CAN-CBX-COM2

CAN - RS-232, RS-422, RS-485 or TTY-Interface

Manual

to Product C.3055.02, C.3055.03

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

N O T E

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

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

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

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

Trademark Notices

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

Document-File:	I:\Texte\Doku\MANUALS\CAN\CBX\CAN-CBX-COM2\Englisch\CAN-CBX-COM2_Manual_en_11.wpd
Date of print:	2015-02-05

Software:	CANopen
Revision:	V1.ODJ

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.

Revision	Chapter	Changes versus previous version
1.0	-	First English Version
1.1	-	Typographical Conventions inserted
	2.	Note to termination connector in step 2 deleted
	3.	Chapter revised, notes inserted
	4.5	Chapter “Software Support” moved
	8.8	Description of Self-Start-Feature ($1F80_h$, $1F91_h$) inserted
	9.	References revised
	10.	EU-Conformity updated

Technical details are subject to change without further notice.

Safety Instructions

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

Qualified Personal

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

Conformity

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

Warning: In a residential, commercial or light industrial environment the CBX-module may cause radio interferences in which case the user may be required to take adequate measures.

Intended Use

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

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

Service Note

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

Disposal

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

Contents

Typographical Conventions

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

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

The following indicators are used to highlight noticeable descriptions.

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

Note:
Notes to point out something important or useful.

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

1. Overview

1.1 Description of the Module

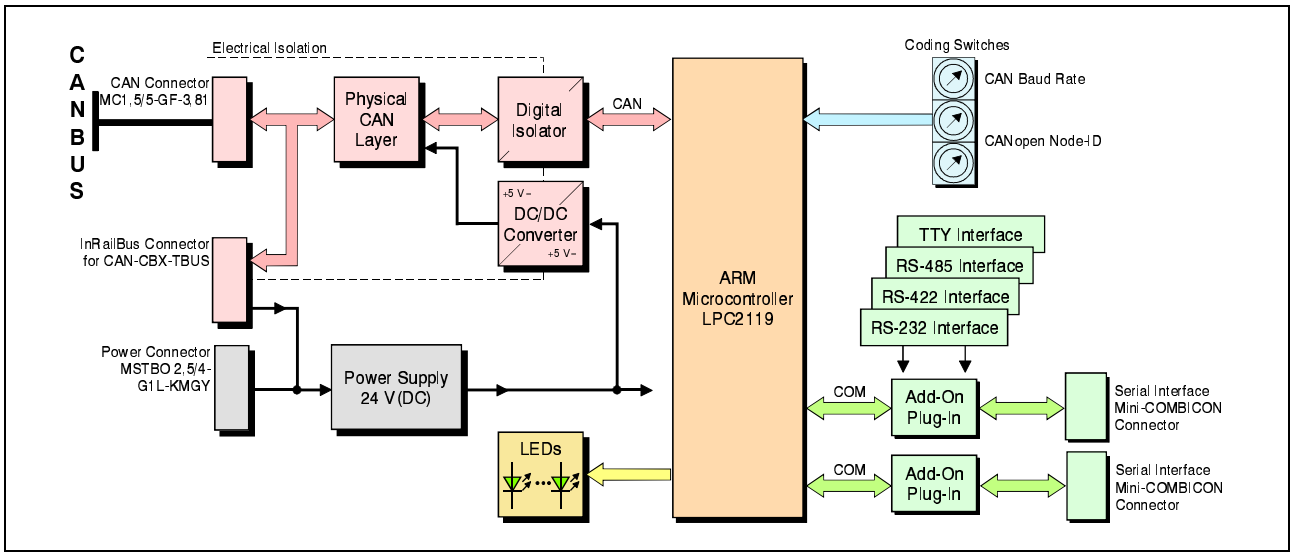


Fig. 1: Block circuit diagram of the CAN-CBX-COM2 module

The CAN-CBX-COM2 module is a CANopen® Module, that integrates the communication of two serial interfaces into a CANopen network.
The basic version comes with two RS-232 interfaces.
The serial interface COM B can be individually configured as RS-232, RS-422, RS-485 or TTY interface via add-on modules (Piggybacks).
As CAN-CBX module the CAN-CBX-COM2 can either be operated as individual module or it can be connected to the InRailBus.

The CAN-CBX-COM2 module is equipped with an ARM7®-Microcontroller, which operates as controller for the serial interfaces and the CAN-Bus. The CAN data are buffered in the local SRAM. The firmware is held in the flash. The parameters are saved in a serial EEPROM.

The CAN-CBX-COM2 features the possibility to connect the power supply and the CAN bus signals via individual connectors or via the InRailBus connector (TBUS-connector) integrated in the mounting rail. Individual modules can then be removed without interrupting the bus signals.

The CAN interface is designed according to ISO11898-2 high-speed layer with electrical isolation and supports bit rates up to 1 Mbit/s.

The firmware supports CANopen according to CiA® 301 [1].
The CANopen node number and the CAN-bit rate can be easily set via coding switches.

2. Quick Start

Step	Action	See page
	**Read the safety instructions at the beginning of this document carefully, before you start with the hardware installation!**	4
	**Please note the chapters “Installation and Wiring of the Module“ and “Correctly Wiring Electrically Isolated CAN Networks“!**	22, 37
1	Mount the CAN-CBX-COM2 module and connect the interfaces (power supply voltage, CAN, serial interfaces).	12
2	Please note that the CAN bus has to be terminated at both ends! esd offers special T-connectors and termination connectors. Additionally the CAN_GND signal has to be connected to earth at exactly one point in the CAN network. A CAN node with electrical connection to earth potential acts as an earth potential.	-
3	Set the baud rate (only if it differs from the default setting) The default baud rate is 1 MBit/s. The baud rate can be set via the coding switch BAUD, as described in chapter: “Setting the Node-ID via Coding Switch”.	18
4	Set the module number (node-ID). The node ID can be set via the coding switches LOW and HIGH. It may be set to values between 1 and 127 (01-7F$_h$).	17
5	Apply the 24 V power supply voltage.	-
6	Start the module with the NMT-Start Command	-

CAN-Identifier	Len	Data	
0	2 Byte	1	0 (Node-ID, 0 = start all modules)

<table>
<tr>
<td>7</td>
<td>Transmit data CAN -> serial
<table>
<tr><th>CAN-Identifier</th><th>Len</th><th>Data</th></tr>
<tr><td>200_h + Node-ID</td><td>8 Byte</td><td>0...8 byte user data
COM A</td></tr>
<tr><td>400_h + Node-ID</td><td>8 Byte</td><td>0...8 byte user data
COM B</td></tr>
</table>
</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>Receive data serial -> CAN
<table>
<tr><th>CAN-Identifier</th><th>Len</th><th>Data</th></tr>
<tr><td>180_h + Node-ID</td><td>8 Byte</td><td>0...8 byte user data
COM A</td></tr>
<tr><td>380_h + Node-ID</td><td>8 Byte</td><td>0...8 byte user data
COM B</td></tr>
</table>
</td>
<td>-</td>
</tr>
</table>

3. Hardware Installation

3.1 Connecting Diagram

3.1.1 Power Supply and CAN

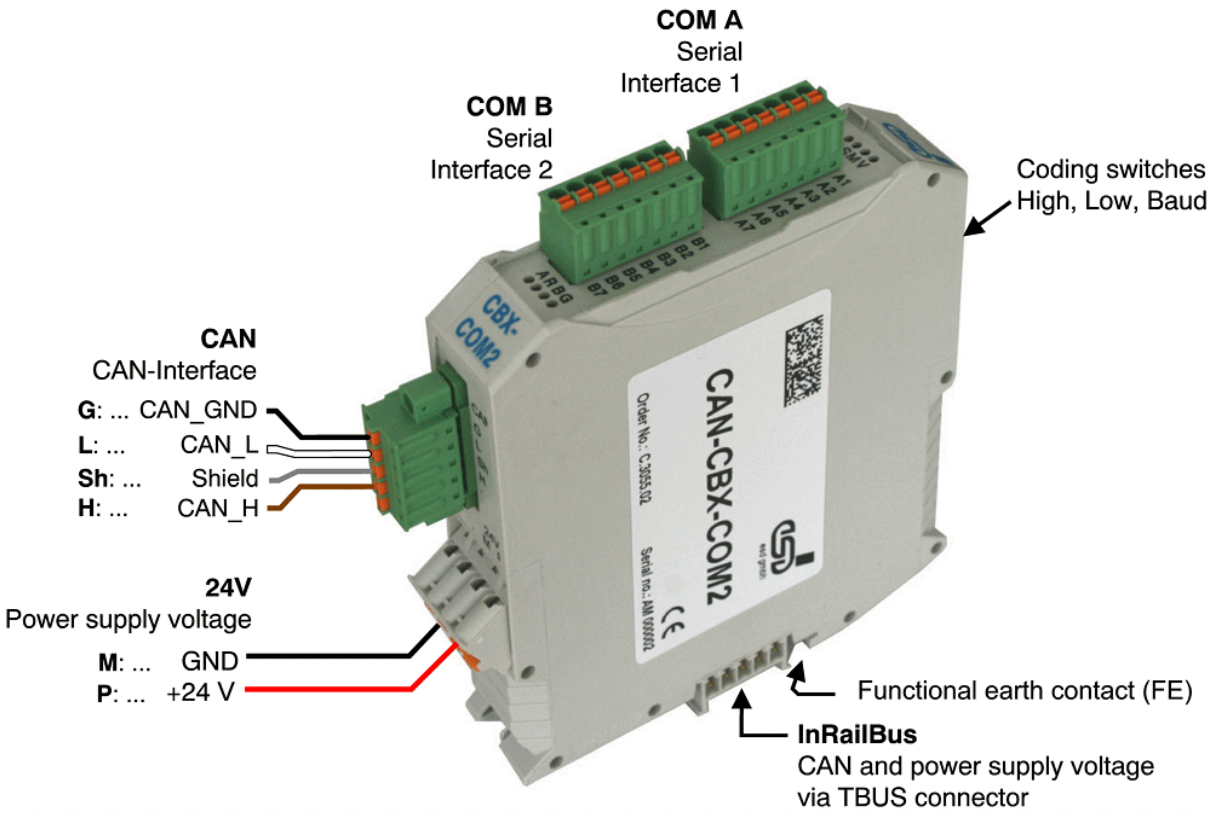


Fig. 2: Connections of the CAN-CBX-COM2 module

Note:
Please refer to page 36 for information on conductor connection and conductor cross section.
The connector pin assignments can be found on page 26 and following.

3.2 LED-Display

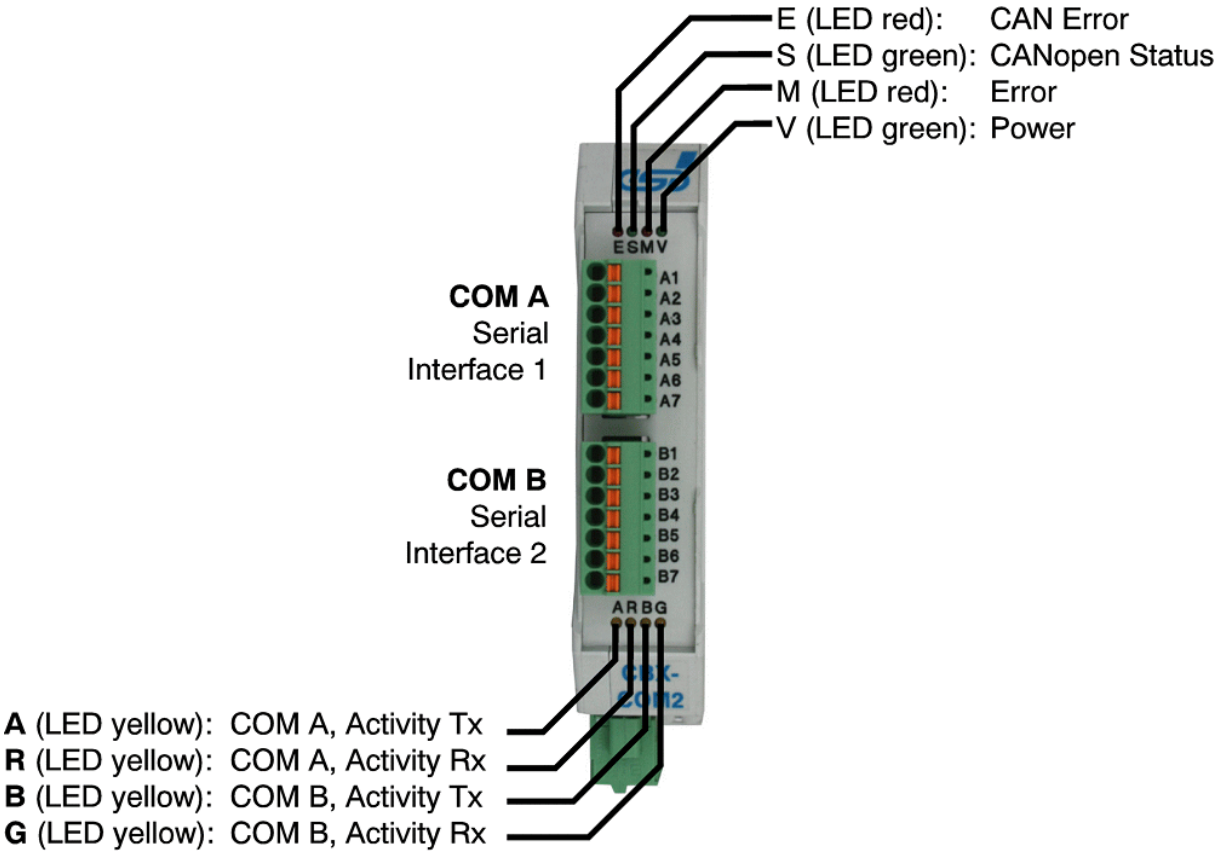


Fig. 3: Position of the LEDs in the front panel

The CAN-CBX-COM2 module is equipped with four Status LEDs and four Activity LEDs.

3.2.1 Indication of Activity-LEDs

LED	Name	Colour	Indication	Meaning
A	COM A, Activity Tx	yellow	off	CAN-CBX-COM2 does not transmit data on COM A
			blinking	CAN-CBX-COM2 transmits data on COM A
R	COM A, Activity Rx	yellow	off	CAN-CBX-COM2 does not receive data on COM A
			blinking	CAN-CBX-COM2 receives data on COM A
B	COM B, Activity Tx	yellow	off	CAN-CBX-COM2 does not transmit data on COM B
			blinking	CAN-CBX-COM2 transmits data on COM B
G	COM B, Activity Rx	yellow	off	CAN-CBX-COM2 does not receive data on COM B
			blinking	CAN-CBX-COM2 receives data on COM B

Table 2: Indication of Activity-LEDs

3.2.2 Flash Rates of the Status LEDs

The CAN-CBX-COM2 module is equipped with 4 status LEDs.
The terms of the indicator states of the LEDs are chosen in accordance with the terms recommended by the CiA [3]. The indicator states are described in the following chapters. In principle 8 flash rates are defined.

