	2
	3
	4
	5
M24	6
	7
	8
	9
	10

Pin Position:



Pin Assignment:

Pin	Signal
11	L + 24 V
12	IO1
13	IO2
14	IO3
15	IO4
16	IO5
17	IO6
18	IO7
19	IO8
20	M24

Signal Description:

L +24 V... supply voltage of the digital outputs
M24... reference potential
IO1-8... signals of the digital IOs 1-8

Attention!
The maximum current load of the connector pins is 6A/pin. If all 8 outputs are to be operated with the maximum admissible load, the supply voltage has to be connected to pins 1 and 11. These pins are connected to each other on the PCB.

6. Correctly Wiring Electrically Isolated CAN Networks

Generally all instructions applying for wiring regarding an electromagnetic compatible installation, wiring, cross sections of wires, material to be used, minimum distances, lightning protection, etc. have to be followed.

The following **general rules** for the CAN wiring must be followed:

1.	A CAN net must not branch (exception: short dead-end feeders) and has to be terminated by the wave impedance of the wire (generally 120 W ±10%) at both ends (between the signals CAN_L and CAN_H and **not** at GND)!
2.	A CAN data wire requires **two twisted** wires and a wire to conduct the reference potential (CAN_GND)! For this the shield of the wire should be used!
3.	The reference potential CAN_GND has to be connected to the earth potential (PE) at **one** point. Exactly **one** connection to earth has to be established!
4.	The bit rate has to be adapted to the wire length.
5.	Dead-end feeders have to kept as short as possible (l < 0.3 m)!
6.	When using double shielded wires the external shield has to be connected to the earth potential (PE) at **one** point. There must be not more than **one** connection to earth.
7.	A suitable type of wire (wave impedance ca. 120 Ω ±10%) has to be used and the voltage loss in the wire has to be considered!
8.	CAN wires should not be laid directly next to disturbing sources. If this cannot be avoided, double shielded wires are preferable.

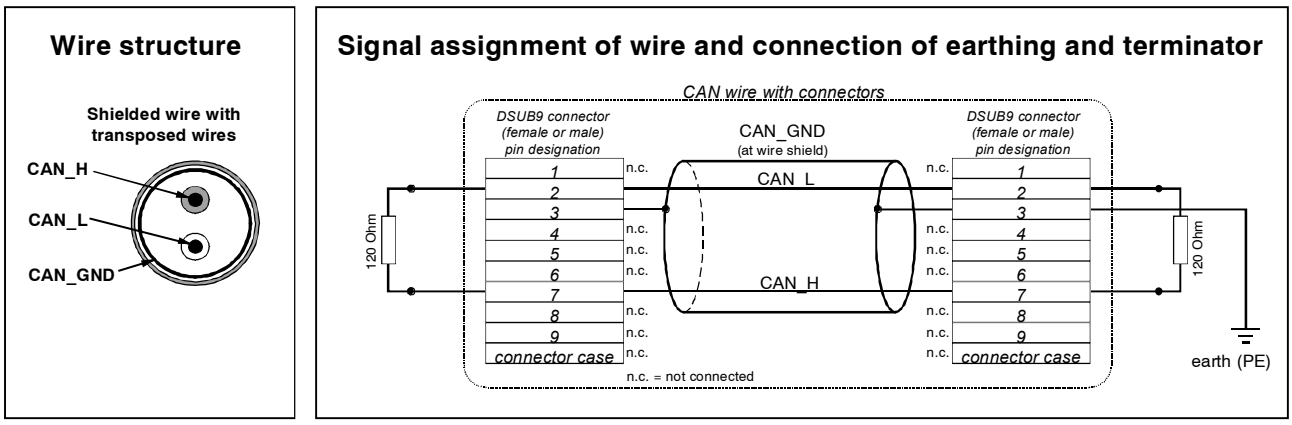


Figure: Structure and connection of wire

Cabling

- for devices which have only one CAN connector per net use T-connector and dead-end feeder (shorter than 0.3 m) (available as accessory)

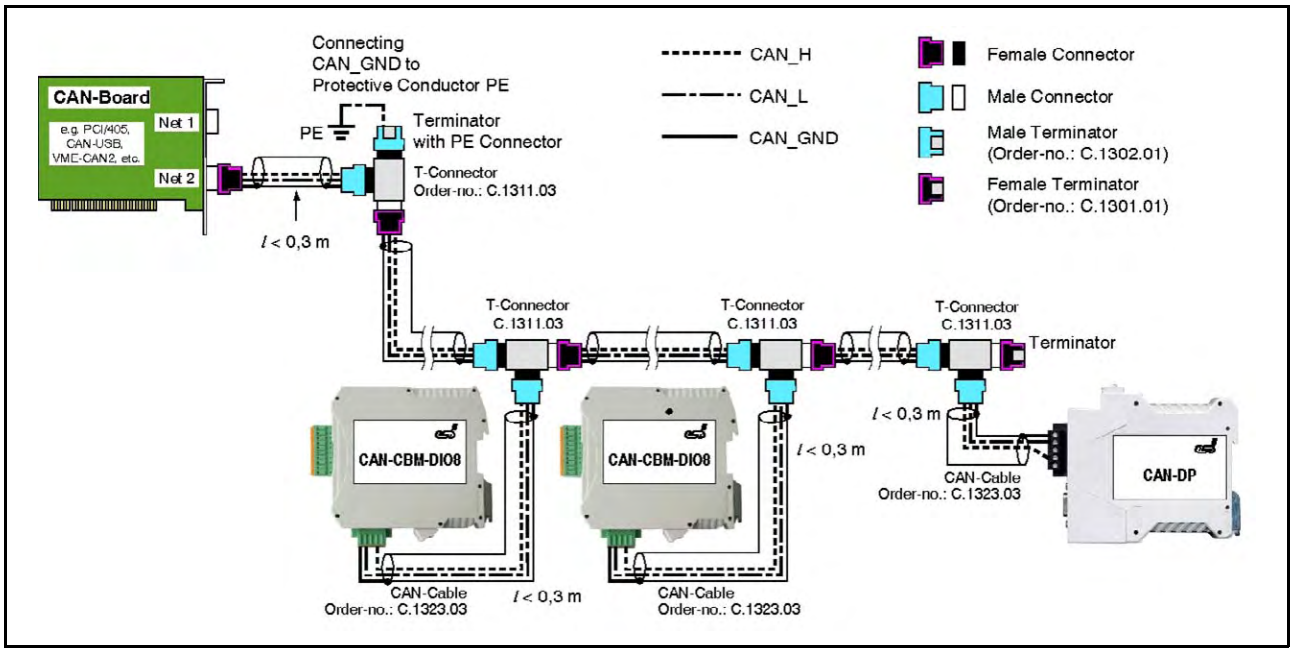


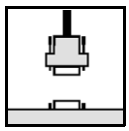
Figure: Example for correct wiring (when using single shielded wires)

Terminal Resistance

- use **external** terminator, because this can later be found again more easily!
- 9-pin DSUB-terminator with male and female contacts and earth terminal are available as accessories

Earthing

- CAN_GND has to be conducted in the CAN wire, because the individual esd modules are electrically isolated from each other!
- CAN_GND has to be connected to the earth potential (PE) at **exactly one** point in the net!
- each CAN user without electrically isolated interface works as an earthing, therefore: do not connect more than one user without potential separation!
- Earthing CAN e.g. be made at a connector



Wire Length

❍ Optical couplers are delaying the CAN signals. By using fast optical couplers 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 dead-end feeders.

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
666.6	80	-
500	130	100
333.3	180	-
250	270	250
166	420	-
125	570	500
100	710	650
66.6	1000	-
50	1400	1000
33.3	2000	-
20	3600	2500
12.5	5400	-
10	7300	5000

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

Examples for CAN Wires

Manufacturer	Type of wire
U.I. LAPP GmbH Schulze-Delitzsch-Straße 25 70565 Stuttgart Germany www.lappkabel.de	e.g. UNITRONIC ®-BUS CAN UL/CSA (UL/CSA approved) UNITRONIC ®-BUS-FD P CAN UL/CSA (UL/CSA approved)
ConCab GmbH Äußerer Eichwald 74535 Mainhardt Germany www.concab.de	e.g. BUS-PVC-C (1 x 2 x 0.22 mm²) Order No.: 93 022 016 (UL appr.) BUS-Schleppflex-PUR-C (1 x 2 x 0.25 mm²) Order No.: 94 025 016 (UL appr.)
SAB Bröckskes GmbH&Co. KG Grefrather Straße 204-212b 41749 Viersen Germany www.sab-brockskes.de	e.g. SABIX® CB 620 (1 x 2 x 0.25 mm²) Order No.: 56202251 CB 627 (1 x 2 x 0.25 mm²) Order No.: 06272251 (UL appr.)

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

7. 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.

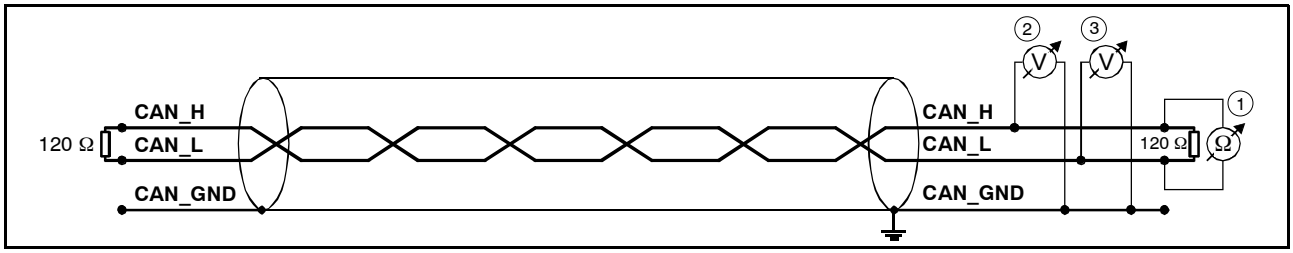


Figure: Simplified diagram of a CAN network

7.1 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 middle and ends of the network (1) (see figure above).