Indicator state	Flash rate
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)

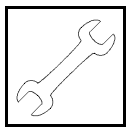
Table 3: Flash rates

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

3.2.3 Operation of the CAN-Error LED

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

Table 4: Indicator states of the red CAN Error-LED



3.2.4 Operation of the CANopen-Status LED

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

Table 5: Indicator states of the CANopen Status-LED

3.2.5 Operation of the Error-LED

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

Table 6: Indicator state of the Error-LED

3.2.6 Operation of the Power-LED

LED indication			Display function	
Label	Name	Colour	Indicator state	Description
V	Power	green	off	no power supply voltage; or the module is in Bootloader-Mode, this state is indicated by the CANopen status-LED (3 Flashes) (see page 16)
			on	power supply voltage is on and application software is running

Table 7: Indicator state of the Power-LED

3.2.7 Special Indicator States

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

LED indication	Description
- CAN-Error LED: on and - CANopen-Status LED: 3 flashes	Invalid node ID: The coding switches for the node-ID are set to an invalid ID-value, when switching on. The module will be stopped.

Table 8: Special Indicator States

3.2.8 Assignment of LED Labelling to the Name in the Schematic Diagram

Labelling on the CAN-CBX-COM2	Name in the Schematic Diagram *[1]
E	LED240A
S	LED240B
M	LED240C
V	LED240D
A	LED320A
R	LED320B
B	LED320C
G	LED320D

*[1] The schematic diagram is not part of this manual.

3.3 Coding Switches

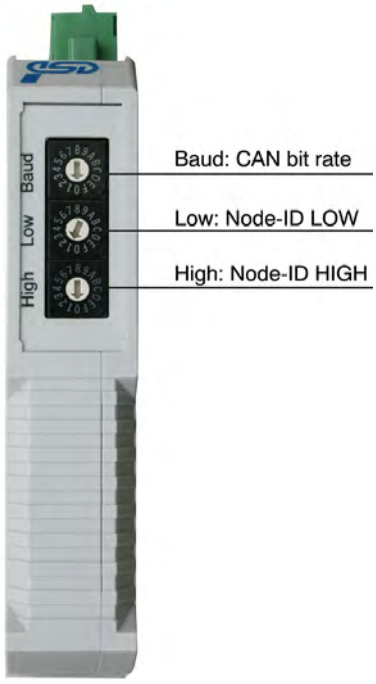


Fig. 4: Position of the coding switches

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

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

3.3.1 Setting the Node-ID via Coding Switch

The address range of the CAN-CBX-module can be set *decimal* from 1 to 127 or *hexadecimal* from 01_h to $7F_h$.
The three higher-order bits (higher-order nibble) can be set with coding switch **HIGH**, the four lower-order bits can be set with coding switch **LOW**.

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

3.3.2 Setting the Baud Rate

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

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

Setting	Bit rate [Kbit/s]
0	1000
1	$666.\overline{6}$
2	500
3	$333.\overline{3}$
4	250
5	166
6	125
7	100
8	$66.\overline{6}$
9	50
A_h	$33.\overline{3}$
B_h	20
C_h	12.5
D_h	10
E_h	800
F_h	$83.\overline{3}$ *[2]

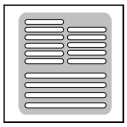
*[2] from firmware version 1.02 on

Table 9: Index of the baud rate

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

Labelling on the CAN-CBX-COM2	Name in the Schematic Diagram *[3]
Baud	SW401
Low	SW400
High	SW402

*[3)] The schematic diagram is not part of this manual.



4. Technical Data

4.1 General technical Data

Power supply voltage	nominal voltage: 24 V/DC input voltage range: 24 V ±20% current consumption (24 V, 20 ˚C): 30 mA
Connectors	24V (4-pin PCB plug connector with spring-cage connection, X100) - 24V-power supply voltage COM A (7-pin PCB plug connector, X300) - Serial interface 1 COM B (7-pin PCB plug connector, X310) - Serial interface 2 CAN (5-pin PCB plug connector, X400) - CAN interface InRailBus (5-pin TBUS-connector, X101) - CAN interface and power supply voltage via InRailBus Only for test and programming purposes (internal): X220 (10-pin socket connector)
Temperature range	0 ˚C ... +60 ˚C ambient temperature
Humidity	max. 90%, non-condensing
Protection class	IP20
Pollution degree	maximum permissible according to DIN EN 61131-2: Pollution Degree 2
Housing	Plastic housing for carrier rail mounting NS35/7,5 DIN EN 60715
Dimensions	width: 22.5 mm, height: 99 mm, depth: 114.5 mm (without connectors)
Weight	approx. 135 g

Table 9: General technical data

4.2 CPU

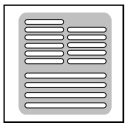
CPU	32 bit µC LPC2119FBD64
RAM	16 Kbyte integrated
Flash	128 Kbyte integrated
EEPROM	4 Kbyte

Table 10: Microcontroller

4.3 CAN Interface

Number	1
Connection	5-pin PCB plug connector with spring-cage connection or via InRailBus-connector (CAN-CBX-TBUS)
CAN Controller	integrated in LPC2119FBD64, ISO11898-1
Electrical isolation of CAN interfaces against other units	Isolation voltage U: 500 V (= withstand-impulse voltage according to DIN EN 60664-1)
Physical layer CAN	physical layer according to ISO 11898-2, transfer rate programmable from 10 Kbit/s up to 1 Mbit/s
Bus termination	has to be set externally if required

Table 11: Data of the CAN interface



4.4 Serial Interface

Number of serial interfaces	two individually configurable serial interfaces
Electrical isolation	none
Interface	Standard: - CAN-CBX-COM2 2x RS-232 (Hardware-Handshake)(esd order No.: C.3055.02) - CAN-CBX-COM2 1x RS-232 1x RS-485 (esd order No.: C.3055.03) Options: RS-422, RS-485, TTY-active/passive
Bit rate	RS-232: up to 115.2 Kbit/s RS-422: up to 115.2 Kbit/s RS-485: up to 115.2 Kbit/s TTY: up to 38.4 Kbit/s
Connector	7-pin PCB plug connector with spring-cage connection

Table 12: Data of the serial interface

4.5 Software Support

The firmware of the module supports the CiA® CANopen specification CiA 301 [1].

The CAN-CBX-COM2 EDS-file can be downloaded from the esd website www.esd.eu.

4.6 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:

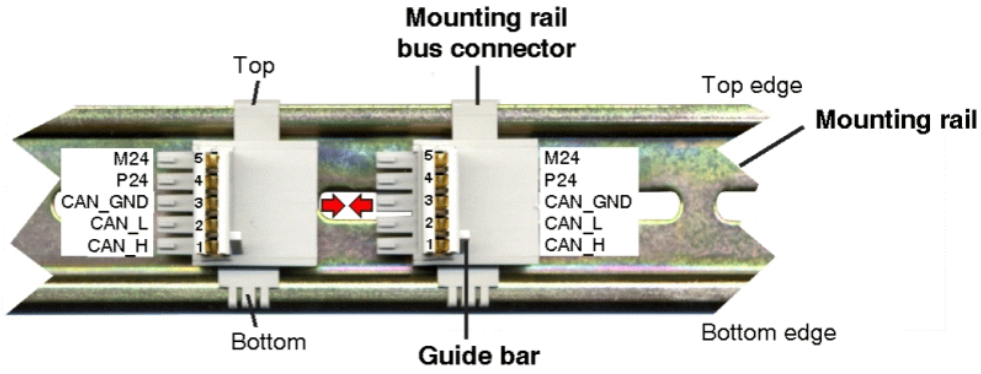


Figure 5: Mounting rail with bus connector

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.

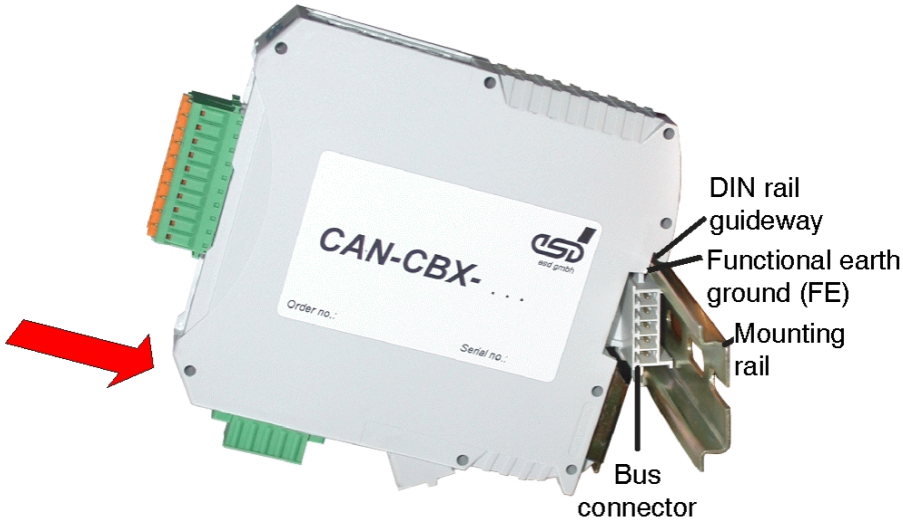


Figure 6 : Mounting CAN-CBX modules

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.

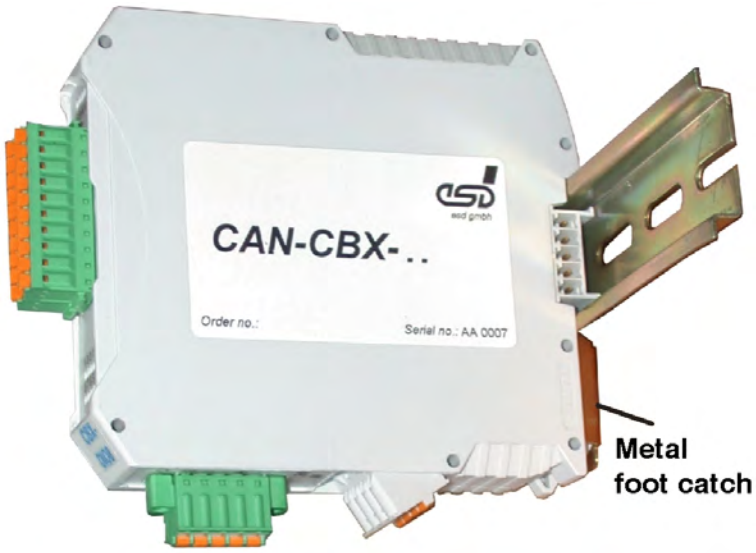


Figure 7: Mounted CAN-CBX module

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

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

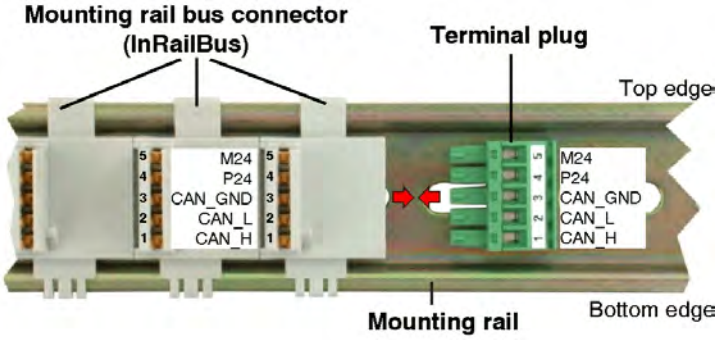


Fig. 8: Mounting rail with InRailBus and terminal plug

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

4.6.2 Connection of the Power Supply Voltage

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

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

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

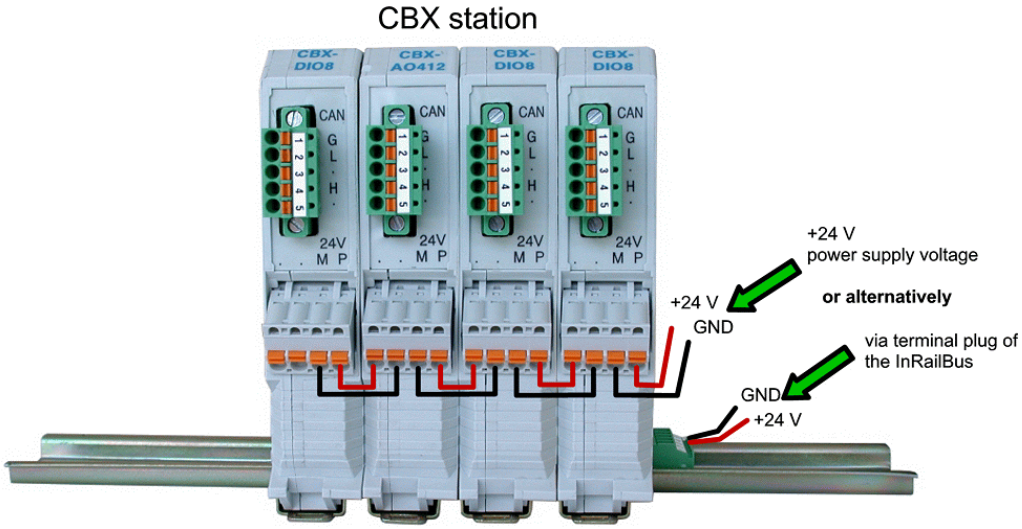


Fig. 9: Connecting the power supply voltage to the CAN-CBX station

Earthing of the Mounting Rail

Note:
The module is connected with the mounting rail via its functional earth contact.
This improves the stability against electromagnetic disturbances.
Thus the mounting rail shall be connected to an appropriate functional earth contact in the environment or in the installation.
Please note, that the impedance of the connection has to be kept low.
The functional earth contact of the module does not ensure electrical safety.

4.6.3 Connection of CAN

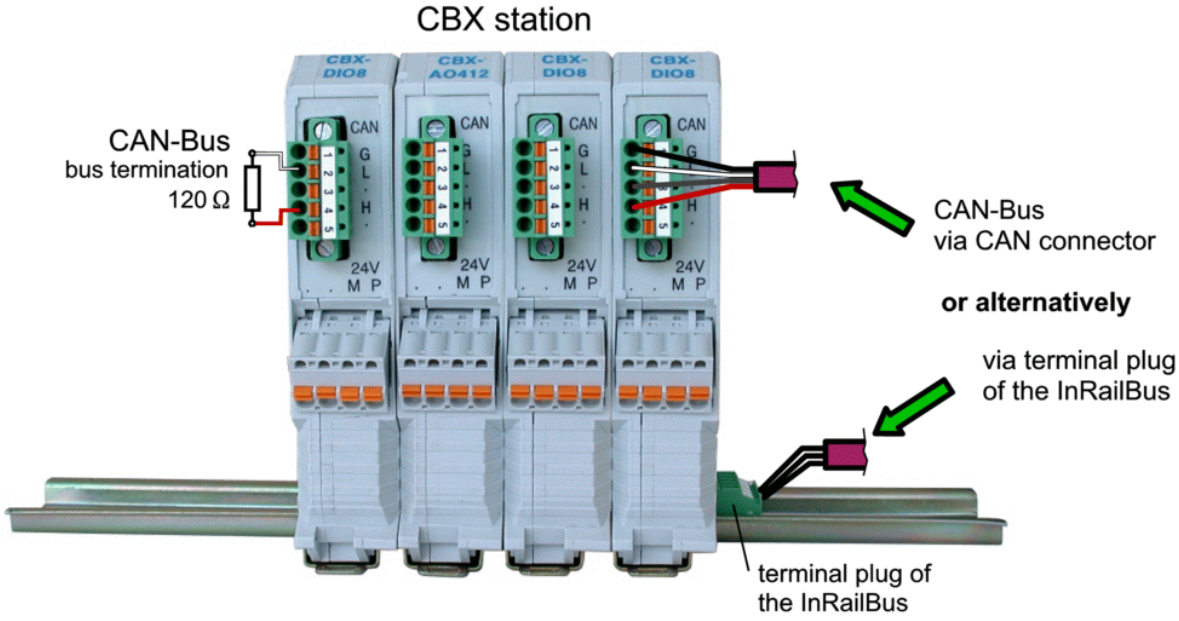


Fig. 10: Connecting the CAN signals to the CAN-CBX station

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

4.7 Remove the CAN-CBX Module from the InRailBus

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

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

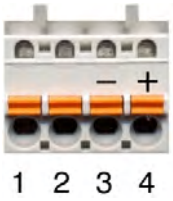
Note:
It is possible to remove individual devices from the CBX station without interrupting the InRailBus connection, because the contact chain will not be disrupted.