The measured value should be between 50 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
- 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.

7.2 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).

7.3 Ground

The shield of the CAN network has to be grounded at only one location. This test will indicate if the shielding is grounded in several places. To test it, please

1. Disconnect the shield wire (Shield) from the ground.
2. Measure the DC resistance between Shield and ground (see picture on the right hand).
3. Connect Shield wire to ground.

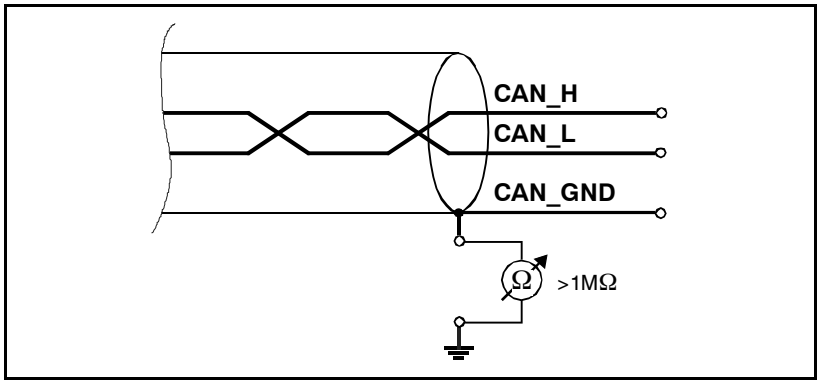


Fig.: Simplified schematic diagram of ground test measurement

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

7.4 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. Disconnect the node from the network. Leave the node unpowered (4) (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.

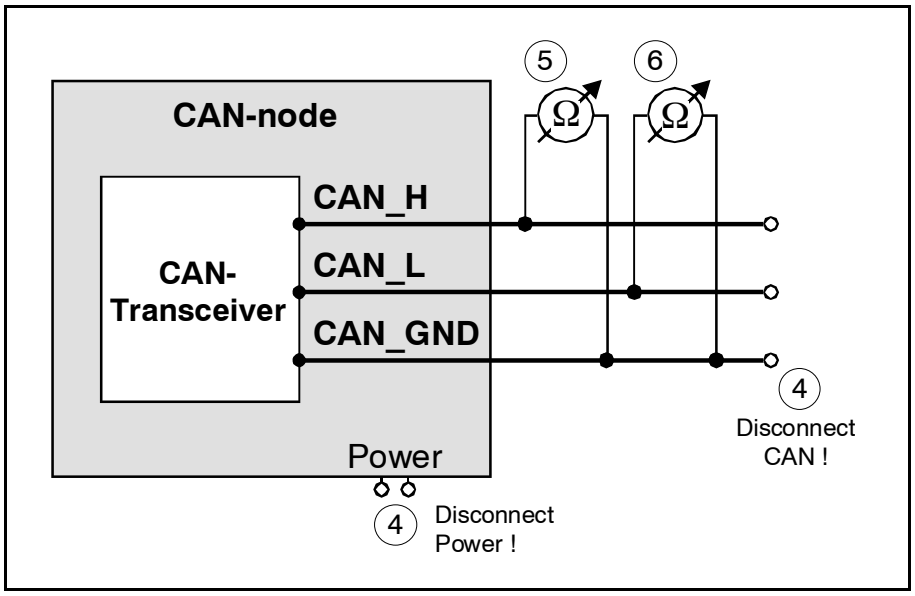


Figure: Simplified diagram of a CAN node

8. Software

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 CAL / CANopen documentation 'CiA Draft Standard 301, V 4.02' and 'CiA Draft Standard Proposal 401, V 2.1'.

8.1 Definition of Terms

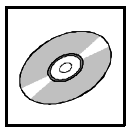
COB ...	Communication Object
Emergency-Id...	Emergency Data Object
NMT...	Network Management (Master)
Rx...	receive
SDO...	Service Data Object
Sync...	Sync(frame) Telegram
Tx...	transmit

PDOs (Process Data Objects)

PDOs are used to transmit process data.
In the 'Receive'-PDO (RxPDO) process data is received by the CAN-CBX-DIO8 module.
In the 'Transmit'-PDO (TxPDO) the CAN-CBX-DIO8 module transmits data to 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-DIO8 module can be initialized with the 'Minimum Capability Device' boot-up as described in CiA-Draft Standard 301 in chapter 9.4.

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 on CAN-identifier '0000_h' (= Start Remote Node all Devices).

8.3 CANopen Object Directory

The object directory is basically a (sorted) group of objects which can be accessed via the 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 (CiA-Draft DS 301) or a manufacturer-specific code. By means of the MSBs of the index the object class of the parameter is defined.

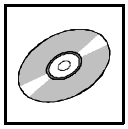
Part of the object directory are among others:

Index [Hex]	Object	Example
0001 ... 009F	definition of data types	-
1000 ... 1FFF	Communication Profile Area	1001_h : error register
2000 ... 5FFF	Manufacturer Specific Profile Area	2200_h : Reset_Output_State
6000 ... 9FFF	Standardized Device Profile Area	according to application profile DS 40x
A000 ... FFFF	reserved	-

8.3.1 Access on the Object Directory via SDOs

SDOs (Service Data Objects) are used to get access to the object directory of a device.
An SDO therefore represents a 'channel' to access the parameter of the device. The access via this channel can be made in CAN-CBX-DIO8 module state *operational* and *pre-operational*.

The SDOs are transmitted on ID '**600_h + NodeID**' (request). The receiver acknowledges the parameter on ID '**580_h + NodeID**' (response).



Software

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	00	14_h	01	$7F_h$	04_h	00	00
	(write)	(Index=1400_h) (Receive-PDO-Comm-Para)		(COB-def.)	COB = $047F_h$			

Description of the SDOs:

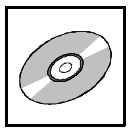
Identifier . . . . . The parameters are transmitted on ID '600_h + NodeID' (request).
The receiver acknowledges the parameters on ID '580_h + NodeID' (response).

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-DIO8 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-DIO8 module reports a communication error



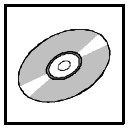
Frequently Used Command Codes

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 CiA DS 301, chapter “Service Data Object”.

Command	Number of data bytes	Command code [Hex]
Write Request (Initiate Domain Download)	1 2 3 4	2F 2B 27 23
Write Response (Initiate Domain Download)	-	60
Read Request (Initiate Domain Upload)	-	40
Read Response (Initiate Domain Upload)	1 2 3 4	4F 4B 47 43
Error Response (Abort Domain Transfer)	-	80

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’.

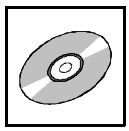


Software

Error Codes of the SDO Domain Transfer

The following error codes might occur (according to CiA DS 301, chapter “Abort SDO Transfer Protocol”):

Abort code [Hex]	Description
0x05040001	wrong command specifier
0x06010002	wrong write access
0x06020000	wrong index
0x06040041	object can not be mapped to PDO
0x06060000	access failed due to an hardware error
0x06070010	wrong number of data bytes
0x06070012	service parameter too long
0x06070013	service parameter too small
0x06090011	wrong sub-index
0x06090030	transmitted parameter is outside the accepted value range
0x08000000	undefined cause of error
0x08000020	Data cannot be transferred or stored to the application
0x08000022	Data cannot be transferred or stored to the application because of the present device state
0x08000024	access to flash failed



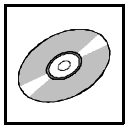
8.4 Overview of Used CANopen Identifiers

Function	Identifier [Hex]	Description
Network management	0	NMT
SYNC	80	Sync to all, (configurable via object 1005_h)
Emergency Message	80 + *Node-ID*	configurable via object 1014_h
Tx-PDO	180 + *Node-ID*	PDO from CAN-CBX-DIO8 (Tx) (object 1800_h)
Rx-PDO	200 + *Node-ID*	PDO to CAN-CBX-DIO8 (Rx) (object 1400_h)
Tx-SDO	580 + *Node-ID*	SDO from CAN-CBX-DIO8 (Tx)
Rx-SDO	600 + *Node-ID*	SDO to CAN-CBX-DIO8 (Rx)
Node Guarding	700 + *Node-ID*	configurable via object $100E_h$

Node-ID: CANopen address [1_h...$7F_h$]

8.4.1 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 19). 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)



Software

8.5 PDO Assignment

PDOs (Process Data Objects) are used to transmit process data:

CAN identifier	Length	Transmission direction	Assignment
200_h + Node-ID	1 byte	to CAN-CBX-DIO8 (Rx/Receive-PDO)	Setting of the outputs
180_h + Node-ID	1 byte	from CAN-CBX-DIO8 (Tx/Transmit-PDO)	Request the state of the inputs

Rx-PDO1 (-> CAN-CBX-DIO8)

CAN identifier: 200_h + Node-ID

Byte	0	1	2	3	4	5	6	7
Parameter	*Write_ Output_ DO8-DO1*	-	-	-	-	-	-	-

Parameter description:

Name	Description	Data type	See page
Write_ Output_ DO8-DO1	Setting of the digital outputs	Byte	87

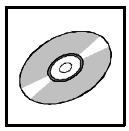
Tx-PDO1 (CAN-CBX-DIO8->)

CAN identifier: 180_h + Node-ID

Byte	0	1	2	3	4	5	6	7
Parameter	*Read_ Input_ DI8-DI1*	-	-	-	-	-	-	-

Parameter description:

Name	Description	Data type	See page
Read_ Input_ DI8-DI1	state of the digital inputs	Byte	8080



8.6 Setting and Reading of the Out/Inputs

8.6.1 Status Message of the Digital Inputs

The following transmission types are available at the CAN-CBX-DIO8:

- *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).

8.6.2 Digital Outputs

The digital outputs are set, as soon as an output setting object is received by the CAN-CBX-DIO8 (e.g. object 6200_h via Rx-PDO).

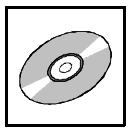
8.6.3 Supported Transmission Types Based on DS 301

Transmission Type	PDO-Transmission					supported by CAN-CBX-DIO8
	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

8.7 Implemented CANopen-Objects

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

Index [Hex]	**Sub-index** (max.) [Dec]	**Description**	**Data type**	**R/W**	**Default value**
1000	-	Device Type	unsigned 32	ro	00030191_h
1001	-	Error Register	unsigned 8	ro	00_h
1003	10	Pre-Defined-Error-Field	unsigned 32	rw	00_h
1005	-	COB-ID-Sync	unsigned 32	rw	80_h
1008	-	Manufacturer Device Name	visible string	ro	"CAN-CBX-DIO8"
1009	-	Manufacturer Hardware Version	visible string	ro	x.yy (depending on version)
100A	-	Manufacturer Software Version	visible string	ro	x.yy (depending on version)
100C	-	Guard Time	unsigned 16	rw	0000_h
100D	-	Life Time Factor	unsigned 8	rw	00_h
100E	-	Node Guarding Identifier	unsigned 32	rw	Node-ID + 700_h
1010	3	Store Parameter	unsigned 32	rw	n.a.
1011	3	Restore Parameter	unsigned 32	rw	n.a.
1014	-	COB-ID Emergency Object	unsigned 32	rw	80_h + Node-ID
1015	-	Inhibit Time EMCY	unsigned 16	rw	00_h
1016	1	Consumer Heartbeat Time	unsigned 32	rw	00_h
1017	-	Producer Heartbeat Time	unsigned 16	rw	00_h
1018	4	Identity Object	unsigned 32	ro	Vendor Id: 00000017_h Prod. Code: 13010002_h
1020	2	Verify Configuration	unsigned 32	ro	n.a.
1029	3	Error Behaviour	unsigned 8	ro	00_h



Index [Hex]	**Sub-index**	**Description**	**Data type**	**R/W**
1400	2	1. Receive PDO Parameter	PDO CommPar (20_h)	rw
1600	1	1. Receive PDO Mapping	PDO Mapping (21_h)	ro
1800	5	1. Transmit PDO Parameter	PDO CommPar (20_h)	rw
1801	5	2. Transmit PDO Parameter	PDO CommPar (20_h)	rw
1A00	1	1. Transmit PDO Mapping	PDO Mapping (21_h)	ro
1A01	1	2. Transmit PDO Mapping	PDO Mapping (21_h)	ro

8.7.1 Device Type (1000_h)

INDEX	**1000_h**
Name	*device type*
Data type	unsigned 32
Access type	ro
Default value	0003 0191

The value of the *device type* is: 0003.0191_h (Digital input output: 0003_h
Digital profile number: 0191_h)

Example: Reading the Device Type

The CANopen master transmits the read request on identifier '603_h' (600_h + Node-ID) to the CAN-CBX-DIO8 module with the module no. 3 (Node-ID=3_h):

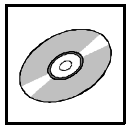
ID	RTR	LEN	DATA							
			1	2	3	4	5	6	7	8
603_h	0_h	8_h	40_h	00_h	10_h	00_h	00_h	00_h	00_h	00_h
			Read Request	Index=1000_h		Sub-index				

The CAN-CBX-DIO8 module no. 3 responds to the master by means of read response on 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	00_h	10_h	00_h	91_h	01_h	03_h	00_h
			Read Response	Index=1000_h		Sub-index	dig. Profile No.191		Digital Output	

value of device type: 0003.0191_h

The data field is always structured following the rule 'LSB first, MSB last' (see page 41, data field).



8.7.2 Error Register (1001_h)

This object is an error register for the CAN-CBX-DIO8.

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	-
4	*communication error* (overrun, error state)
5	-
6	-
7	-

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

The following messages are possible:

00_h no errors
01_h generic error
02_h current error
04_h voltage error
10_h communication error

8.7.3 Pre-defined Error Field (1003_h)

INDEX	**1003_h**
Name	*pre-defined error field*
Data type	unsigned 32
Access type	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.

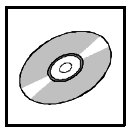
This module supports a maximum of 10 error entries. When the 11th error occurs the oldest error entry is deleted. In order to delete the entire error list, sub-index '0' has to be set to '0'. This is the only permissible write access to the object.

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

Index [Hex]	Sub-index [Dec]	Description	Value range [Hex]	Default	Data type	R/W
1003	0	*no_of_errors_in_list*	0, 1...10	-	unsigned 8	rw
	1	*error-code n*	0...FFFF FFFF	-	unsigned 32	ro
	2	*error-code (n-1)*	0...FFFF FFFF	-	unsigned 32	ro
	:	*:*	:	:	:	:
	10	*error-code (n-9)*	0...FFFF FFFF	-	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 DS 301, Table 21 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: for CAN-CBX-DIO8 always '00', unless *emergency-error-code* = 2300_h (see below)

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

- 8120h - CAN in Error Passive Mode
- 8130_h - Lifeguard Error / Heartbeat Error
- 8140_h - Recovered from "Bus Off"
- 8210_h - PDO too short
- 8220_h - PDO too long
- 6000_h - Software error:
  EEPROM Checksum error (no transmission of this error as Emergency Message)
- 3110_h - VIO too high
  (configurable via object 2300_h)
- 3120_h - VIO too low
  (configurable via object 2300_h)
- 2300_h - Output Error: open load
  (no load detected at least at one output)
- 2320_h - Output Error: short circuit at outputs
  (at least one output shorted to GND)

Emergency Frame

The data of the emergency frame transmitted by the CAN-CBX-DIO8 have the following structure:

Byte:	0	1	2	3	4	5	6	7
Contents:	*emergency-error-code* (see above)		*error-register* 1001_h	*no_of_errors_in_list* $1003{,}01_h$	-			

An Emergency Message is transmitted, if the first error of an output driver occurs and the according bit in object 2220_h is set to '1'. If another error occurs at another output no additional Emergency Message is generated.
If the last error message disappears, again an Emergency Message is transmitted to indicate the error disappearance.

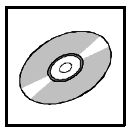
8.7.4 COB-ID SYNC Message (1005_h)

INDEX	**1005_h**
Name	*COB-ID SYNC message*
Data type	unsigned 32
Access type	rw
Default value	80_h

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 $0...7FF_h$.



8.7.5 Manufacturer's Device Name (1008_h)

INDEX	**1008_h**
Name	*manufacturer's device name*
Data type	visible string
Default value	string: 'CAN-CBX-DIO8'

For detailed description of the Domain Uploads, please refer to CiA DS 202-2 (CMS-Protocol Specification).

8.7.6 Manufacturer's Hardware Version (1009_h)

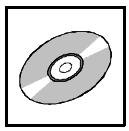
INDEX	**1009_h**
Name	*manufacturer's hardware version*
Data type	visible string
Default value	string: e.g. '1.0'

The hardware version is read similarly to reading the manufacturer device name via the domain upload protocol. Please refer to CiA DS 202-2 (CMS-Protocol Specification) for a detailed description of the upload.

8.7.7 Manufacturer's Software Version $100A_h$

INDEX	**$100A_h$**
Name	*manufacturer's software version*
Data type	visible string
Default value	string: e.g.: '1.2'

Reading the software version is similar to reading the manufacturer device name via the domain upload protocol. Please refer to CiA DS 202-2 (CMS-Protocol Specification) for a detailed description of the upload.



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

The CAN-CBX-DIO8 module supports the node guarding or alternatively the heartbeat function (see page 62)
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 type	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 type	rw
Default value	0
Minimum value	0
Maximum value	FF_h

8.7.9 Node Guarding Identifier ($100E_h$)

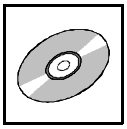
The module only supports 11-bit identifier.

INDEX	**$100E_h$**
Name	*node guarding identifier*
Data type	unsigned 32
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 $1...7FF_h$.



8.7.10 Store Parameters (1010_h)

This object supports saving of parameters to the EEPROM.

In order to avoid storage of parameters by mistake, storage is only executed when a specific signature is written to the appropriate sub-index. The signature is 'save'.
Reading the index returns information about the implemented storage functionalities (refer to CiA DS 301 for more information).

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

Index [Hex]	Sub-index	Description	Value range	Data type	R/W
1010	0	*number_of_entries*	4_h	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

Parameters which can be saved or loaded:

Communication Parameter of the objects 1005_h ... 1029_h

Application Parameter of the objects 6000_h ... 6100_h

Manufacturer Specific Parameter of the objects 2220_h ... 2310_h

8.7.11 Restore Default Parameters (1011_h)

Via this command the default parameters, valid when leaving the manufacturer, are activated again. Every individual setting stored in the EEPROM will be lost. Only command 'Restore all Parameters' is being supported.