5. Connector Assignment

5.1 Power Supply Voltage 24 V (X100)

Device connector: Phoenix-Contact MSTBO 2,5/4-G1L-KMGY
Line connector: Phoenix-Contact FKCT 2,5/4-ST KMGY, 5.0 mm pitch, spring-cage connection,
Phoenix-Contact order no.: 19 21 90 0 (included in the scope of delivery)
For conductor connection and conductor cross section see page 36.

Pin Position:



Pin Assignment:

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

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

Please refer also to the connecting diagram on page 12.

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

Signal Description:

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

5.2 CAN

5.2.1 CAN Interface

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

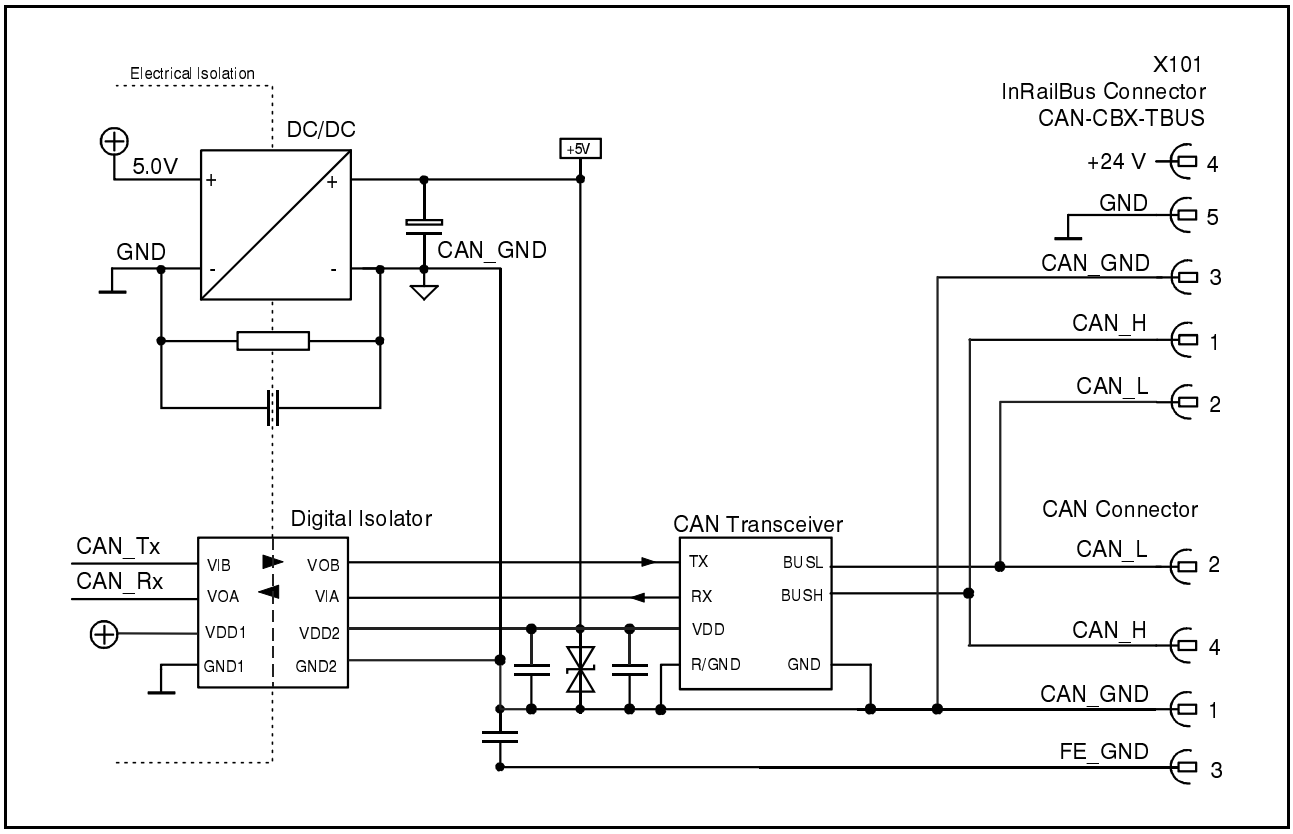
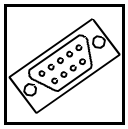


Fig. 11: CAN Interface

The CAN interface can be connected via the CAN connector or optionally via the InRailBus. Use the mounting-rail bus connector of the CBX-InRailBus (CAN-CBX-TBUS), see order information (page 124).



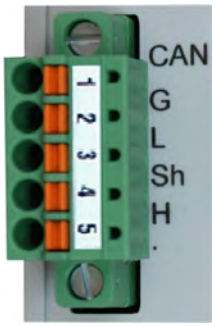
Connector Pin Assignment

5.2.2 CAN Connector

Device Connector: Phoenix-Contact MC 1,5/5-GF-3,81
Line Connector: Phoenix-Contact FK-MCP 1,5/5-STF-3,81, spring-cage connection,
Phoenix-Contact order no.:1851261 (included in the scope of delivery)
For conductor connection and conductor cross section see page 36.

Pin Position:

(line connector with labelling)



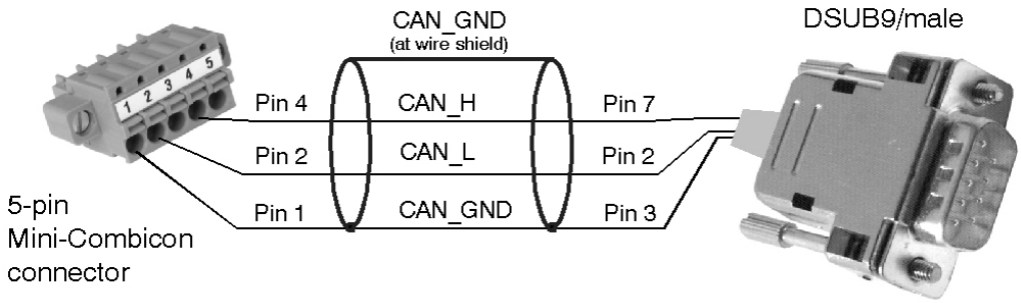
Pin-Assignment:

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

Signal description:

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

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

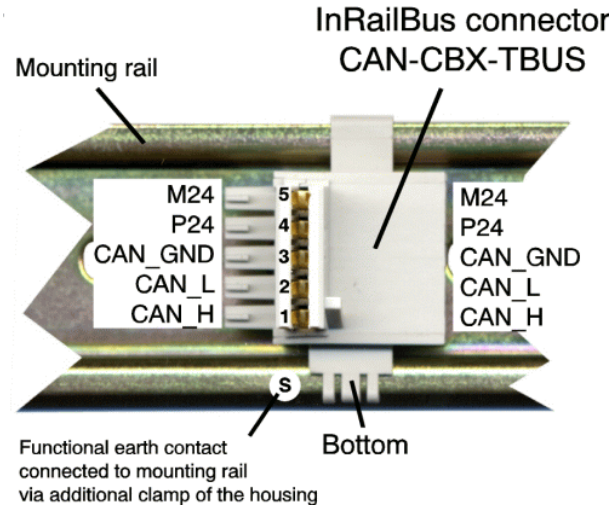


The 9-pin DSUB connector is assigned according to CiA 102

5.2.3 CAN and Power Supply Voltage via InRailBus Connector

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

Pin Position:



Pin Assignment:

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

S	FE (PE_GND)

Signal Description:

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

5.3 Serial Interfaces COM A and COM B

The CAN-CBX-COM2 module is equipped with two serial interfaces (COM A and COM B).
The base version CAN_CBX-COM2 2x RS-232 (esd-order No.: C.3055.02) comes with two RS-232 drivers.
The serial interface COM A is configured as RS-232 interface.
The serial interface COM B can be individually configured as RS-232, RS-422, RS-485 or TTY-interface via Piggybacks.

In the following the wiring of the serial interfaces is shown relating to the data direction. The figures explain the short terms of the signals used in the chapter “Connector Assignment”.

5.3.1 RS-232 Interface

In this example for the connector cable an adapter cable from the 7-pin PCB plug connector to the DSUB9 socket is shown. This cable is designed for the direct connection of the CAN-CBX-COM2 to a PC without a Null modem adapter.
For the individual connector assignment we recommend an adapter cable with freely configurable DSUB connectors and sockets. Please ask for cables configured according to your specific requirements, if required.

Attention !
Please note that the function of the control signals depends on the software drivers of the serial interface used.

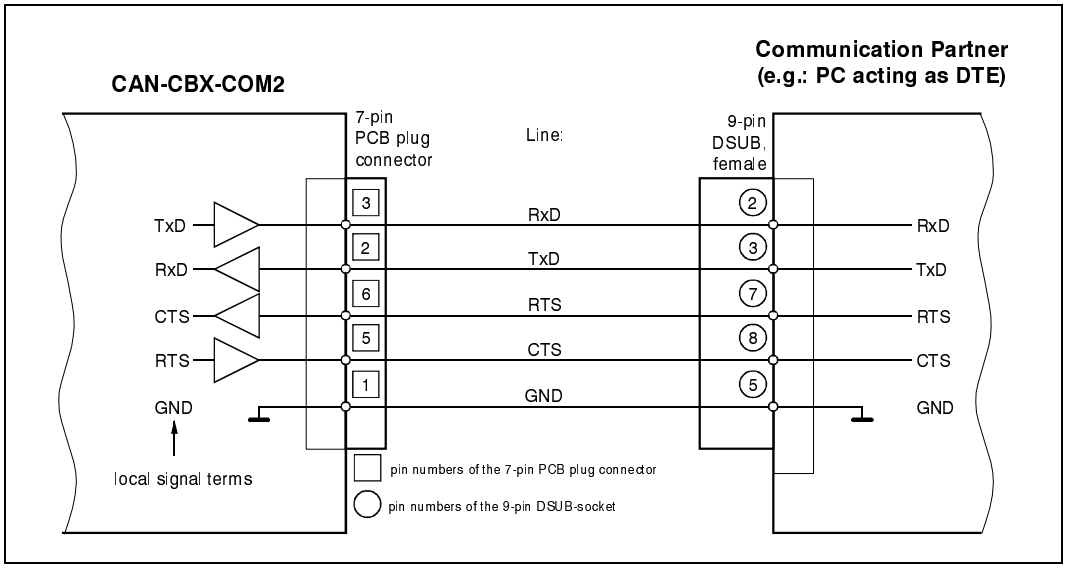


Fig. 12: Connection scheme for RS-232 operation (Example: CAN-CBX-COM2 <-> PC)

5.3.2 Option: RS-422 Interface

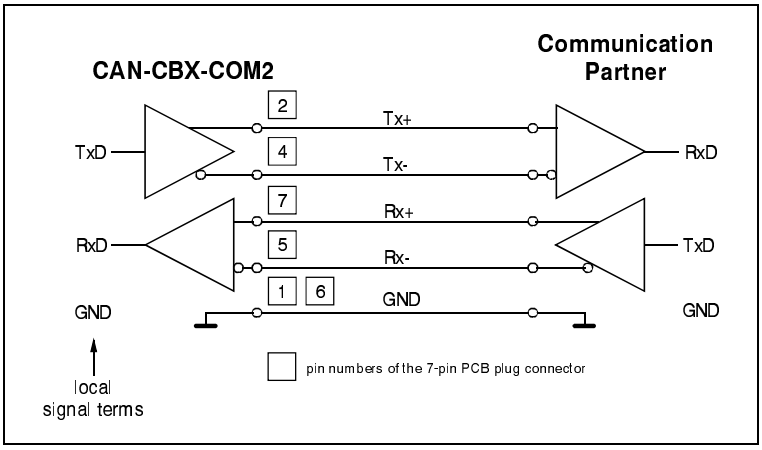


Fig. 13: Connection scheme for RS-422 operation, optional

5.3.3 Option: RS-485 Interface

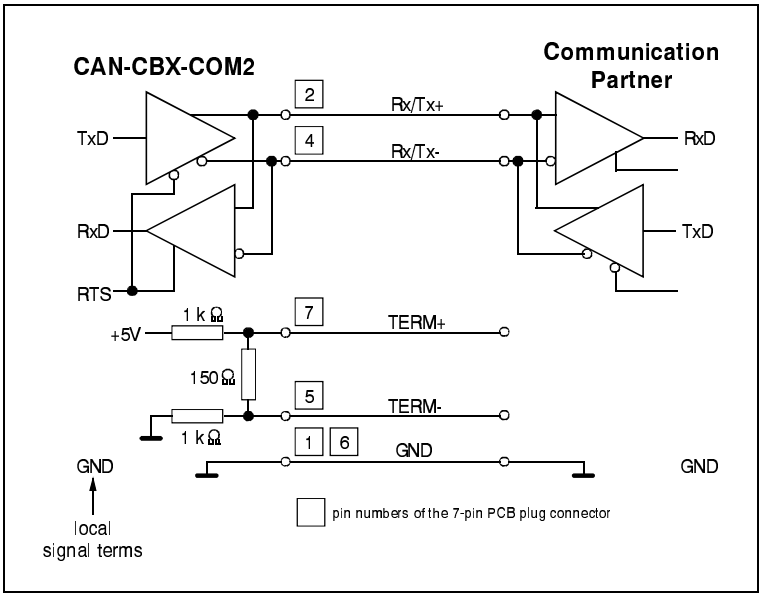


Fig. 14: Connection scheme for RS-485 operation, optional

In RS-485 operation the pins 5 and 7 of the PCB plug connector are connected to a termination resistor network on the piggyback. To activate the termination the signal Rx/Tx+ has to be connected to TERM+ and the signal Rx/Tx- to TERM-.

5.3.4 Option: TTY(20 mA) Interface

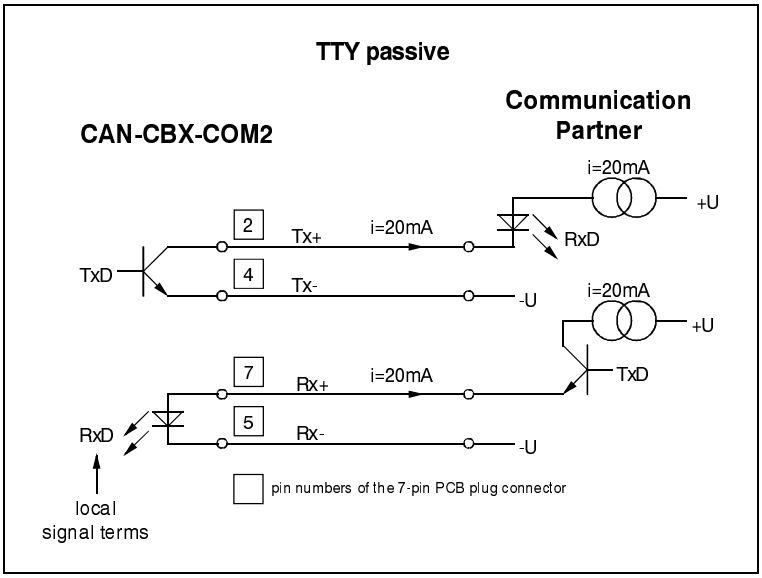


Fig. 15: Connection scheme for TTY operation (passive), optional

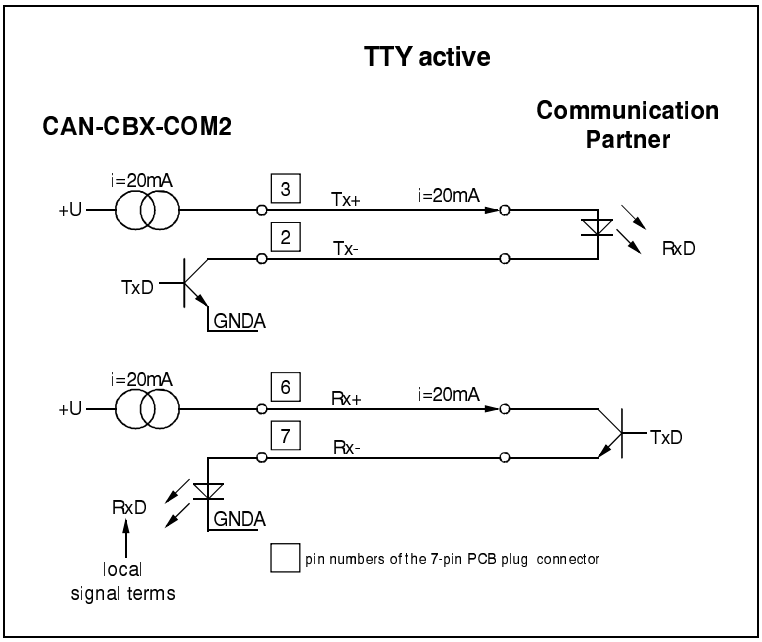


Fig. 16: Connection scheme for TTY operation (active), optional

5.3.5 Connector Assignment

Device connector: 2x Phoenix Contact MC 1,5/7-G-3,5
Line connector: 2x Phoenix Contact FK-MCP 1,5/7-3,5 (spring-cage connection)
Phoenix Contact Order No.: 1939960 (included in the scope of delivery)
For conductor connection and conductor cross section see page 36.

Pin Position:

Serial interface **COM A**

Serial Interface **COM B**

The serial interface COM A is configured as RS-232 interface

COM B can be individually configured as RS-232-, RS-422-, RS-485- or TTY-interface via Piggybacks

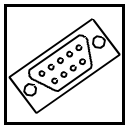
The base version
CAN-CBX-COM2 2x RS-232
(esd-order No.: C.3055.02)
comes with two RS-232 drivers.

Further options are available on request.

Pin Assignment of the basic version CAN-CBX-COM2 2xRS-232 (esd order no.: C.3055.02):

In this version both serial interfaces are configured as RS-232 interfaces.

Connector Pin	Signal COM A/COM B RS-232
1	GND
2	RxD
3	TxD
4	-
5	RTS
6	CTS
7	-



Connector Pin Assignment

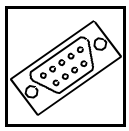
Pin Assignment of the version CAN-CBX-COM2 1xRS-232 1x RS-485 (esd order no.: C.3055.03):

In this version the serial interface COM A is configured as RS-232 interface and COM B as RS-485 interface.

Connector Pin	Signal	
	COM A RS-232	COM B RS-485
1	GND	GND
2	RxD	Tx/Rx+
3	TxD	-
4	-	Tx/Rx-
5	RTS	Term-
6	CTS	GND
7	-	Term+

Option:
On request the serial interface COM B can be optionally configured as RS-422, TTYp or TTYa interface via piggybacks.

Connector Pin	Signal COM B		
	RS-422	TTYp	TTYa
1	GND	GND	GND
2	Tx+	Tx+	Tx-
3	-	I1	Tx+
4	Tx-	Tx-	-12 V
5	Rx-	Rx-	-12 V
6	GND	I2	Rx+
7	Rx+	Rx+	Rx-



5.4 Assignment of the Labelling on the Module

Labelling on CAN-CBX-COM2	Name in schematic diagram*[3]
COM A	X300
COM B	X310
CAN	X400
24V	X100
InRailBus	X101

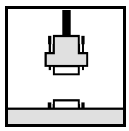
*[3] The schematic diagram is not part of this manual.

5.5 Conductor Connection/Conductor Cross Sections

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

Interface	Power Supply Voltage 24 V [4]	CAN-Connector [5]	Serial Interface [6]
Connector type plug component (Range of articles)	FKCT 2,5/..-ST KMGY	FK-MCP 1,5/..-STF-3,81	FK-MCP 1,5/.. ST-3,5
Connection method	spring-cage connection	spring-cage connection	spring-cage connection
Stripping length	10 mm	9 mm	9 mm
Conductor cross section solid min.	0.2 mm²	0.14 mm²	0.14 mm²
Conductor cross section solid max.	2.5 mm²	1.5 mm²	1.5 mm²
Conductor cross section stranded min.	0.2 mm²	0.14 mm²	0.14 mm²
Conductor cross section stranded max.	2.5 mm²	1.5 mm²	1.5 mm²
Conductor cross section stranded, with ferrule without plastic sleeve min.	0.25 mm²	0.25 mm²	0.25 mm²
Conductor cross section stranded, with ferrule without plastic sleeve max.	2.5 mm²	1.5 mm²	1.5 mm²
Conductor cross section stranded, with ferrule with plastic sleeve min.	0.25 mm²	0.25 mm²	0.25 mm²
Conductor cross section stranded, with ferrule with plastic sleeve max.	2.5 mm²	0.5 mm²	0.5 mm²
Conductor cross section AWG/kcmil min.	24	26	26
Conductor cross section AWG/kcmil max	12	16	16
2 conductors with same cross section, solid min.	n.a.	n.a.	n.a.
2 conductors with same cross section, solid max.	n.a.	n.a.	n.a.
2 conductors with same cross section, stranded min.	n.a.	n.a.	n.a.
2 conductors with same cross section, stranded max.	n.a.	n.a.	n.a.
2 conductors with same cross section, stranded, ferrules without plastic sleeve, min.	n.a.	n.a.	n.a.
2 conductors with same cross section, stranded, ferrules without plastic sleeve, max.	n.a.	n.a.	n.a.
2 conductors with same cross section, stranded, TWIN ferrules with plastic sleeve, min.	0.5 mm²	n.a.	n.a.
2 conductors with same cross section, stranded, TWIN ferrules with plastic sleeve, max.	1 mm²	n.a.	n.a.
Minimum AWG according to UL/CUL	26	28	28
Maximum AWG according to UL/CUL	12	16	16

n.a. ... not allowed



6. Correct Wiring of Electrically Isolated CAN Networks

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

6.1 Standards concerning CAN Wiring

The flexibility in CAN network design is one of the key strengths of the various extensions and additional standards like e.g. CANopen, ARINC825, DeviceNet and NMEA2000 that have been built on the original ISO 11898-2 CAN standard. In using this flexibility comes the responsibility of good network design and balancing these tradeoffs.

Many CAN organizations and standards have scaled the use of CAN for applications outside the original ISO 11898. They have made system level tradeoffs for data rate, cable length, and parasitic loading of the bus.

However for CAN network design margin must be given for signal loss across the complete system and cabling, parasitic loadings, network imbalances, ground offsets against earth potential and signal integrity. **Therefore the practical maximum number of nodes, bus length and stub length are typically much lower.**

esd has concentrated her recommendations concerning CAN wiring on the specifications of the ISO 11898-2. Thus this wiring hints forgoes to describe the special features of the derived standards CANopen, ARINC825, DeviceNet and NMEA2000.

The consistent compliance to ISO 11898-2 offers significant advantages:

- Durable operation due to well proven design specifications
- Minimizing potential failures due to sufficient margin to physical limits
- Trouble-free maintenance during future network modifications or during fault diagnostics due to lack of exceptions

Of course reliable networks can be designed according to the specifications of CANopen, ARINC825, DeviceNet and NMEA2000, **however it must be observed that it is strictly not recommended to mix the wiring guidelines of the various specifications!**

6.2 Light Industrial Environment (Single Twisted Pair Cable)

6.2.1 General Rules

Note:
esd grants the EU Conformity of the product, if the CAN wiring is carried out with at least single shielded single twisted pair cables that match the requirements of ISO 118982-2. Single shielded double twisted pair cable wiring as described in chapter 6.3 ensures the EU Conformity as well.

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

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

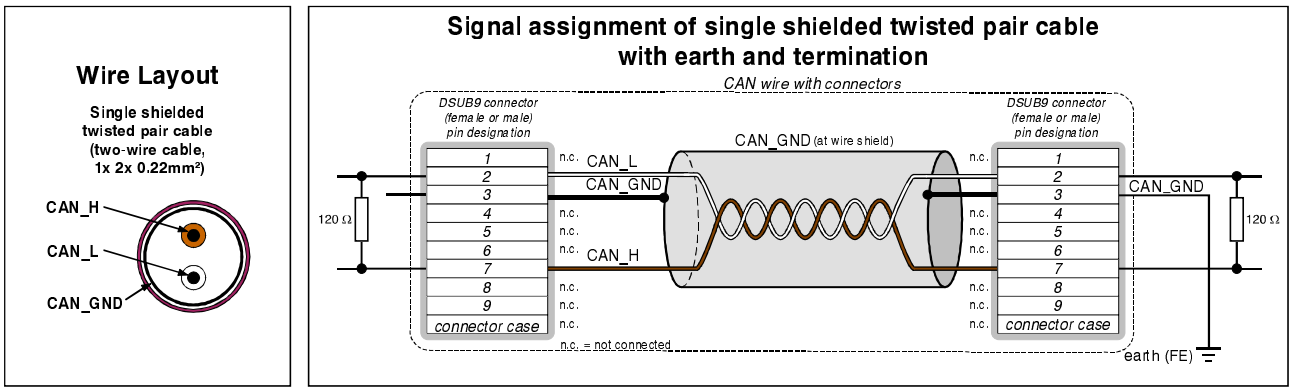


Figure. 17: CAN wiring for light industrial environment

6.2.2 Cabling

- To connect CAN devices with just one CAN connector per net use a short stub (< 0.3 m) and a T-connector (available as accessory). If this devices are located at the end of the CAN network, the CAN terminator “CAN-Termination-DSUB9” can be used.

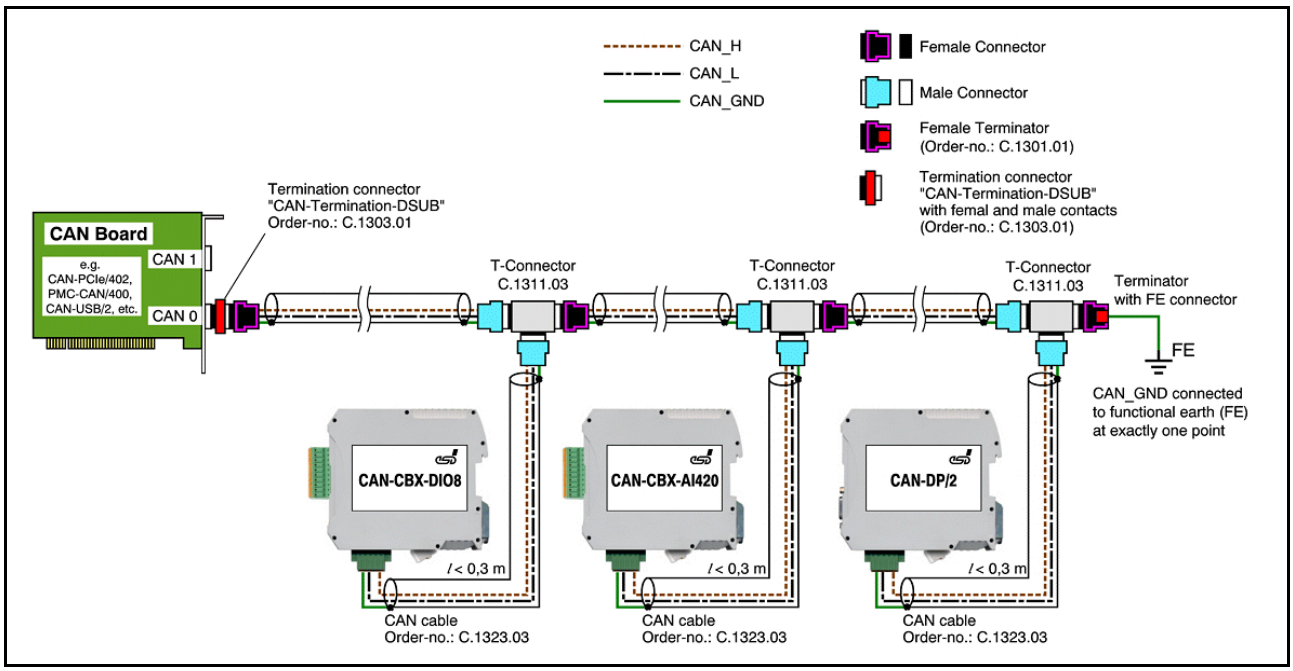


Figure. 18: Example for proper wiring with single shielded single twisted pair wires

6.2.3 Termination

- A termination resistor has to be connected at both ends of the CAN bus.
  If an integrated CAN termination resistor which is equipped at the CAN interface at the end of the bus is connected, this one has to be used for termination instead of an external CAN termination plug.

- 9-pin DSUB-termination connectors with integrated termination resistor and male and female contacts (gender changer) are available from esd (order no. C.1303.01).

- DSUB termination connectors with male contacts (order no. C.1302.01) or female contacts (order no. C.1301.01) and additional functional earth contact are available, if CAN termination and grounding of CAN_GND is required.

6.3 Heavy Industrial Environment (Double Twisted Pair Cable)

6.3.1 General Rules

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

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

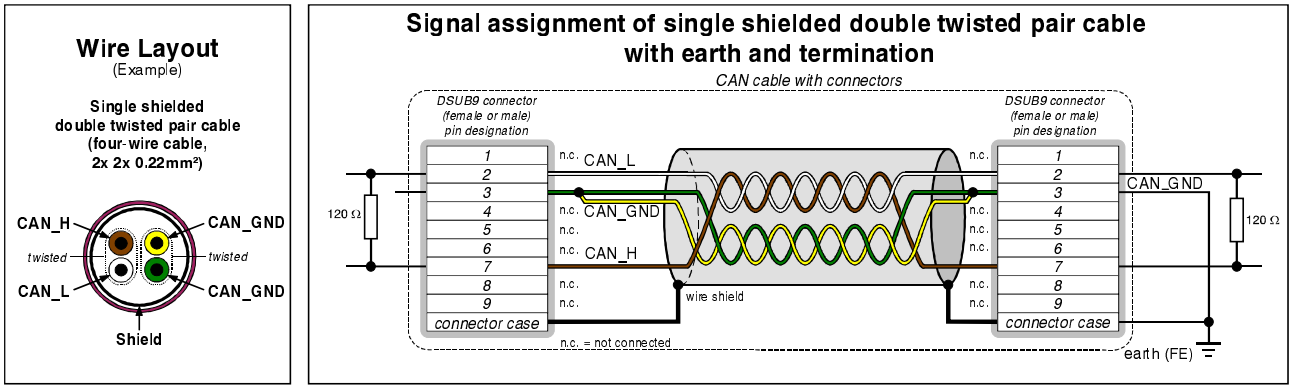


Fig. 19: CAN wiring for heavy industrial environment

6.3.2 Device Cabling

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

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

If a mixed application of single twisted and double twisted cables is unavoidable, take care that the CAN_GND line is not interrupted!