In order to avoid restoring of default parameters by mistake, restoring is only executed when a specific signature is written to the appropriate sub-index. The signature is 'load'.
Reading the index provides information about its parameter restoring capability (refer to CiA DS 301 for more information).

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

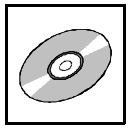
Index [Hex]	Sub-index	Description	Value range	Data type	R/W
1011	0	*number_of_entries*	3	unsigned 8	ro
	1	*load_all_default_parameters* (objects 1000_h ... $9FFF_h$)	no default, write: 64 61 6F $6C_h$ (= ASCII: 'd' 'a' 'o' 'l')	unsigned 32	rw
	2	*load_communication_parameter* (objects 1000_h ... $1FFF_h$)		unsigned 32	rw
	3	*load_application_parameter* (objects 6000_h ... $9FFF_h$)		unsigned 32	rw
	4	*load_manufacturer_parameter* (objects 2000_h ... $5FFF_h$)		unsigned 32	rw

Parameters which can be saved or loaded:

Communication Parameter of the objects 1005_h ... 1029_h

Application Parameter of the objects 6000_h ... 6100_h

Manufacturer Specific Parameter of the objects 2220_h ... 2310_h



8.7.12 COB_ID Emergency Object (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:

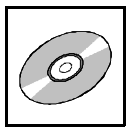
Bit-No.	Value	Meaning
31 (MSB)	0/1	0: EMCY exists / is valid 1: EMCY does not exist / EMCY is not valid
30	0	reserved (always 0)
29	0	always 0 (11-bit ID)
28...11	0	always 0 (29-bit IDs are not supported)
10...0 (LSB)	x	bits 0...10 of COB-ID

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

8.7.13 Inhibit Time EMCY (1015_h)

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

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



8.7.14 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 consumer 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-DIO8 module.

Each module can act as a heartbeat producer and a heartbeat consumer. One CAN-network can contain several heartbeat producers and heartbeat consumers.

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	Index [Hex]
1016	0	*number_of_entries*	1	1	unsigned 8	ro
	1	*consumer-heartbeat_time*	0...007FFFFF	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 Cycle time 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.

8.7.15 Producer Heartbeat Time (1017_h)

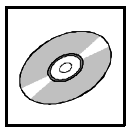
INDEX	**1017_h**
Name	*producer heartbeat time*
Data type	unsigned 16
Default value	0 ms

The Producer Heartbeat time defines the cycle time with which the CAN-CBX-DIO8 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 55).
If the value of the producer-heartbeat-time is set to '0', transmitting heartbeats by this module is stopped.

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
1017	0	*producer-heartbeat_time*	0...FFFF	0 ms	unsigned 16	rw

producer-heartbeat_time Cycle time 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.



8.7.16 Identity Object (1018_h)

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

this object contains general information about the CAN module.

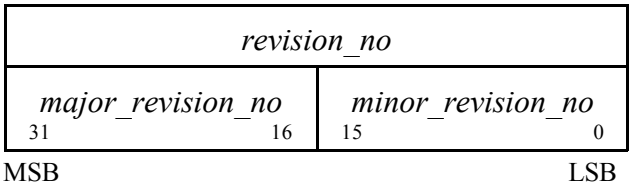
Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	Index [Hex]
1018	0	*no_of_entries*	0...4	4	unsigned 8	ro
	1	*vendor_id*	0...FFFF FFFF	0000 0017_h	unsigned 32	ro
	2	*product_code*	0...FFFF FFFF	2301 0002_h	unsigned32	ro
	3	*revision_number*	0...FFFF FFFF	0	unsigned32	ro
	4	*serial_number*	0...FFFF FFFF	-	unsigned32	ro

Parameter Description:

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.
Example:
Value '2301 0002_h' corresponds to article number 'C.3010.02'.

revision_number Here the software version is stored. In accordance with DS 301 the two MSB represent the revision numbers of the major changes and the two LSB show the revision number of minor corrections or changes.

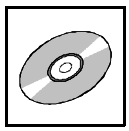


serial_number Here the serial number of the hardware is read. The first two characters of the serial number are letters which designate the manufacturing lot. The following characters represent the actual serial number.

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.



8.7.17 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 whether the configuration complies with the expected configuration or not in the future.

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
1020	0	*no_of_entries*	2	2	unsigned 8	ro
	1	*configuration_date*	0...FFFFFFFF	0	unsigned 32	rw
	2	*configuration_time*	0...FFFFFFFF	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.

8.7.18 Error Behaviour Object (1029_h)

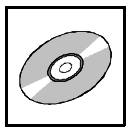
INDEX	**1029_h**
Name	*error behaviour object*
Data type	unsigned 8
Default value	No

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

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
1029	0	*no_of_error_classes*	6	6	unsigned 8	ro
	1	*communication_error*	0...2	0	unsigned 8	rw
	2	*output_error*	0...2	0	unsigned 8	rw
	3	*input_error*	0...2	0	unsigned 8	ro
	4	*VIO_extreme_low*	0...2	1	unsigned 8	rw
	5	*VIO_low_warning*	0...2	1	unsigned 8	rw
	6	*VIO_high_warning*	0...2	1	unsigned 8	rw

Parameter Description:

Parameter	Description
no_of_error_classes	number of error classes (here always '3')
communication_error	heartbeat/lifeguard error and *Bus off*
output_error	output error
input_error	input error (not supported at the moment)
VIO_extreme_low	output driver power supply error: VIO < 5 V
VIO_low_warning	output driver power supply error: VIO is lower than the value defined in object 2300_h, sub-index 2.
VIO_high_warning	output driver power supply error: VIO is higher than the value defined in object 2300_h, sub-index 3.



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

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

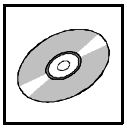
8.7.19 Receive PDO Communication Parameter 1400_h

Object 'Receive PDO Communication Parameter 1400_h' defines the parameters of a receive PDO (Rx-PDO).

INDEX	**1400_h**
Name	*receive PDO parameter*
Data type	PDOCommPar

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
1400	0	*no_of_entries*	2	2_h	unsigned 8	ro
	1	*COB_ID used by PDO1*	1... 8000 07FF	200_h + Node-ID	unsigned 32	rw
	2	*transmission type*	0...FF	255_d	unsigned 8	rw

All *transmission types* are supported.



8.7.20 Receive PDO Mapping Parameter 1600_h

Object 'Receive PDO Mapping Parameter 1600_h' defines the assignment of receive data to Rx-PDOs.

INDEX	**1600_h**
Name	*receive PDO mapping*
Data type	PDO Mapping

The following table shows the assignment of the Receive PDO Mapping parameters for the default configuration:

Index [Hex]	Subindex	Description	Value range [Hex]	Default	Data type	R/W
1600	0	*no_of_mapped_application_objects_in_PDO*	1...8	1	unsigned 8	rw
	1	*1st_application_object*	00050008, 6200 0108	6200 0108_h	unsigned 32	rw
	2	*2nd_application_object*		0005 0008_h	unsigned 32	rw
	3	*3rd_application_object*		0005 0008_h	unsigned 32	rw
	4	*4th_application_object*		0005 0008_h	unsigned 32	rw
	5	*5th_application_object*		0005 0008_h	unsigned 32	rw
	6	*6th_application_object*		0005 0008_h	unsigned 32	rw
	7	*7th_application_object*		0005 0008_h	unsigned 32	rw
	8	*8th_application_object*		0005 0008_h	unsigned 32	rw

Parallel connection of up to 8 CAN-CBX-DIO8-Modules:

In general there is the possibility to access several CAN-CBX-DIO8 modules synchronously. The CAN-CBX-DIO8 modules can be configured so that up to 8 CAN-CBX-DIO8 modules can be addressed with a single CAN frame simultaneously. Therefore all 64 digital outputs can be set simultaneously. The Receive COB-IDs of the CAN-CBX-DIO8 modules must be set to the same value using object 1400_h. Refer to DS-301 object 1600_h also.
Subindex 0_h contains the number of valid entries within the mapping record (see following table). The number of valid entries shall be the same for all modules.

Value	Description
00_h	Mapping disabled
01_h	Subindex 01_h valid
02_h	Subindex 01_h and 02_h valid
03_h	Subindex from 01_h- 03_h valid
:	:
08_h	Subindex from 01_h - 08_h valid

Subindices from 01_h to 08_h contain the information of the mapped application objects. The entry describes the content of the PDO by their index, subindex and length. For the CAN-CBX-DIO8 module the value 6200 0108_h (index 6200_h, subindex 01_h and length 08_h) may only be contained once. The other subindices contain the value 0005 0008_h as placeholder for the so called dummy mapping.

Example :

There are three CAN-CBX-DIO8 modules which shall be addressed via RPDO-Mapping simultaneously. Therefore the RPDO COB-IDs of the modules have to be configured to the same value. For further information refer to standard DS-301.

The object 1600_h must be configured differently for the three modules. For the first module the *application_object* has to be contained in subindex 1_h, for the second module in subindex 2_h and for the third module in subindex 3_h (see Figure15, 16, 17, 18). For a higher number of modules (here up to 8 modules) the entries have to be continued respectively.

In the preceding or following subindices (here subindex 1_h and 2_h) which are not used, the value 0005 0008_h has to be entered for the dummy-mapping. Subindex 0_h contains the number of subindices, according to the number of the valid objects.

Entry in object 1600_h for module 3:

Index [Hex]	Subindex	Description	Value
1600	0	*no_of_mapped_application_ objects_in_PDO*	03_h
	1	*1st_application_object*	0005 0008_h
	2	*2nd_application_object*	0005 0008_h
	3	*3rd_application_object*	**6200 0108_h**

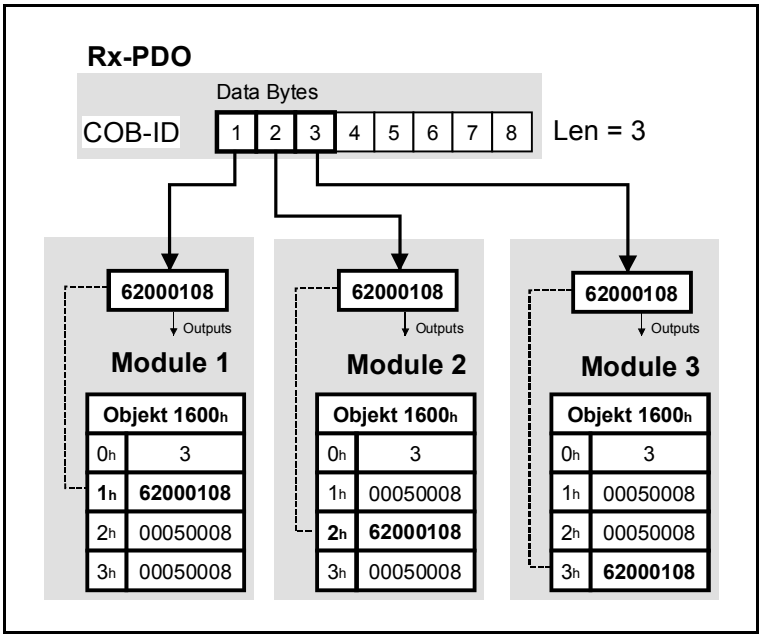


Figure 15: Example for the Rx-PDO mapping with three CAN-CBX-DIO8 modules

8.7.21 Object Transmit PDO1 Communication Parameter 1800_h

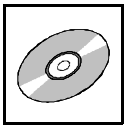
This object defines the parameters of a transmit PDO1.

INDEX	**1800_h**
Name	*transmit PDO1 parameter*
Data type	PDOCommPar

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
1800	0	*number_of_entries*	0...FF	5	unsigned 8	ro
	1	*COB-ID used by PDO1*	1...800007FF	180_h +Node-ID	unsigned 32	rw
	2	*transmission type*	0...FF	255_d	unsigned 8	rw
	3	*inhibit time*	0...FFFF	0	unsigned 16	rw
	4	*reserved*	0..FF	0	unsigned 8	const
	5	*event timer*	0...FFFF	0	unsigned 16	rw

The *transmission types* 0, 1...240, 252, 253 and 255 are supported.

Per default the first bit of the parameter *COB-ID used by_PDO1* is not set (**0**000 0000 +180_h +Node-ID).
The object is valid per default.



8.7.22 Object Transmit PDO2 Communication Parameter 1801_h

This object defines the parameters of a transmit PDO2.

INDEX	**1801_h**
Name	*transmit PDO2 parameter*
Data type	PDOCommPar

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
1801	0	*number_of_entries*	0...FF	5	unsigned 8	ro
	1	*COB-ID used by PDO2*	1...800007FF	8000 0000 +280_h +Node-ID	unsigned 32	rw
	2	*transmission type*	0...FF	255_d	unsigned 8	rw
	3	*inhibit time*	0...FFFF	0	unsigned 16	rw
	4	*reserved*	0..FF	0	unsigned 8	const
	5	*event timer*	0...FFFF	0	unsigned 16	rw

The *transmission types* 0, 1...240, 252, 253 and 255 are supported.

Per default the first bit of the parameter *COB-ID used by_PDO2* is set (**8**000 0000 +280_h +Node-ID). The object is not valid per default.

8.7.23 Transmit PDO1 Mapping Parameter $1A00_h$

The object 'Transmit PDO1 Mapping Parameter $1A00_h$' defines the assignment of transmit data to Tx-PDO1s.

INDEX	**$1A00_h$**
Name	*transmit PDO1 mapping*
Data type	PDO Mapping

The following table shows the assignment of Transmit PDO1 Mapping parameters:

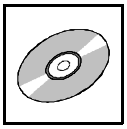
Index [Hex]	Sub-index	Description	Value range [Hex]	Default value	Data type	R/W
1A00	0	*number_of_entries*	0...FF	1_h	unsigned 8	ro
	1	*object_to_be_mapped*	0...FFFF FFFF	60000108_h	unsigned 32	rw

The Tx-PDO mapping is freely configurable.

Objects, that can be mapped (with index, sub-index and length in bit):

6000 01 08
2402 01 10 ... 2402 08 10
2403 01 20 ... 2403 08 20
0005 00 08

Note: In the state of delivery the PDO1-Mapping is valid per default.
The Transmit PDO1-Mapping parameters can not be modified, if the PDO1 is valid.
To change the value of the parameter *object_to_be_mapped* the value of the parameter COB-*ID_used_by_PDO1* in object 1800_h (page 72) has to be set to *not valid*.



8.7.24 Transmit PDO2 Mapping Parameter $1A01_h$

The object 'Transmit PDO2 Mapping Parameter $1A01_h$' defines the assignment of transmit data to Tx-PDO2s.

INDEX	**$1A01_h$**
Name	*transmit PDO2 mapping*
Data type	PDO Mapping

The following table shows the assignment of Transmit PDO2 Mapping parameters:

Index [Hex]	Sub-index	Description	Value range [Hex]	Default value	Data type	R/W
1A01	0	*number of entries*	0...FF	4_h	unsigned 8	rw
	1	*object_to_be_mapped_1*	0...FFFFFFFF	2402 01 10_h	unsigned 32	rw
	2	*object_to_be_mapped_2*	0...FFFFFFFF	2402 02 10_h	unsigned 32	rw
	3	*object_to_be_mapped_3*	0...FFFFFFFF	2402 03 10_h	unsigned 32	rw
	4	*object_to_be_mapped_4*	0...FFFFFFFF	2402 04 10_h	unsigned 32	rw

The Tx-PDO mapping is freely configurable.
The number of the objects, that can be mapped depends on their length. Maximum 8 byte can be transferred. From this follows that maximal four 16-bit values (see object 2402_h) or two 32-bit values (see object 2403_h) can be transferred in the PDO. Combinations are possible.

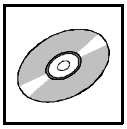
Objects, that can be mapped (with index, sub-index and length in bit):

6000 01 08
2402 01 10 ... 2402 08 10
2403 01 20 ... 2403 08 20
0005 00 08

Note: In the state of delivery the PDO2-Mapping is not valid per default.
If the PDO2 is not valid the Transmit PDO2-Mapping parameters can be modified.
To activate the PDO2-Mapping the value of the parameter COB-*ID_used_by_PDO2* in object 1801_h (page 73) has to be set to *valid*.

The following table shows an example of an assignment of the Transmit PDO2 mapping parameters:

Index [Hex]	Sub-index	Description	Value	Datatype
1A01	0	*number of entries in PDO2*	4_h	unsigned 8
	1	*object_to_be_mapped_1*	0005 00 08_h	unsigned 8
	2	*object_to_be_mapped_2*	6000 01 08_h	unsigned 8
	3	*object_to_be_mapped_3*	2403 05 20_h	unsigned 32
	4	*object_to_be_mapped_4*	2402 01 10_h	unsigned 16



8.8 Device Profile Area

8.8.1 Implemented Objects 6000_h-6207_h

Index [Hex]	Name	Data type
6000	*Read Input 8-bit*	unsigned8
6002	*Set Groups of 8 Input Polarities*	unsigned8
6003	*Filter Constant Input 8-bit*	unsigned8
6005	*Global Interrupt Enable Digital*	boolean
6006	*Interrupt Mask Any Change 8-bit*	unsigned8
6007	*Interrupt Mask Low-to-High 8-bit*	unsigned8
6008	*Interrupt Mask High-to-Low 8-bit*	unsigned8
6200	*Write Output 8-bit*	unsigned8
6202	*Change Polarity Output*	unsigned8
6206	*Error Mode Output 8-bit*	unsigned8
6207	*Error Value Output 8-bit*	unsigned8