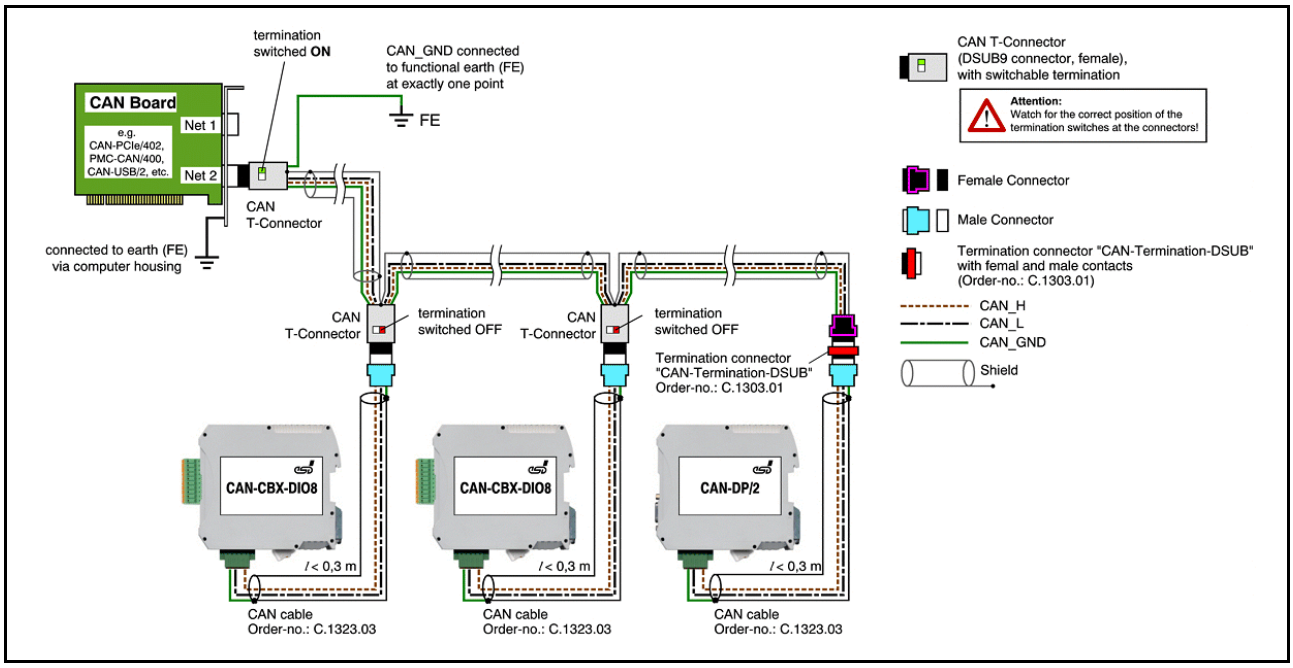


Fig. 20: Example for proper wiring with single shielded double twisted pair cables

6.3.3 Termination

- A termination resistor has to be connected at both ends of the CAN bus. If an integrated CAN termination resistor which is equipped at the CAN interface at the end of the bus is connected, this one has to be used for termination instead of an external CAN termination plug.
- 9-pin DSUB-termination connectors with integrated termination resistor and male and female contacts (gender changer) are available from esd (order no. C.1303.01).
- 9-pin DSUB-connectors with integrated switchable termination resistor can be ordered e.g. from ERNI (ERBIC CAN BUS MAX, female contacts, order no.:154039).

6.4 Electrical Grounding

- For CAN devices with electrical isolation the CAN_GND must be connected between the CAN devices.
- CAN_GND should be connected to the earth potential (FE) at **exactly one** point of the network.
- Each *CAN interface with electrical connection to earth potential* acts as an earthing point. For this reason it is recommended not to connect more than one *CAN device with electrical connection to earth potential*.
- Grounding can be made e.g. at a connector (e.g. order no. C.1302.01 or C.1301.01)

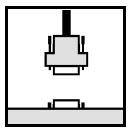
6.5 Bus Length

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
$666.\overline{6}$	80	-
500	130	100
$333.\overline{3}$	180	-
250	270	250
166	420	-
125	570	500
100	710	650
$83.\overline{3}$	850	-
$66.\overline{6}$	1000	-
50	1400	1000
$33.\overline{3}$	2000	-
20	3600	2500
12.5	5400	-
10	7300	5000

Table 13: Recommended cable lengths at typical bit rates (with esd-CAN interfaces)

- Optical couplers are delaying the CAN signals. esd modules typically reach a wire length of 37 m at 1 Mbit/s within a proper terminated CAN network without impedance disturbances like e.g. caused by cable stubs > 0.3 m.

Note:
Please note the recommendations of ISO 11898 regarding to the configuration of the conductor cross section in dependance of the cable length.



6.6 Examples for CAN Cables

esd recommends the following two-wire and four-wire cable types for CAN network design. These cable types are used by esd for ready-made CAN cables, too.

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

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

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

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

Note:
Ready-made CAN cables with standard or custom length can be ordered from **esd.**

7. CAN Troubleshooting Guide

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

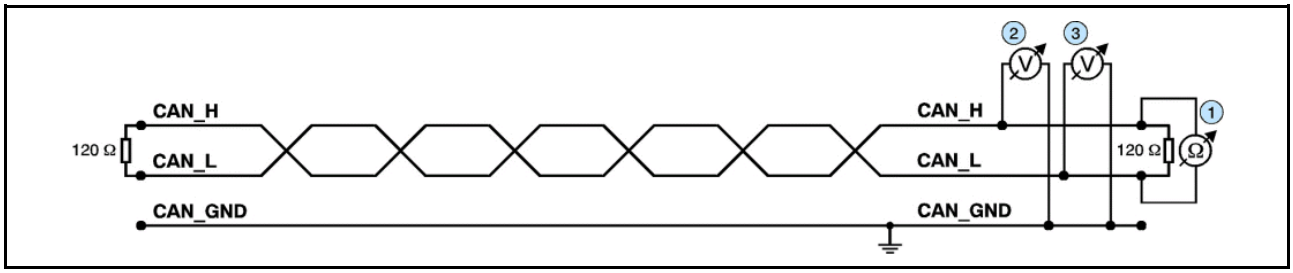


Figure. 21: Simplified diagram of a CAN network

7.1 Termination

The termination is used to match the 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 avoided. 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 one end of the network (1) (see figure above)

The measured value should be between 50 Ω and 70 Ω.

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

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

7.2 Electrical Grounding

CAN_GND of the CAN network should be connected to Functional earth potential (FE) at only **one** point. This test will check if the CAN_GND is grounded in several places.
To test it, please

1. Disconnect the CAN_GND from the earth potential (FE).

2. Measure the DC resistance between CAN_GND and earth potential (see figure on the right).

3. Reconnect CAN_GND to earth potential.

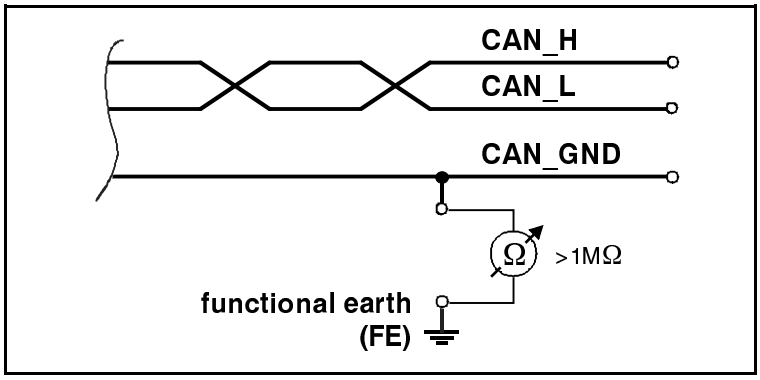


Fig. 22: Simplified schematic diagram of ground test measurement

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

7.3 Short Circuit in CAN Wiring

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

7.4 CAN_H/CAN_L Voltage

Each node contains a CAN transceiver that outputs differential signals. When the network communication is idle the CAN_H and CAN_L voltages are approximately 2.5 V measured to CAN_GND. 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 CAN_GND ② (see figure at previous page).
4. Measure the DC voltage between CAN_L and CAN_ GND ③ (see figure at previous page).

Normally the voltage should be between 2.0 V and 3.0 V.

If it is lower than 2.0 V or higher than 3.0 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.

To find the node with a faulty transceiver within a network please test the CAN transceiver resistance (see below) of the nodes.

7.5 CAN Transceiver Resistance Test

CAN transceivers have circuits that control CAN_H and CAN_L. Experience has shown that electrical damage of the circuits may increase the leakage current in these circuits.

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

1. Switch off the node and disconnect it from the network (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).

The measured resistance has to be about 500 kΩ for each signal. If it is much lower, the CAN transceiver is probably faulty. Another indication for a faulty transceiver is a very high deviation between the two measured input resistances (>> 200%).

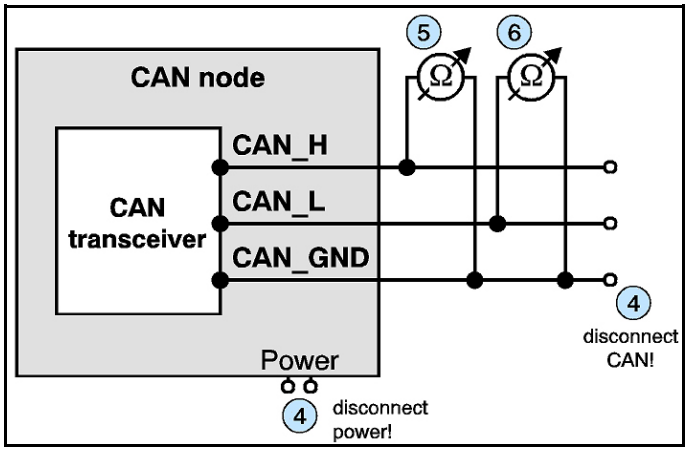


Figure 23: Simplified diagram of a CAN node

7.6 Support by esd

If you have executed the fault diagnostic steps of this troubleshooting guide and you even can not find a solution for your problem our support department will be able to assist.
Please contact our support via email at **support@esd.eu** or by phone **+40-511-37298-130**.

8. CANopen Firmware

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

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

8.1 Definition of Terms

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

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

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

8.2 NMT-Boot-up

The CAN-CBX module can be initialized with the 'Minimum Capability Device' boot-up as described in [1]. (Needs firmware version 1.02 or higher).

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

8.3 The CANopen-Object Directory

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

Part of the object directory are among others:

Index	Object	Example
0001_h ... $009F_h$	definition of data types	-
1000_h ... $1FFF_h$	Communication Profile Area	1001_h = Error rRegister
2000_h ... $5FFF_h$	Manufacturer Specific Profile Area	2800_h, 14_d = Tx-Cycle-Time
6000_h ... $9FFF_h$	Standardized Device Profile Area	not used for CAN-CBX-COM2
$A000_h$ ... $FFFF_h$	reserved	-

8.4 Communication Parameters of the PDOs

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

The **Node-ID** (module No.) is configured via coding switches Low and High. Please refer to chapter "Coding Switches" (page 17) for a detailed description of possible configurations.

8.4.1 Access on the Object Directory

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

This chapter does not describe all possible, but only some important modes of access in the CAN-CBX-COM2 module.
Definitions for the access modes can be taken from [1].

An SDO CAN Frame is structured as follows:

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

Example:

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

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

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

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

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

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

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

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

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

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

Error Codes of the SDO Domain Transfer

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

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

8.4.2 Non-volatile Storage of Parameters to EEPROM

After the transfer the parameters are immediately active.
The non-volatile storage of the parameters however is not carried out automatically. It must be initiated with a write access to object 1010_h and should only be carried out if the module is in the state *pre-operational*.

The storage mode is shown in the content of the object 1010_h:
Bit 1 of object 1010_h, sub-index 1 is not set, i.e the CAN-CBM-COM2 module does not save the configuration automatically. The storage must be initiated by writing the character string ‘save’ (73_h 61_h 76_h 65_h, order from CAN telegram) to object 1010_h, sub-index 1_d.

Read request for the current memory mode:

Identifier	Command code	Index (low)	Index (high)	Sub-index	Data 1	Data 2	Data 3	Data 4
600_h+ Node-ID	40_h (read request)	10_h	10_h	01_h	00_h	00_h	00_h	00_h
					Data are not evaluated			

Response of CAN-CBX-COM2 module (default setting):

Identifier	Command code	Index (low)	Index (high)	Sub-index	Data 1	Data 2	Data 3	Data 4
600_h+ Node-ID	$4F_h$ (read response, 1 byte)	10_h	10_h	01_h	01_h	00_h	00_h	00_h
					Storing of the parameter only at request			

Command for the storage of the parameters:

Identifier	Command code	Index (low)	Index (high)	Sub-index	Data 1	Data 2	Data 3	Data 4
600_h+ Node-ID	23_h (write, 4 bytes)	10_h	10_h	01_h	73_h ‘s’	61_h ‘a’	76_h ‘v’	65_h ‘e’
					= Storing of the parameters			

8.4.3 Restoring of Default Status of the parameters

The default status of the parameters is restored in two steps:

1. Call parameter ‘Restore Default-Parameter’: Index 1011_h, sub-index 1
2. Reset module via NMT-service ‘Reset’: Transmit the data 81xx (xx for Node-ID) on identifier ‘0’.