8.8.2 Interrelation of the Implemented Objects of the Digital Inputs

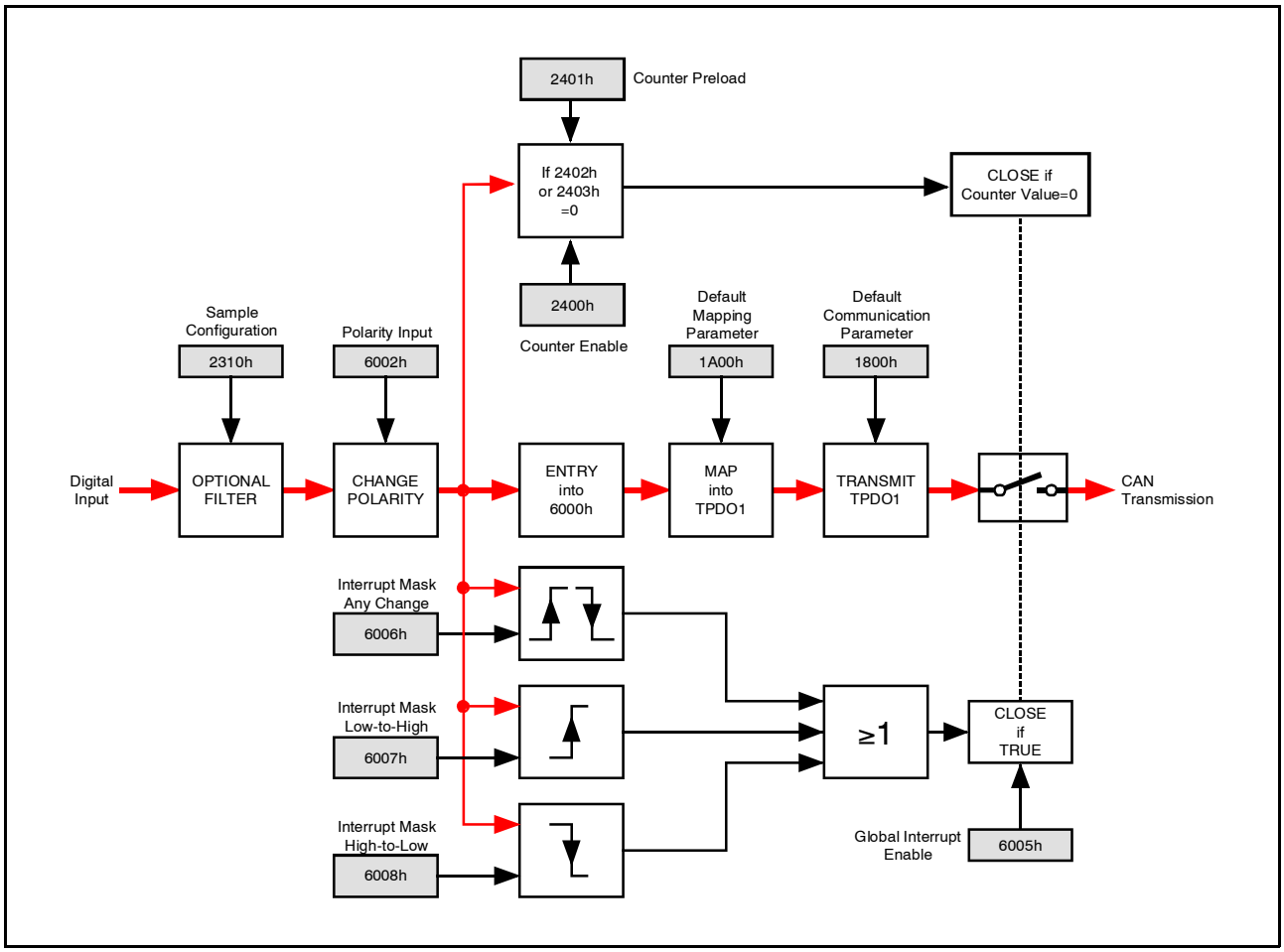


Fig. 16: Overview of the objects for the digital inputs

8.8.3 Interrelation of the Implemented Objects of the Digital Outputs

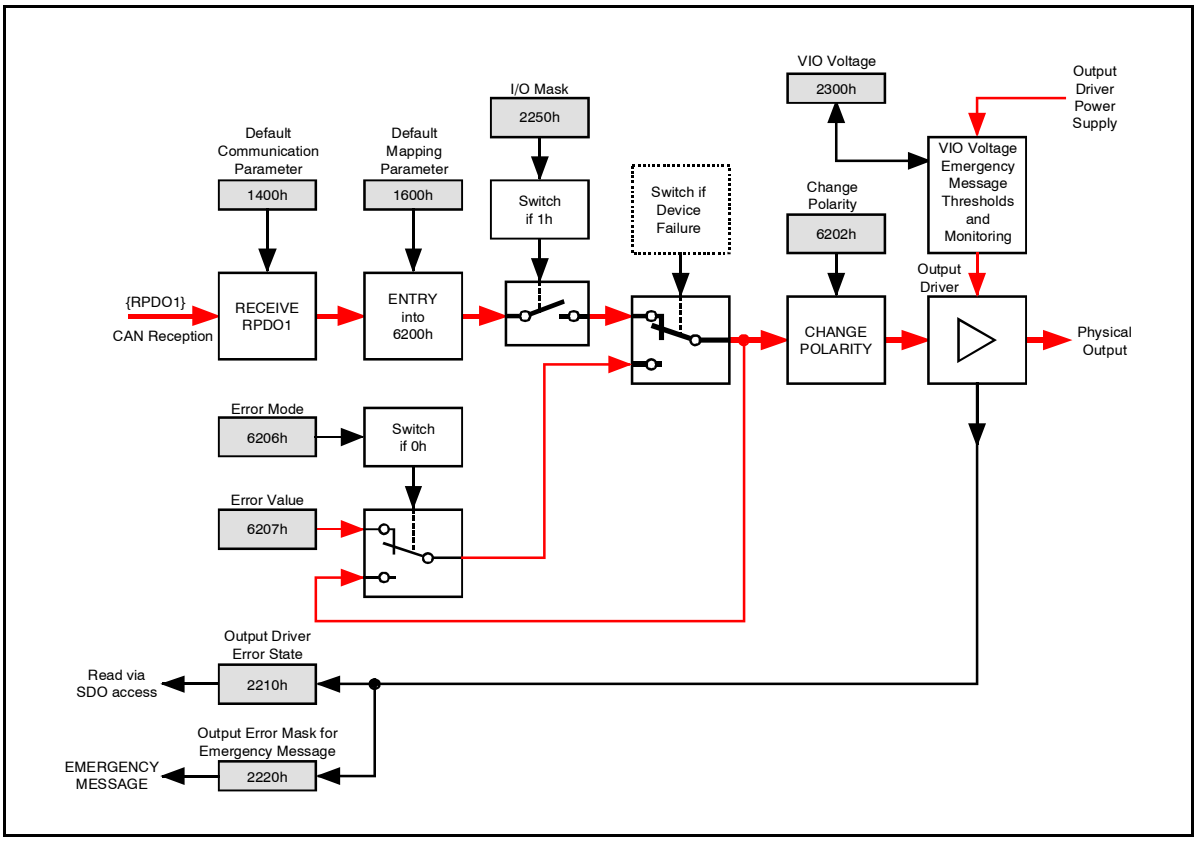


Fig. 17: Overview of the objects for the digital outputs

8.8.4 Read Input 8-Bit (6000_h)

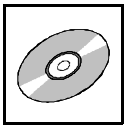
Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
6000	0	*no_of_entries*	1	1	unsigned 8	ro
	1	*read_input_DI8-DI1*	0...FF	-	unsigned 8	ro

Assignment of the variables *read_input_DI8-DI1*:

Index: 6000_h, Sub-index: 1

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Input:	*DI8*	*DI7*	*DI6*	*DI5*	*DI4*	*DI3*	*DI2*	*DI1*

An input bit is read as '1', if the corresponding input is active, i.e. voltage is 'on' (if the according bit in object 6002_h 'Polarity Input 8-Bit' is set to '0').



8.8.5 Polarity Input 8-Bit (6002_h)

With this object the polarity of the 8 digital inputs can be set individually.

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
6002	0	*no_of _entries*	1	1	unsigned 8	rw
	1	*polarity_input_DI8-DI1*	0...FF	0	unsigned 8	rw

Assignment of the parameter *polarity_input_DI8-DI1*:

Index: 6002_h, Sub-index: 1

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Input:	*DI8*	*DI7*	*DI6*	*DI5*	*DI4*	*DI3*	*DI2*	*DI1*

Bit value *DIx*	Input
1	inverted
0	not inverted

8.8.6 Filter Constant Input 8-Bit (6003_h)

With this parameter the digital filter can be individually activated for each input. Via the object 2310_h the sample rate of the digital inputs is defined.

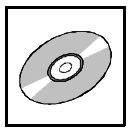
Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
6003	0	*no_of _entries*	1	1	unsigned 8	rw
	1	*filter constant _DI8-DI1*	0...FF	0	unsigned 8	rw

Assignment of the parameter *filter constant_DI8-DI1*:

Index: 6003_h, Sub-index: 1

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Input:	*DI8*	*DI7*	*DI6*	*DI5*	*DI4*	*DI3*	*DI2*	*DI1*

Bit value *DIx*	Input
1	filter enabled
0	filter disabled



8.8.7 Global Interrupt Enable Digital (6005_h)

Index [Hex]	Sub-index	Description	Value range [Boolean]	Default	Data type	R/W
6005	0	*Global Interrupt Enable-Bit* = false : interrupts disabled *Global Interrupt Enable-Bit* = true : interrupts enabled	True, False	1	boolean	rw

8.8.8 Interrupt Mask Any Change 8-Bit (6006_h)

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
6006	0	*no_of_entries*	1	-	unsigned 8	const
	1	*IRQ-mask_any_8_DI8-DI1*	0...FF	FF	unsigned 8	rw

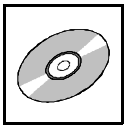
Assignment of parameter *IRQ-mask_any_8_DI8-DI1*:

Index: 6006_h, Sub-index: 1

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Input:	*DI8*	*DI7*	*DI6*	*DI5*	*DI4*	*DI3*	*DI2*	*DI1*

This object determines, which input port lines shall activate an interrupt by **rising or falling edge** detection.

Bit-value *DIx*	Interrupt-enable
0	interrupt disabled
1	interrupt enabled



8.8.9 Interrupt Mask Low to High 8-Bit (6007_h)

Index [Hex]	Sub-index [Dec]	Description	Value range [Hex]	Default	Data type	R/W
6007	0	*no_of_entries*	1	1	unsigned 8	ro
	1	*IRQ-mask_LH_8_DI8-DI1*	0...FF	0	unsigned 8	rw

Assignment of the parameter *IRQ-mask_LH_8_DI8-DI1*:

Index: 6007_h, Sub-index: 1

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Input:	*DI8*	*DI7*	*DI6*	*DI5*	*DI4*	*DI3*	*DI2*	*DI1*

For activated IRQ an interrupt is generated by **rising edge** detection of the input signal.

Bit-value *DIx*	Interrupt-enable
0	interrupt disabled
1	interrupt enabled

8.8.10 Interrupt Mask High to Low 8-Bit (6008_h)

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
6008	0	*no_of_entries*	1	1	unsigned 8	ro
	1	*IRQ-mask_HL_8_DI8-DI1*	0...FF	0	unsigned 8	rw

Assignment of the parameter *IRQ-mask_HL_8_DI8-DI1*:

Index: 6008_h, Sub-index: 1

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Input:	*DI8*	*DI7*	*DI6*	*DI5*	*DI4*	*DI3*	*DI2*	*DI1*

For activated IRQ an interrupt is generated by **falling edge** detection of the input signal.

Bit Value *DIx*	Interrupt enable
0	interrupt disabled
1	interrupt enabled

8.8.11 Write Output 8-Bit (6200_h)

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
6200	0	*no_of_entries*	1	1	unsigned 8	const
	1	*write_output_DO8-DO1*	0...FF	0	unsigned 8	rw

Assignment of the variable *write_output_DO8-DO1*:

Index: 6200_h, Sub-index: 1

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Output:	*DO8*	*DO7*	*DO6*	*DO5*	*DO4*	*DO3*	*DO2*	*DO1*

If an output bit is set to '1', the corresponding output is activated, i.e. the output voltage is 'on'.

8.8.12 Change Polarity Output 8-Bit (6202_h)

This object defines the state of the 8 digital outputs. Every output can be inverted individually.

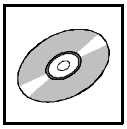
Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
6202	0	*no_of_entries*	1	1	unsigned 8	const
	1	*change polarity output_DO8-DO1*	0...FF	0	unsigned 8	rw

Assignment of the parameter *change_polarity_output_DO8-DO1*:

The parameter determines which digital output can be inverted.

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Contents:	*DO8*	*DO7*	*DO6*	*DO5*	*DO4*	*DO3*	*DO2*	*DO1*

Setting an output bit to ‘1’ inverts the corresponding output.



8.8.13 Error Mode Output 8-Bit (6206_h)

The error mode is evaluated if the module is in the *Stopped* state. This object in combination with object 1029_h indicates, whether an output is set to an error-value defined in object 6207_h in case of an internal device error.

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
6206	0	*no_of_entries*	1	1	unsigned 8	const
	1	*error_mode_output_DO8-DO1*	0...FF	FF_h	unsigned 8	rw

Assignment of the parameter *error_mode_output_DO8-DO1*:

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Contents:	*DO8*	*DO7*	*DO6*	*DO5*	*DO4*	*DO3*	*DO2*	*DO1*

Value	Meaning
1	Output shall take the pre-defined value specified in object 6207_h.
0	Output shall be kept unchanged if an error occurs.

8.8.14 Error Value Output 8-Bit (6207_h)

Provided that the corresponding error mode (object 6206_h) is active, device failures shall set the output to the value configured by this object.

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
6207	0	*no_of_entries*	1	1	unsigned 8	const
	1	*error_value_output_DO8-DO1*	0...FF	0	unsigned 8	rw

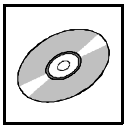
Assignment of the parameter *error_value_output_DO8-DO1*:

Index: 6207_h, Sub-index: 1

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Contents:	*DO8*	*DO7*	*DO6*	*DO5*	*DO4*	*DO3*	*DO2*	*DO1*

The parameter contains the value, the outputs shall be set to, in case of fault.

Value	Meaning
1	Output shall be set to '1' (enabled) in case of fault, if object 6206_h is enabled.
0	Output shall be set to '0' (disabled) in case of fault, if object 6206_h is enabled.



8.9 Manufacturer Specific Profile Area

8.9.1 Implemented Objects 2210_h ... 2403_h

Index [Hex]	Name	Data Type
2210	*Output Errors 8-Bit*	unsigned 8
2220	*Output Error Mask Bit 1 to 8*	unsigned 8
2250	*I/O Mask 8-Bit*	unsigned 8
2300	*VIO Voltage 16-Bit*	unsigned 16
2310	*Sample Configuration*	unsigned 16
2400	*Counter Enable*	unsigned 8
2401	*Counter Preload*	unsigned 32
2402	*Counter Value 16- Bit*	unsigned 16
2403	*Counter Value 32-Bit*	unsigned 32

8.9.2 Output Errors 8-Bit (2210_h)

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
2210	0	*no_of_entries*	2	2	unsigned 8	ro
	1	*output_error_open_DO8-DO1*	0...FF	0	unsigned 8	ro
	2	*output_error_short_DO8-DO1*	0...FF	0	unsigned 8	ro

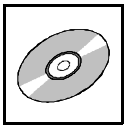
The bit mask, returned by this object, indicates at which outputs errors have occurred (error states: open, overload).

Assignment of the parameters *output_error_open_DO8-DO1, resp. output_error_short_DO8-DO1*:

Index: 2210_h, Sub-index: 1/2

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Output:	*error_ DIO8*	*error_ DIO7*	*error_ DIO6*	*error_ DIO5*	*error_ DIO4*	*error_ DIO3*	*error_ DIO2*	*error_ DIO1*

The output bits are set to ‘1’, in case of error at the corresponding output.



8.9.3 Output Error Produces Emergency 8-Bit (2220_h)

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
2220	0	*no_of_entries*	2	2	unsigned 8	ro
	1	*output_error_mask_open_DO8-DO1*	0...FF	FF	unsigned 8	rw
	2	*output_error_mask_short_DO8-DO1*	0...FF	FF	unsigned 8	rw

This object defines a bit mask, that determines if an Emergency Message is send when an error occurs or not (please refer also objects 1003_h and 1014_h).

Assignment of the parameters *output_error_mask_open_DO8-DO1*, resp. *output_error_mask_open_DO8-DO1*:

Index: 2220_h, Sub-index: 1/2

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Output:	*DIO8*	*DIO7*	*DIO6*	*DIO5*	*DIO4*	*DIO3*	*DIO2*	*DIO1*

Note:
An Emergency Message is transmitted, if the first error of an output driver occurs and the according bit in object *Output Error Mask Bit 1 to 8* is set to '1'. If another error occurs at another output no additional Emergency Message is generated.
If the last error message disappears, again an Emergency Message is transmitted to indicate the error disappearance.

Note:
The object *Output Error Mask Bit 1 to 8* has no effect on the current error state that is returned in the object *Output Errors 8-Bit* (2210_h).

8.9.4 I/O Mask 8-Bit (2250_h)

Index [Hex]	Sub-index	Description	Value range [Hex]	Default	Data type	R/W
2250	0	*no_of_entries*	1	-	unsigned 8	ro
	1	*I/O_mask_DIO8-DIO1*	0...FF	0	unsigned 8	rw

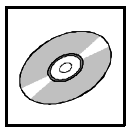
This object defines a bit mask, that determines which of the 8 I/O channels shall be used as input and which as output.

Assignment of the parameter *I/O_mask_DIO8-DIO1*:

Index: 2250_h, Sub-index: 1

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Output:	*DIO8*	*DIO7*	*DIO6*	*DIO5*	*DIO4*	*DIO3*	*DIO2*	*DIO1*

Value	Meaning
1	channel is used as output
0	channel is used as input



8.9.5 VIO Voltage 16-Bit (2300_h)

Index [Hex]	Sub-index	Description	Value range [Hex]	Default [Hex]	Data type	R/W
2300	0	*no_of_entries*	4	4	unsigned 8	ro
	1	*VIO_voltage*	0...FFFF	-	unsigned 16	ro
	2	*VIO_low_emergency_voltage*	0...FFFF	B4	unsigned 16	rw
	3	*VIO_high_emergency_voltage*	0...FFFF	118	unsigned 16	rw

With this object the operating voltage for the digital outputs can be read as analog value. Additionally the voltage limits for the operating voltage of the output drivers (that generate an Emergency Message if exceeded) can be defined.

Evaluation of the voltage values for sub-index 1, 2, 3:

Variable_value = (Voltage in V) x 10_d

Assignment of the variable *VIO_Voltage*:

This variable returns the measured value of the supply voltage of the digital outputs.

Example:

Value of the variable *VIO_Voltage* = $F0_h = 240_d$

Voltage in V = value of the variable / $10_d = 240_d / 10_d = 24.0$ V

Assignment of the parameters *VIO_low_emergency_voltage*:

This variable defines the voltage limit that initiates an Emergency Message, if the value falls below.

Example:

0 = disable

$B4_h$ = 18 V -> voltage limit below an Emergency Message is initiated

Assignment of the parameters *VIO_high_emergency_voltage*:

This variable defines the voltage limit that initiates an Emergency Message, if the value falls below.