8.5 Overview of used CANopen-Identifiers

Function	Identifier	Description
Network management	0	NMT
SYNC	80_h	SYNC to all, (configurable via object 1005_h)
Emergency Message	80_h + *NodeID*	configurable via object 1014_h
RPDO1	200_h + *NodeID*	PDO1 to CAN-CBX-COM2 (Receive-PDO1 COM A)
TPDO1	180_h + *NodeID*	PDO1 to CAN-CBX-COM2 (object 1801_h) (Transmit-PDO1 COM A)
RPDO3	400_h + *NodeID*	PDO3 to CAN-CBX-COM2 (Receive-PDO3 COM B)
TPDO3	380_h + *NodeID*	PDO3 to CAN-CBX-COM2 (object 1802_h) (Transmit-PDO3 COM B)
Client-SDO	580_h + *NodeID*	SDO to CAN-CBX-COM2 (Tx)
Server-SDO	600_h + *NodeID*	SDO to CAN-CBX-COM2 (Rx)
Node Guarding	700_h + *NodeID*	not configurable

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

8.5.1 Setting the COB-ID

The COB-IDs which can be set (except the one of SYNC), are deduced initially from the setting of the Node-ID via the coding switches (see page 17). If the COB-IDs are set via SDO, this setting is valid even if the coding switches are set to another Node-ID after that.
To accept the Node-ID from the coding switches again, the *Comm defaults* or all defaults have to be restored (object 1011_h)

8.6 Default PDO-Assignment

The PDOs (Process Data Objects) are used to transmit the serial data. The following tables show the possible content of the TPDOs and RPDOs:

PDO	CAN Identifier	Length	Transmission Direction	Default
TPDO1	180_h + Node-ID	up to 8 bytes	from CAN-CBX-COM2 Transmit PDO)	COM A data serial -> CAN, byte 0...7
TPDO3	380_h + Node-ID	up to 8 bytes		COM B data serial -> CAN, byte 0...7

PDO	CAN Identifier	Length	Transmission Direction	Default
RPDO1	200_h + Node-ID	up to 8 bytes	to CAN-CBX-COM2 Receive PDO)	COM A data serial <- CAN, byte 0...7
RPDO3	400_h + Node-ID	up to 8 bytes		COM B data serial <- CAN, byte 0...7

8.7 Communication Profile Area

8.7.1 Used Names and Abbreviations

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

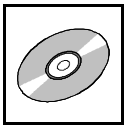
PDO-Mappable	PDO-Mapping is enabled for this Sub-index of the PDO
Save to EEPROM	the value of this parameter is stored in the local EEPROM, if the command 'save' is called (see page 52)
Data type	data type (e.g. unsigned 8, unsigned 32)
Access mode	allowed access modes to this parameter ro... read_only This parameter can only be read. Write accesses will cause an error message. const.... constant This parameter can not be set by the user. It is readable. Write accesses will cause an error message. rw... read&write This parameter can be read or written.
Value range	value range of the parameter
Default value	default setting of the parameter
Name/Description	name and short description of the parameter

8.8 Implemented CANopen-Objects

A detailed description of the objects can be taken from the standard CiA 301 [1].

8.8.1 Overview of used 1000-Objects and Default Values

Index	Name	Sub-Index (max.)	Data type	Access mode	Product-specific values
1000_h	Device Type	-	Unsigned 32	ro	default: $0080\ 0000_h$
1001_h	Error Register	-	Unsigned 8	ro	supported error bits: 0: generic 7: manufacturer
1002_h	Manufacturer Status	-	Unsigned 32	ro	Status
1003_h	Predefined Error	A_h	Unsigned 32	ro	-
1005_h	COB-ID SYNC-Message	-	Unsigned 32	rw	default: 80_h
1006_h	Communication Cycle Period	-	Unsigned 32	rw	default: 0
1008_h	Manufacturer Device Name	-	Visible String	ro	default:'CAN-CBX-COM2'
1009_h	Manufacturer Hardware Version	-	Visible String	ro	default depending on version
$100A_h$	Manufacturer Software Version	-	Visible String	ro	default depending on version
$100C_h$	Guard Time	-	Unsigned 16	rw	default: 0
$100D_h$	Life Time Factor	-	Unsigned 8	rw	default: 0
1010_h	Store Parameters	4	Unsigned 32	rw	Parameters which can be saved or loaded: 1005_h ...$1029_{h,}$ All parameters of the objects 2800_h, 2810_h, 2880_h, 2881_h, 2900_h, 2910_h, 2980_h, 2981_h
1011_h	Restore Default Parameters	4	Unsigned 32	rw	
1014_h	COB-ID Emergency	-	Unsigned 32	ro	default: 80_h + Node-ID
1015_h	Inhibit Time Emergency	-	Unsigned 16	rw	default:0
1016_h	Consumer Heartbeat Time	1	Unsigned 32	rw	default: 0
1017_h	Producer Heartbeat Time	0	Unsigned 16	rw	default: 0
1018_h	Identity Object	4	Unsigned 32	ro	Vendor Id: 00000017_h Prod. Code: 23055002_h (= C.3055.02)
1019_h	Synchronous Counter Overflow value	-	Unsigned 8	rw	default: 0
1020_h	Verify Configuration	2	Unsigned 32	ro	--
1029_h	Error Behaviour Object	4	Unsigned 8	rw	default: 0



Index	Sub-index (max.)	Description	Data type	Access mode
1400_h	6	Receive PDO Communication Parameter	PDOCommPar	rw
1402_h	6	Receive PDO Communication Parameter	PDOCommPar	rw
1600_h	8	Receive PDO Mapping Parameter	PDOMapping	rw
1602_h	8	Receive PDO Mapping Parameter	PDOMapping	rw
1800_h	6	Transmit PDO Communication Parameter	PDOCommPar	rw
1802_h	6	Transmit PDO Communication Parameter	PDOCommPar	rw
$1A00_h$	8	Transmit PDO Mapping Parameter	PDOMapping	rw
$1A02_h$	8	Transmit PDO Mapping Parameter	PDOMapping	rw

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

8.8.2 Device Type (1000_h)

INDEX	**1000_h**
Name	*device type*
Data type	unsigned 32
Access mode	ro
Default value	0080 0000_h

The value of the *device type* is: 0080.0000_h (Additional Information: 0080_h Device Profile Number: 0000_h)

Example: Reading the Device Type

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

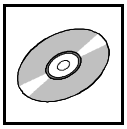
ID	RTR	LEN	DATA							
			1	2	3	4	5	6	7	8
603_h	0_h	8_h	40_h	00_h	10_h	00_h	00_h	00_h	00_h	00_h
			Read Request	Index=1000_h		Sub Index				

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

ID	RTR	LEN	DATA							
			1	2	3	4	5	6	7	8
583_h	0_h	8_h	43_h	00_h	10_h	00_h	**00_h**	**00_h**	**80_h**	**00_h**
			Read Response	Index=1000_h		Sub Index	Device Profile Nr. 0000_h		Additional information 0080_h	

value of device type: 0080 0000_h

The data field is always structured following the rule 'LSB first, MSB last'.



8.8.3 Error Register (1001_h)

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

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

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

Bit	Meaning	supported by CAN-CBX-COM2?
0	*generic*	✓
1	*curent*	-
2	*voltage*	-
3	*temperature*	-
4	*communication error* (overrun, error state)	-
5	*device profile*	-
6	reserved	-
7	*manufacturer-specific error*	✓

Bits which are not supported (-) are always returned as ‘0’.

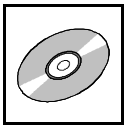
The following messages can occur:

00_h - no error
81_h - one of the error conditions defined by esd has occurred
(any manufacturer-specific error)

8.8.4 Manufacturer Status Register (1002_h)

INDEX	**1002_h**
Name	manufacturer status register
Data type	unsigned 32
Default value	No

The bits of the status register are manufacturer-specific and may only be used for internal purposes.



8.8.5 Pre-defined Error Field (1003_h)

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

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

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

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

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

Meaning of the variables:

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

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

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

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

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

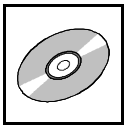
- 8110_h - CAN overrun error
  - Sample rate is set too high, thus the firmware is not able to transmit all data to the CAN bus.
- 8120_h - CAN in error passive mode
- 8130_h - Lifeguard error / heartbeat error
- 8140_h - Recovered from "Bus Off"
- 8240_h - Unexpected SYNC data length
- 6000_h - Software error:
  -EEPROM checksum error (no transmission of this error message as emergency message)
- 6110_h - Internal Software error
  e.g.:
  - saved data had invalid checksum and default data is loaded
- $FF10_h$ - Data loss (A/D data overflow)
- 5000_h - Hardware error (e.g. A/D-converter defective)

Emergency Message

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

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

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



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

INDEX	**1005_h**
Name	*COB-ID SYNC message*
Data type	unsigned 32
Access mode	rw
Default value	80_h
Low limit	01_h
High limit	C000 $07FF_h$

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 $0...7FF_h$.

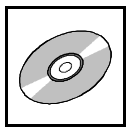
8.8.7 Communication Cycle Period (1006_h)

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

This object contains the cycle time of the SYNC frames.

Index	Sub-index	Description	Value range	Default value	Data type	Access mode
1006_h	0	*communication cycle period*	0, 0000 01F4$_h$ ... FFFF FFFF$_h$	0 µs	unsigned 32	rw

communication cycle period Cycle time of the SYNC frame in [µs].
If the value is '0' no SYNC messages are transmitted.



8.8.8 Manufacturer Device Name (1008_h)

INDEX	**1008_h**
Name	*manufacturer device name*
Data type	visible string
Default value	see chapter 8.8.1 (page 56)

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

8.8.9 Manufacturer Hardware Version (1009_h)

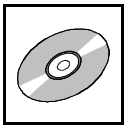
INDEX	**1009_h**
Name	*manufacturer hardware version*
Data type	visible string
Default value	string: e.g. ‘1.00’ (depending on version)

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

8.8.10 Manufacturer Software Version ($100A_h$)

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

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



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

The CAN-CBX module supports the node guarding or alternatively the heartbeat function (see page 76).

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

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

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

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

8.8.12 Store Parameters (1010_h)

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

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

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

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

Assignment of the variables

save all parameters

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

save_communication_parameter

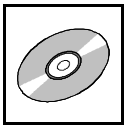
saves all communication parameters of those objects (objects 1000_h ... $1FFF_h$, if available), which have a read/write (rw) right of access (here e.g. 1005_h ... 1029_h).

save_application_parameter

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

save_manufacturer_parameter

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



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

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

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

Bit	Value	Description
auto	0	CAN-CBX module does **not** save the parameters autonomously
	1	CAN-CBX module **saves** the parameters autonomously
cmd	0	CAN-CBX module does **not** save the parameters on command
	1	CAN-CBX module **saves** the parameters on command

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

8.8.13 Restore Default Parameters (1011_h)

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

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

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

Assignment of the variables

restore all parameters

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

restore_communication_parameter

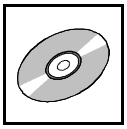
restores all communication default parameters of those objects (objects 1000_h ... $1FFF_h$, if available, here e.g. 1005_h ... 1029_h).

restore_application_parameter

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

restore_manufacturer_parameter

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



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

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

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

Bit	Value	Description
cmd	0	The default parameters of the CAN-CBX-module can **not** be loaded
	1	The default parameters of the CAN-CBX-module can **be loaded**

8.8.14 COB_ID Emergency Message (1014_h)

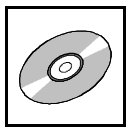
INDEX	**1014_h**
Name	*COB-ID emergency object*
Data type	unsigned 32
Default value	80_h + Node-ID

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

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

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

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



8.8.15 Inhibit Time EMCY (1015_h)

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

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

8.8.16 Consumer Heartbeat Time (1016_h)

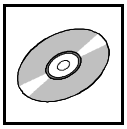
INDEX	**1016_h**
Name	*consumer heartbeat time*
Data type	unsigned 32
Default value	No

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

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

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

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



Meaning of the variable *consumer-heartbeat_time_x*:

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

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

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

Example:

consumer-heartbeat_time = 0031 $03E8_h$

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

8.8.17 Producer Heartbeat Time (1017_h)

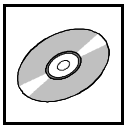
INDEX	**1017_h**
Name	*producer heartbeat time*
Data type	unsigned 16
Default value	0 ms

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

If the value of the producer heartbeat time is higher than '0', it is active and stops the node-/ life-guarding (see page 67).
If the value of the producer-heartbeat-time is set to '0', transmitting heartbeats by this module is stopped.

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

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



8.8.18 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	Sub-index	Description	Value range	Default value	Data type	Access mode
1018_h	0	*no_of_entries*	04	04	unsigned 8	ro
	1	*vendor_id*	0...FFFF FFFF$_h$	0000 0017$_h$	unsigned 32	ro
	2	*product_code*	0...FFFF FFFF$_h$	2305 5002$_h$	unsigned 32	ro
	3	*revision_number*	0...FFFF FFFF$_h$	*SW Rev.*	unsigned 32	ro
	4	*serial_number*	0...FFFF FFFF$_h$	*HW serial number*	unsigned 32	ro

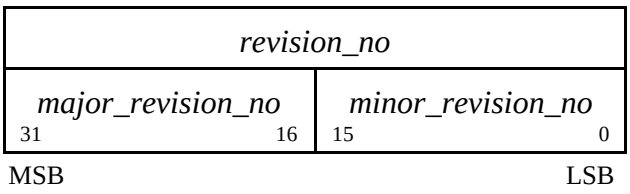
Description of the variables:

vendor_id This variable contains the esd-vendor-ID. This is always 0000 0017$_h$.

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

Example: '2**305 5002**$_h$' corresponds to article number 'C.3055.02'.

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

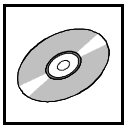


serial_number Here the serial number of the hardware is read. The first two characters of the serial number are letters which designate the manufacturing lot. The following characters represent the actual serial number.

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

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

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



8.8.19 Synchronous Counter Overflow Value (1019_h)

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

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

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

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

8.8.20 Verify Configuration (1020_h)

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

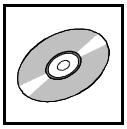
In this object the date and the time of the last configuration can be stored to check later whether the configuration complies with the expected configuration or not.
The content of the parameters is not evaluated by the firmware.

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

Parameter Description:

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

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



8.8.21 Error Behaviour Object (1029_h)

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

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

Index	Sub-index	Description	Value range	Default value	Data type	Access mode
1029_h	0	*no_of_error_classes*	4	4	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	rw
	4	*manufacturer_error*	0...2	0	unsigned 8	rw

Meaning of the variables:

Variable	Meaning
no_of_error_classes	number of error-classes (here always ‘4’)
communication_error	heartbeat/lifeguard error and *Bus off*
output_error	not used here
input_error	not used here
manufacturer_error	not used here

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

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

8.8.22 NMT Startup ($1F80_h$)

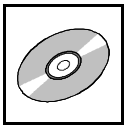
INDEX	**$1F80_h$**
Name	*NMT startup*
Data type	unsigned 32
Default value	2

This object is implemented from firmware version 1.02 on

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

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

Value	Meaning
$0000\ 0002_h$	Auto startup disabled (default)
$0000\ 0008_h$	Auto startup enabled
all other values	reserved



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

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

This object is implemented from firmware version 1.02 on

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

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

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

8.8.24 Receive PDO Communication Parameter 1400_h, 1402_h

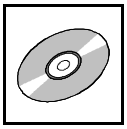
Objects 'Receive PDO Communication Parameters1400_h, 1402_h' define the features of a receive PDO (Rx-PDO). The CAN-CBM-COM2 module uses maximum **two** Rx-PDOs.

The number of Rx-PDO is fixed. Always the maximum number of Rx-PDOs is mapped. The number of bytes which shall be evaluated are defined by the user via the number of send bytes.

INDEX	**1400h**	**1402_h**
Name	receive PDO1 communication parameter	receive PDO3 communication parameter
Data type	PDOCommPar	PDOCommPar

Index	Sub-Index	PDO-mappable	Save to EEPROM	Access mode	Data type	Default value	Description
1400_h	0	no	no	ro	unsigned 8	6	number of entries
	1	no	yes	rw	unsigned 32	200_h + Node-ID	COB-ID used by PDO1
	2	no	no	ro	unsigned 8	FF_h	transmission type
	3	no	no	ro	unsigned 16	0	inhibit time
	4	no	no	rw	usigned 8	0	compatibility entry
	5	no	no	rw	unsigned 16	0	event-timer
	6	no	no	rw	unsigned 8	0	SYNC start value
1402_h	0	no	no	ro	unsigned 8	6	number of entries
	1	no	yes	rw	unsigned 32	400_h + Node-ID	COB-ID used by PDO3
	2	no	no	ro	unsigned 8	FF_h	transmission type
	3	no	no	ro	unsigned 16	0	inhibit time
	4	no	no	rw	unsigned 8	0	compatibility entry
	5	no	no	rw	unsigned 16	0	event-timer
	6	no	no	rw	unsigned 8	0	SYNC start value

Value range see [1].



Example: Definition of a new receive PDO with index '1400_h' at CAN identifier '$035A_h$'

The hexadecimal byte sequence has to be mapped on ID = 600 + Node-ID.

Byte	Name	Value	Content
0	Command	23_h	Write Request
1	Index Low	00_h	New PDO 1400_h in Intel-Notation
2	Index High	14_h	
3	Sub-index	1	defines COB-ID
4	Data 1	$5A_h$	COB-ID $035A_h$ in Intel-Notation
5	Data 2	03_h	
6	Data 3	0	
7	Data 4	0	

8.8.25 Receive PDO-Mapping Parameter 1600_h, 1603_h

Objects 'Receive PDO-Mapping Parameter 1600_h, 1603_h' change the assignment of the receive data to Rx-PDOs.

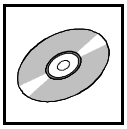
INDEX	**1600_h**	**1602_h**
Name	receive PDO1 mapping	receive PDO3 mapping
Data type	PDO Mapping	PDO Mapping

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

Index	Sub-Index	PDO-mappable	Save to EEPROM	Access mode	Data type	Default value	Description
1600_h	0	no	no	ro	unsigned 8	8	number of entries
	1	no	no	ro	CANopen Map Struct	$2880\ 00\ 08_h$	Data CAN -> serial, byte 1...8
	2	no	no	ro		$2880\ 00\ 08_h$	Data CAN -> serial, byte 1...8
	3	no	no	ro		$2880\ 00\ 08_h$	Data CAN -> serial, byte 1...8
	4	no	no	ro		$2880\ 00\ 08_h$	Data CAN -> serial, byte 1...8
	5	no	no	ro		$2880\ 00\ 08_h$	Data CAN -> serial, byte 1...8
	6	no	no	ro		$2880\ 00\ 08_h$	Data CAN -> serial, byte 1...8
	7	no	no	ro		$2880\ 00\ 08_h$	Data CAN -> serial, byte 1...8
	8	no	no	ro		$2880\ 00\ 08_h$	Data CAN -> serial, byte 1...8
1602_h	0	no	no	ro	unsigned 8	8	number of entries
	1	no	no	ro	CANopen Map Struct	$2980\ 00\ 08_h$	Data CAN -> serial, byte 1...8
	2	no	no	ro		$2980\ 00\ 08_h$	Data CAN -> serial, byte 1...8
	3	no	no	ro		$2980\ 00\ 08_h$	Data CAN -> serial, byte 1...8
	4	no	no	ro		$2980\ 00\ 08_h$	Data CAN -> serial, byte 1...8
	5	no	no	ro		$2980\ 00\ 08_h$	Data CAN -> serial, byte 1...8
	6	no	no	ro		$2980\ 00\ 08_h$	Data CAN -> serial, byte 1...8
	7	no	no	ro		$2980\ 00\ 08_h$	Data CAN -> serial, byte 1...8
	8	no	no	ro		$2980\ 00\ 08_h$	Data CAN -> serial, byte 1...8

Value range see [1].

Note:
The $288x_h$- and $298x_h$-objects registered in the PDO-Mapping are not accessible via SDOs. They are used as dummies to correspond to the structure of the objects 1600_h, 1602_h, $1A00_h$ and $1A02_h$.



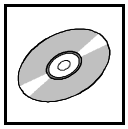
8.8.26 Object Transmit PDO Communication Parameter 1800_h, 1802_h

This objects define the parameters of the transmit-PDOs.

INDEX	**1800_h**	**1802_h**
Name	transmit PDO1 communication parameter	transmit PDO3 communication parameter
Data type	PDOCommPar	PDOCommPar

Index	Sub-Index	PDO-mappable	Save to EEPROM	Access Mode	Data type	Default value	Description
1800_h	0	no	no	ro	unsigned 8	6	number of entries
	1	no	yes	rw	unsigned 32	180_h + Node-ID	COB-ID used by PDO1
	2	no	yes	rw	unsigned 8	FF_h	transmission type
	3	no	no	rw	unsigned 16	0	inhibit time
	4	no	no	rw	unsigned 8	0	reserved
	5	no	yes	rw	unsigned 16	0	event timer
	6	no	no	rw	unsigned 8	0	SYNC start value
1802_h	0	no	no	ro	unsigned 8	6	number of entries
	1	no	yes	rw	unsigned 32	380_h + Node-ID	COB-ID used by PDO3
	2	no	yes	rw	unsigned 8	FF_h	transmission type
	3	no	no	rw	unsigned 16	0	inhibit time
	4	no	no	rw	unsigned 8	0	reserved
	5	no	yes	rw	unsigned 16	0	event timer
	6	no	no	rw	unsigned 8	0	SYNC start value

value range see [1]

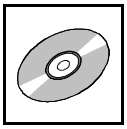


Implemented CANopen Objects

Example: Definition of a new Transmit PDOs with index '1800_h' with CAN identifier' $047F_h$'

The hexadecimal byte sequence has to be mapped on ID = 600_h + Node-ID.

Byte	Name	Value	Content
0	Command	23_h	Write Request
1	Index Low	00_h	new PDO 1800_h in Intel-Notation
2	Index High	18_h	
3	Sub-index	1	define COB-ID
4	Data 1	$7F_h$	COB $047F_h$ in Intel-Notation
5	Data 2	04_h	
6	Data 3	0	
7	Data 4	0	



8.8.27 Transmit PDO Mapping Parameter $1A00_h$, $1A02_h$

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

INDEX	**$1A00_h$**	**$1A02_h$**
Name	transmit PDO1 mapping	transmit PDO3 mapping
Data type	PDO Mapping	PDO Mapping

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

Index	Sub-Index	PDO-mappable	Save to EEPROM	Access mode	Data type	Default value	Description
$1A00_h$	0	no	no	ro	unsigned 8	8	number of entries
	1	no	no	ro	CANopen Map Struct	$2881\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
	2	no	no	ro		$2881\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
	3	no	no	ro		$2881\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
	4	no	no	ro		$2881\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
	5	no	no	ro		$2881\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
	6	no	no	ro		$2881\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
	7	no	no	ro		$2881\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
	8	no	no	ro		$2881\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
$1A02_h$	0	no	no	ro	unsigned 8	8	number of entries
	1	no	no	ro	CANopen Map Struct	$2981\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
	2	no	no	ro		$2981\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
	3	no	no	ro		$2981\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
	4	no	no	ro		$2981\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
	5	no	no	ro		$2981\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
	6	no	no	ro		$2981\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
	7	no	no	ro		$2981\ 00\ 08_h$	Data serial -> CAN, Byte 0...7
	8	no	no	ro		$2981\ 00\ 08_h$	Data serial -> CAN, Byte 0...7

Value range see [1].

Note:
The $288x_h$- and $298x_h$-objects registered in the PDO-Mapping are not accessible via SDOs. They are used as dummies to correspond to the structure of the objects 1600_h, 1602_h, $1A00_h$ and $1A02_h$.

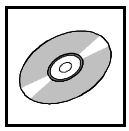
8.9 Transmission Modes

8.9.1 Supported transmission modes according to CiA 301, Table 55

Transmission Type	PDO transmission					Supported by CAN-CBX-COM2
	cyclic	acyclic	synchronous	asynchro-nous	RTR	
0	-	X	X	-	-	NO
1...240	X	-	X	-	-	NO
241...251	reserved					NO
252	-	-	X	-	X	NO
253	-	-	-	X	X	NO
254	-	-	-	X	X	NO
255	-	-	-	X	X	**YES**

Table 14: Transmission modes

Das CAN-CBX-COM2 module only supports transmission type 255. It transmits Tx frames depending on the parameters *MinChar* and *MaxChar*, *TxByRxTimeout* and after receiving RTRs.



8.10 Data Transfer and PDO Assignment

8.10.1 Function Description of local Firmware

Serial data is buffered between CAN and serial interfaces in both directions via a 256 byte sized ring buffer. The data which is received first is transmitted again first (FIFO).

If the ring buffer is full and further data is received, this data will be lost. Therefore, the transmission rates of CAN and the serial interfaces have to be synchronized.

8.10.1.1 Data Transfer CAN -> Serial Interface

The number of CAN data which is to be stored in the ring buffer within an interval has to be smaller than the number of data transmitted via the serial interface, because otherwise the ring buffer might overflow.

The number of data received per interval depends on the number of data bytes transmitted, the frequency of transmissions, the bit rate of the CAN, and the degree of CAN bus utilisation.

If data is lost during operation, breaks between transmissions have to be extended and/or the number of bytes transmitted has to be reduced.

The data is transmitted from CAN via identifier RPDO1 to the module. Up to 8 bytes ca be selected to be transmitted.

The data transfer from CAN to serial interface has the following chronological course:

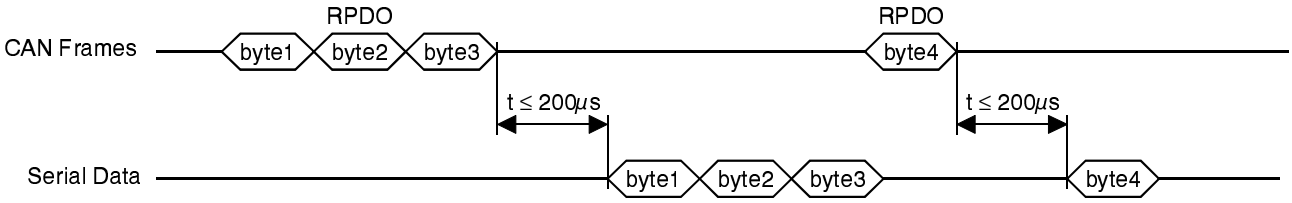


Fig. 24: Data transfer CAN -> serial interface (parameter *TxMinCharStart* = 0)

In addition, the CAN-CBM-COM2 module offers an RTR handshake at which the module informs the CAN partner via RTR, that the local ring buffer still can receive data. No data are sent any more if the ring buffer is full (see page 106).

The eight data bytes of the RPDOs are shown below:

RPDO	Data Bytes							
	0	1	2	3	4	5	6	7
RPDO1	1 ... 8 bytes data: CAN -> serial							

Table 15: Receiving data to be transmitted via Rx-identifier

8.10.1.2 Data Transfer Serial Interface-> CAN

The number of data of the serial interface to be stored in the ring buffer per interval has to be lower than the number of data to be transmitted via the CAN.

The CAN bus limits the data flow via bus capacity (priority), CAN-bit rate, the frequency of transmissions and the number of bytes transmitted.

The module offers possibilities to influence the last two factors via the parameters *Inhibit-Time*, *MaxChar* and *MinChar* (see objects 2800_h and 2900_h).

By *Inhibit-Time* the delay between two transmissions on the CAN bus is determined.

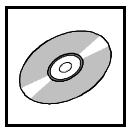
MinChar and *MaxChar* define the minimum and maximum number of data bytes which are to be transmitted on the CAN bus within a CAN frame (see page 97).

The data is transmitted by the module via TPDO1 on the CAN bus.

The eight data bytes of the TPDOs are:

TPDO	Data Bytes							
	0	1	2	3	4	5	6	7
TPDO1	1 ... 8 bytes data: serial -> CAN							

Table 16: Transmission of data received by serial interface via Tx-identifier



8.11 Device Profile Area

8.11.1 Overview of Implemented Objects 6000_h...$9FFF_h$

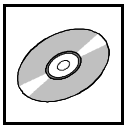
The module does not support objects in the range 6000_h...$9FFF_h$

8.12 Manufacturer Specific Profile Area

8.12.1 Overview of Implemented Objects 2800_h - 2981_h

Index	Object	Content of Sub-index 0	Meaning
2800_h	Serial communication parameter interface COM A	10_h	-
2810_h	Cyclic_String COM A	$0...F_h$	cyclically transmittable character string
2880_h	Dummy Rx object COM A	-	place holder for serial data COM A
2881_h	Dummy Tx object COM A	-	place holder for serial data COM A
2900_h	Serial communication parameter interface COM B	10_h	-
2910_h	Cyclic_String COM B	$0...F_h$	cyclically transmittable character string
2980_h	Dummy Rx object COM B	-	place holder for serial data COM B
2981_h	Dummy Tx object COM B	-	place holder for serial data COM B

Table 17: Overview of the manufacturer-specific parameters



8.12.2 Serial Communication Parameter 2800_h, 2900_h

The serial communication parameters of object 2800_h and object 2900_h are described together in this chapter, because they are identically structured. The communication parameters of object 2800_h are used for interface COM A and the communication parameters of object 2900_h are used for interface COM B.

8.12.2.1 Overview

COM A: 2800_h
COM B: 2900_h

Index	Sub-index	Description	Value range	Default Value	Data type	Access Mode	Save to EEPROM	see Page
2800_h/ 2900_h	0	*Number of entries*	10_h	10_h	unsigned 8	ro	no	-
	1	*Inhibit-Time*	00...FF_h	14_h	unsigned 8	rw	yes	96
	2	*MaxChar_MinChar*	00...FF_h	11_h	unsigned 8	rw	yes	97
	3	*Serial_Baudrate*	00...FF_h	22_h (9600 Baud)	unsigned 8	rw	yes	99
	4	*Serial_Mode*	00...FF_h	73_h	unsigned 8	rw	yes	101
	5	*TxMinCharStart*	00...FF_h	00_h	unsigned 8	rw	yes	105
	6	*Protocol*	00...63_h	00_h	unsigned 8	rw	yes	106
	7	*Handshake_Off*	00...FF_h	$F6_h$	unsigned 8	rw	yes	108
	8	*Handshake_On*	10...FF_h	$0A_h$	unsigned 8	rw	yes	108
	9	*TxByRxTimeout*	00...FF_h	00_h	unsigned 8	rw	yes	110
	A_h	*End_Char_1*	00...FF_h	$0D_h$	unsigned 8	rw	yes	111
	B_h	*End_Char_2*	00...FF_h	$0A_h$	unsigned 8	rw	yes	111
	C_h	*Special_Baudrates*	0001... $07FF_h$	0034_h	unsigned 16	rw	yes	112
	D_h	*RTR_T_Cycle_Time*	0000... $FFFF_h$	1388_h	unsigned 16	rw	yes	113
	E_h	*Tx_Cycle_Time*	0000... $FFFF_h$	0000_h	unsigned 16	rw	yes	114
	F_h	*Break_Code*	0000... $FFFF_h$	0000_h	unsigned 16	rw	yes	115
	10_h	*Break_Time_Factor*	00...FF_h	$0B_h$	unsigned 8	rw	yes	116

Table 18: Overview of the parameters of object 2800_h and object 2900_h

8.12.2.2 *Inhibit-Time*

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

Parameter name	***Inhibit-Time***
Object index, Sub-index	2800_h, 01_h (COM A)/ 2900_h, 01_h (COM B)
Access mode	rw
Data type	unsigned 8
Value range	$00...FF_h$
Unit	ms
Default value	14_h = 20 ms

Description: The Parameter *Inhibit-Time* defines the waiting period between two transmission cycles of serial data on the CAN bus.