Example:

0 = disable

118_h = 28 V -> voltage limit above an Emergency Message is initiated

8.9.6 Sample Configuration (2310_h)

Index [Hex]	Sub-index	Description	Value range [Hex]	Default [Hex]	Data type	R/W
2310	0	*no_of_entries*	1	-	unsigned 8	ro
	1	*sample_rate*	C8...FFFF	$C8_h$	unsigned 16	rw

This object enables and defines the filter of the digital inputs.

Assignment of the parameter *sample_rate*:
This parameter defines the sample rate that is used to read the digital inputs. The sample rate is set in steps of 0.5 µs. The minimum value is 100 µs.

Value of parameter *sample_rate* [Dec]	Resulting sample rate [µs]
199	current parameter value is not changed (old value will stay valid + SDO error message)
200	100 (default value)
201	1005
:	:
65535	327675

8.9.7 Function of the Counter

A 32-bit counter can be enabled (object 2400_h) for each input. In the default setting the edge-sensitive counters count the rising edges of the input signals.
For falling edge detection the parameter *Polarity_Input* (object 6002_h) has to be adapted.

The initial value of the 32-bit counter is *counter_preload_x*, which is defined in object 2401_h. With every step the counter will be decremented by '1'. The *counter_value_x* (see objects 2402_h and 2403_h) contains the current value of the counter *x* (*x* = 1...8).

If the 32-bit counter (*counter_value_x_32-bit*) results in '0', an event is generated and the *counter_value_x_32-bit* is set to the *counter_preload_x*-value again.

The interrelation of the objects of the digital inputs are described in figure 16 on page 78.

Note:
If the counter function of an input is used, it is strongly recommended to disable the local interrupts via the *Interrupt Mask*-objects 6006_h, 6007_h and 6008_h (reduces bus load of the CAN bus).

The maximum attainable input clock rate results from the sample rate of the inputs (see object *Sample Configuration* 2310_h):

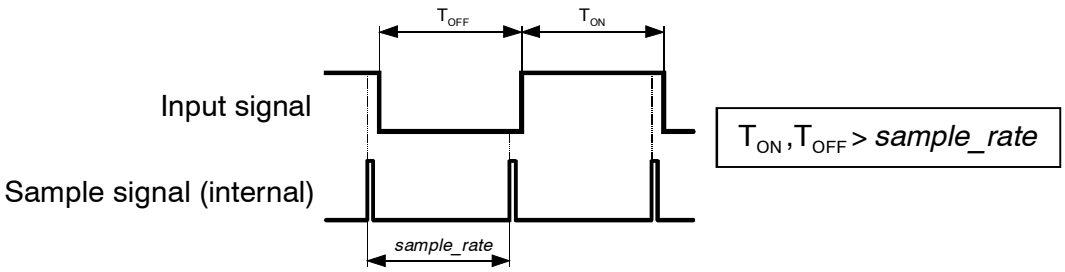


Fig. 18: Interrelation of maximum count frequency and sample rate

So the absolute maximum of the countable input frequency is 5 kHz.

8.9.8 Counter Enable (2400_h)

Index [Hex]	Sub-index	Description	Value range [Hex]	Default [Hex]	Data type	R/W
2400	0	*counter_enable*	0...FF	0	unsigned 8	rw

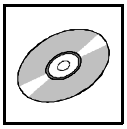
With this object the counters 1..8 can be enabled. The counters count the number of changes of the digital inputs 1..8. In this process either changes with rising or with falling edges are counted.
By default rising edges are counted. To count the falling edges, adapt the parameter *Polarity_Input* (object 6002_h).

Assignment of the variable *counter enable*:

Index: 2400_h, Subindex: 0

Bit:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Port:	*counter8*	*counter7*	*counter6*	*counter5*	*counter4*	*counter3*	*counter2*	*counter1*

Value	Description
1	counter enabled
0	counter disabled



8.9.9 Counter Preload (2401_h)

Index [Hex]	Sub-index	Description	Value range [Hex]	Default [Hex]	Data type	R/W
2401	0	*no_of_ objects*	1...8	8	unsigned 8	ro
	1	*counter_preload_1*	0... FFFF FFFF	0	unsigned 32	rw
	2	*counter_preload_2*		0	unsigned 32	rw
	3	*counter_preload_3*		0	unsigned 32	rw
	4	*counter_preload_4*		0	unsigned 32	rw
	5	*counter_preload_5*		0	unsigned 32	rw
	6	*counter_preload_6*		0	unsigned 32	rw
	7	*counter_preload_7*		0	unsigned 32	rw
	8	*counter_preload_8*		0	unsigned 32	rw

In this object the initial values *counter_preload_1...8* of the counters 1...8 are defined.
Beginning from the initial value *counter_preload_x* the counter value is decremented by '1', if a change occurs.

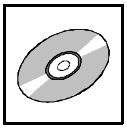
8.9.10 Counter Value 32-Bit (2403h)

Index [Hex]	Sub-index	Description	Value range [Hex]	Default [Hex]	Data type	R/W
2403	0	*no_of_ objects*	1...8	8	unsigned 8	ro
	1	*counter_value_1_32-bit*	0... FFFF FFFF	0	unsigned 32	ro
	2	*counter_value_2_32-bit*		0	unsigned 32	ro
	3	*counter_value_3_32-bit*		0	unsigned 32	ro
	4	*counter_value_4_32-bit*		0	unsigned 32	ro
	5	*counter_value_5_32-bit*		0	unsigned 32	ro
	6	*counter_value_6_32-bit*		0	unsigned 32	ro
	7	*counter_value_7_32-bit*		0	unsigned 32	ro
	8	*counter_value_8_32-bit*		0	unsigned 32	ro

This object contains the current 32-bit values *counter_value_x_32-bit* of the counters 1...8 (see object 2400_h also).

The counters count the number of changes of the digital inputs 1..8. Beginning with the 32-bit initial value *counter_preload_x* the value is decremented by '1', if a change occurs (x = 1...8).

An event is generated, if the value of the **32-bit** counter results in '0'.
If the value of *counter_preload_x* is set to '0000.0000', the preload is 2^{32}!



8.9.11 Counter Value 16-Bit (2402h)

Index [Hex]	Sub-index	Description	Value range [Hex]	Default [Hex]	Data type	R/W
2402	0	*no_of_ objects*	1...8	8	unsigned 8	ro
	1	*counter_value_1_16-bit*	0...FFFF	0	unsigned 16	ro
	2	*counter_value_2_16-bit*		0	unsigned 16	ro
	3	*counter_value_3_16-bit*		0	unsigned 16	ro
	4	*counter_value_4_16-bit*		0	unsigned 16	ro
	5	*counter_value_5_16-bit*		0	unsigned 16	ro
	6	*counter_value_6_16-bit*		0	unsigned 16	ro
	7	*counter_value_7_16-bit*		0	unsigned 16	ro
	8	*counter_value_8_16-bit*		0	unsigned 16	ro

The parameters *counter_value_x_16-bit* contain the least significant 16 bit of the current 32-bit values of the counters 1...8 (see object 2400_h also)(x = 1...8).

The following table shows the position of the 16-bit counter value *counter_value_x_16-bit* in the 32-bit counter value *counter_value_x_32-bit:*

Bit 31...	...Bit 0
counter_preload_x	
counter_value_x_32-bit	
Bit 31... ...Bit 16	**Bit 15... ...Bit 0**
-	***counter_value_x_16-bit***

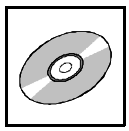
8.10 Firmware Update via DS 302-Objects ($1F50_h$...$1F52_h$)

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!

In normal DS 301 mode the object $1F50_h$ can not be accessed.
The objects $1F51_h$ and $1F52_h$ are available in normal DS 301-mode, too.
For further information about the objects and the firmware-update please refer to the manual 'Firmware-Update via DS 302 Objects'.

Index [Hex]	Sub-index	Description	Data type	R/W
1F50	0	Boot-Loader: Firmware download	domain	rw
1F51	1	Boot-Loader: FLASH command	unsigned 8	rw
1F52	0,1,2	Boot-Loader: Firmware date	unsigned 32	ro



8.10.1 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-DIO8 differs from the value range specified in the DS 302.
For further information about object $1F51_h$ and the firmware-update please refer to the manual 'Firmware Update via DS 302 Objects'

EG-KONFORMITÄTSERKLÄRUNG
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*	**CAN-CBX-DIO8**
Typ, Modell, Artikel-Nr. *Type, Model, Article No.*	**C.3010.xx**
die Anforderungen der Normen *fullfills the requirements of the standards*	**EN 61000-6-4 (01/2007),** **EN 61000-6-2 (08/2005)**
gemäß folgendem Prüfbericht erfüllt. *according to test certificate.*	**H-K01-0247-07**
Das Produkt entspricht damit den EG-Richtlinien *Therefore the product corresponds to the EU-Directives*	**89/336/EWG (23.05.1989),** **92/31/EWG (28.04.1992)**

Diese Erklärung gilt für alle Exemplare, die das CE-Zeichen tragen und verliert ihre Gültigkeit, wenn Veränderungen am Produkt vorgenommen werden.
This declaration is valid for all units with the CE label on it and it lose its validity if a modification is done on the product.

Name / *Name*	Dr. Ing. Werner Schulze
Funktion / *Title*	Geschäftsführer / Managing Director
Datum / *Date*	Hannover, den 07.03.2008

Rechtsgültige Unterschrift / *authorized Signature*

.....

I:\Texte\Doku\MANUALS\CAN\CBX\DIO8\Conform_CAN_CBX_DIO_02.wpd