The parameter *Inhibit-Time* defines how long the CAN controller waits after the last successful transmission of CAN data, before another transmission is triggered. The value is specified in [ms].

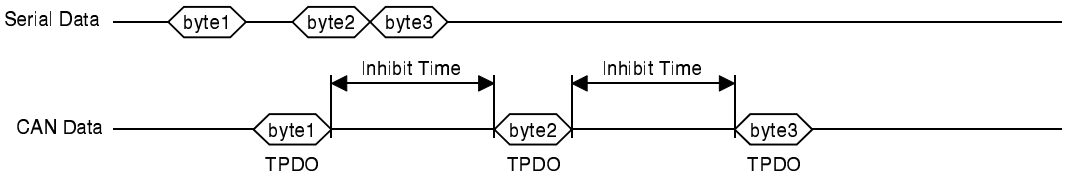
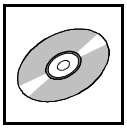


Fig. 25: Function of the parameter *Inhibit-Time* (parameter *MaxChar* = 1)



8.12.2.3 *MaxChar_MinCar*

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

Parameter name	***MaxChar_MinChar***
Object index, Sub-index	2800_h, 02_h (COM A)/ 2900_h, 02_h (COM B)
Access Mode	rw
Data Type	unsigned 8
Value range	$00...FF_h$
Unit	n.a.
Default value	11_h => every byte is transmitted

Description: The nibbles of this parameter contain the number of bytes, from which a transmission to CAN shall be started and the maximum number of data bytes in a CAN-frame. The higher nibble (*MaxChar*) contains the maximum number and the lower nibble (*MinChar*) the minimum number.
The default is 11_h, i.e. every received byte is transmitted separately in a frame.

	MaxChar_MinChar	
Bit:	7 4	3 0
Parameter:	*MaxChar*	*MinChar*

Table 19: Structure of parameter *MaxChar_MinChar*

Transmission command:
MinChar is only evaluated as described above, if values between 1 and 8 are specified.

If values between 9_h and F_h are specified, the functionality of the parameter changes: In this case the local software evaluates the entry of characters which have been specified via user parameter *End_Char_1* (2800_h, A_h / 2900_h, A_h) oder *End_Char_2* (2800_h, B_h / 2900_h, B_h) as transmission command.

In default setting of user parameter *End_Character* these are the flags 'Carriage Return' ($0D_h$) or 'Line Feed' ($0A_h$). As soon as one or both flags are detected, the data which have been received by the serial interface is transmitted to the CAN. If 8 bytes have been received before the flags have been received, the transmission starts automatically.
You can also specify for the transmission, whether the flags are to be transmitted on the CAN together with the data or not.

The following table shows the various options:

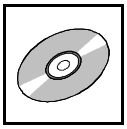
MinChar	Transmission command	Flag transmission to CAN	Comments
1...8	-	-	At least 1 to 8 bytes are always transmitted. Parameter *MaxChar* is also evaluated.
9	<*End_Char_1*>	with <*Char_1*>	Data is transmitted after <*Char_1*> has been received. <*Char_1*> is also transmitted.
A_h	<*End_Char_2*>	with <*Char_2*>	As under '9', but with <*Char_2*>.
B_h	<*End_Char_1*>	without <*Char_1*>	As under '9', but with <*Char_1*> is not transmitted as well.
C_h	<*End_Char_2*>	without <*Char_2*>	As under 'B', but with <*Char_2*>.
D_h	<*End_Char_1*>	without <*Char_1*> and without <*Char_2*>	The end of the message can have <*Char_1*> and <*Char_2*>. The data is transmitted after <*Char_1*> has been received. Neither <*Char_1*> nor <*Char_2*> are transmitted as well.
E_h	<*End_Char_2*>	without <*Char_1*> and without <*Char_2*>	As under 'D', but with <*Char_2*>.
F_h	-	-	reserved

Table 20: Selection of the transmission condition via parameter *MinChar*

Examples: The examples below are representative for the CAN-CBM-COM2 being operated by parameters in default setting, i.e. *Char_1* = <CR>, *Char_2* = <LF>.

1. For *MinChar* value B_h has been selected. Via the serial interface the data 'abcd<CR>' is received. The data 'abcd' would be transmitted on CAN after <CR> had been received.

2. For *MinChar* value E_h has been selected. Via the serial interface the data 'abcd<CR><LF>' is received. The data 'abcd' would be transmitted on the CAN after <LF> had been received.



8.12.2.4 *Serial_Baudrate*

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

Parameter name	***Serial_Baudrate***
Object index, Sub-index	2800_h, 03_h (COM A)/ 2900_h, 03_h (COM B)
Access mode	rw
Data type	unsigned 8
Value range	$00...FF_h$
Unit	n.a.
Default value	22_h => 9600 bit/s

Description: Via the parameter *Serial_Baudrate* the Rx- and Tx-baud rate of the serial interface can be set via in 14 steps.

	Serial_Baudrate	
Bit:	7 4	3 0
Parameter:	*Rx-Baudrate*	*Tx-Baudrate*

Table 21: Structure of the parameters *Serial_Baudrate*

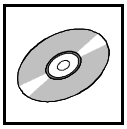
The baud rate determines the speed at which data are transmitted (Tx) or received (Rx) on the serial interfaces. The default setting for Rx- and Tx-baud rates is 9600 kbit/s.
The physically attainable baud rate is limited by the hardware (see page 112).

Note:
For Rx- and Tx-baud rate the same value has to be selected!

Note:
Via parameter *Special_Baudrates* (2800_h, C_h) further baud rates can be set (see page 112)! If you want to do so, the value E_h has to be specified for parameters *Rx-Baudrate* and *Tx-Baudrate*.

Parameter *Rx-(Tx) Baudrate*	Standard baud rate [bit/s]
00_h	38400
01_h	19200
$\underline{02}_h$	**9600**
03_h	4800
04_h	2400
05_h	1200
06_h	600
07_h, 08_h	300
09_h	7200
$0A_h$	14400
$0B_h$	28800
$0C_h$	57600
$0D_h$	115200
$0E_h$	*Special_ Baudrate*
$0F_h$	76800

Table 22: Setting the baud rate of the serial interface in 14 steps



8.12.2.5 *Serial_Mode*

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

Parameter name	***Serial_Mode***
Object index, Sub-index	2800_h, 04_h (COM A)/ 2900_h, 04_h (COM B)
Access mode	rw
Data type	unsigned 8
Value range	$00...FF_h$
Unit	n.a.
Default value	73_h

Description: The bits of the parameter *Serial_Mode* are assigned with the following functions:

Bit	7	6	5	4	3	2	1	0
Assignment	*RTS-Mode*	*CTS-enable*	*Stop-Bit*	*Parity-Mode*		*Parity-Type*	*Bits per Character*	
Default	0	1	1	1	0	0	1	1

Table 23: Assignment of the parameter *Serial_Mode*

The serial interface can only handle 8 bits per character. Furthermore the maximum number of serial bits is limited to 11 by the controller of the serial interface. Therefore, for the bits 4...0 the following permissible combinations result:

Bit						Function
7, 6, 5	4	3	2	1	0	
RTS,CTS, Stop: see following tables	*Parity-Mode*		*Parity-Type*	*Bits per Character*		
	0	0	$O/\overline{E}$	1	1	Tx-Parity acc. to Bit ‘Parity Type’, no Rx-Parity, 1 Stop-bit
	0	1	$O/\overline{E}$	1	1	Tx Parity and Rx-Parity acc. to bit ‘Parity Type’, 1 Stop-bit
	1	0	0	1	1	No Parity, evaluate 1 or 2 Stop-bits
	1	0	1	1	1	Transmitter in multidrop mode, if *RTS-Mode* = ‘1’ (see page 104)
	1	1	a	1	1	Force Tx-Parity to value of ‘a’, 1 Stop-Bit

Table 24: Permissible combinations of the bits 4...0

Explanation of bits of parameter *Serial_Mode*:

RTS-Mode... For the operation as RS-485 interface the RTS-modem mode for RS-485 interfaces has to be selected via this bit.

RTS-Mode	Evaluation
0	RTS on Rx (default setting) hardware-handshake signal
1	RTS-modem (RS-485)

Table 25: Evaluation of *RTS-Mode* bit

CTS enable... This bit is only important, if the RTS-mode is set to '0'!
Via this bit the CTS-function of the serial controller is enabled.
If the bit is '0', the CTS-signal is not evaluated and the controller transmits the available data immediately (if the transmitter is not blocked by other commands; such as 'suspended', <Xoff> received, number of data < *MinChar*).

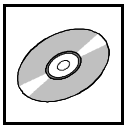
CTS enable bit	CTS-evaluation
0	CTS-input is ignored
1	CTS-input is evaluated: hardware handshake (default setting)

Table 26: Evaluation of *CTS enable* bits

Stop Bit... Here the number of stop bits of the serial interface is determined:

Stop Bit	Number of Stop bits
0	1 Stop bit
1	2 Stop bits (default setting)

Table 27: Number of Stop bits



Parity Mode... By means of these two bits the evaluation of the parity bit is specified:

Parity Mode		Evaluation
Bit 4	Bit 3	
0	0	parity evaluation when receiving data and transmitting the parity bit
0	1	parity bit is only transmitted
1	0	no parity evaluation, no parity transmission (default setting)
1	1	value of the parity bit to be transmitted is specified in bit *Parity Type* (‘forced parity’)

Table 28: Parity evaluation

Parity Type... The polarity of the parity bit is specified by parameter bit *Parity Type*.

Parity Type	Polarity
0	‘even’ (default setting)
1	‘odd’

Table 29: Setting the polarity

Bits per Character... The number of bits per character is set to ‘8’ and cannot be changed. Therefore, these two bits have always to be set to ‘1’!

Explanation to the Multidrop Mode:

Via the bits of the parameter *Serial_Mode* the multidrop mode can be chosen (see table 24).
The multidrop mode can be used for RS-485 networks with multipoint structure. For this the *RTS-Mode* bit has to be set to 1 and the following bits are as described in the following table.

Bit	7	6	5	4	3	2	1	0
Assignment	*RTS-Mode*	*CTS-enable*	*Stop-Bit*	*Parity-Mode*		*Parity-Type*	*Bits per Character*	
Default	1	-	-	1	0	1	1	1

Table 30: Assignment of the parameter *Serial_Mode* for the multidrop mode

The multidrop mode can be used for RS-485 networks with multipoint structure. In this network there is one master and several slaves on a serial bus:

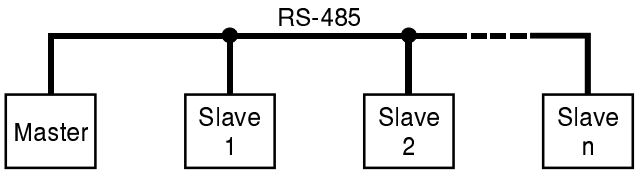


Fig. 26: Structure of a multipoint bus

An addressing is necessary for the distinction of the slaves. The paritity bit is turned into an ‘address-bit’ in the multidrop mode to distinguish the address information from the data. For paritity bit set to ‘1’ the received data are to interpret as address. For paritity bit set to ‘0’, it is process data.

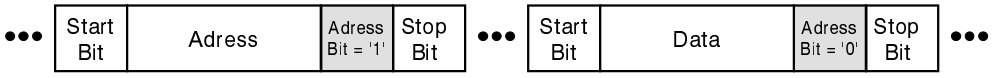
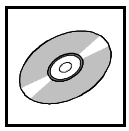


Fig. 27: Structure of the Tx-data in multidrop mode



8.12.2.6 *TxMinCharStart*

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

Parameter name	***TxMinCharStart***
Object index, Sub-index	2800_h, 05_h (COM A)/ 2900_h, 05_h (COM B)
Access mode	rw
Data type	unsigned 8
Value range	$00...FF_h$
Unit	n.a.
Default value	0

Description: This parameter specifies the minimum number of data bytes which have to be stored in the serial Tx-buffer before the transmission of data on the serial interface is started (data CAN -> serial).

This function is required, if strings of more than 8 bytes are to be transmitted together on an RS-485 interface (e.g. with Master/Slave protocols).

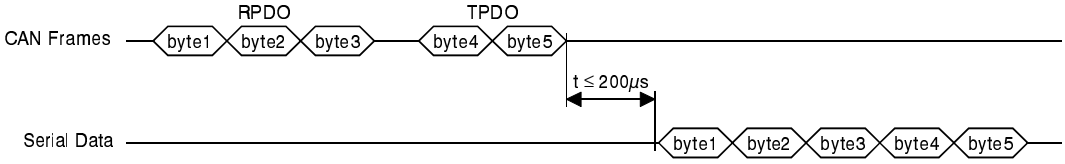


Fig. 28: Function of parameter *TxMinCharStart*

8.12.2.7 *Protocol*

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

Parameter name	***Protocol***
Object index, Sub-index	2800_h, 06_h (COM A)/ 2900_h, 06_h (COM B)
Access mode	rw
Data type	unsigned 8
Value range	$00...63_h$
Unit	n.a.
Default value	00_h

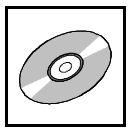
Description: By means of parameter *Protocol* handshake functions of CAN and the serial interface can be selected.

Bit	Name	Meaning
0	*RTR*	0 - no RTR-handshake (default setting) 1 - remote mode (RTR-handshake)
1	*XON/XOFF*	0 - no XON/XOFF 1 - XON/XOFF-handshake enabled
2...4	-	reserved (always set to '0')
5	*Rx_Break*	0 - usage of *Rx_Break* disabled 1 - usage of *Rx_Break* enabled
6	*Tx_Break*	0 - usage of *Tx_Break* disabled 1 - usage of *Tx_Break* enabled
7	-	reserved

Table 31: Bits of the parameter *Protocol*

Bit 0: *RTR_HS* (RTR-Handshake)
In this handshake mode the CAN-CBM-COM2 module responds with an RTR-frame to the Rx-data received by a CAN master (data CAN -> serial) if capacity for at least one more CAN frame is available in the buffer. The RTR-frame is transmitted on the identifier on which the data has been received (RxPDO1).

The module can send RTR-frames cyclically if the data of the CAN master fail to appear. For this option the parameter *RTR_T_Cycle* (2800_h, 13_d, see page 113) has to be set correspondingly and the buffer of the CAN-CBM-COM2 module must be empty.



If the CAN-CBM-COM2 receives an RTR on TxPDO1, the data is transmitted from the Rx-buffer even if the number of data in the buffer is smaller than the number defined in *MinChar*.

Bit 1: *XON/XOFF_EN*
Alternatively to the hardware handshake, this mode can be selected for the serial interface if the handshake is enabled by parameter *Handshake_On/Off*.

Bit 5: *Rx_Break*
This bit can enable or disable the usage of “Rx_Break”.
The *Break* is a zero signal of at least one byte length, so that the start- and the stop-bit have the same level.

Bit 6: *Tx_Break*
This bit can enable or disable the usage of “Tx_Break”.
The *Break* is a zero signal of at least one byte length, so that the start- and the stop-bit have the same level.

8.12.2.8 *Handshake_On, Handshake_Off*

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

Parameter name	***Handshake_Off***	***Handshake_On***
Object index, Sub-index	2800_h, 07_h (COM A)/ 2900_h, 07_h (COM B)	2800_h, 08_h (COM A)/ 2900_h, 08_h (COM B)
Access mode	rw	rw
Data type	unsigned 8	unsigned 8
Value range	$00...FF_h$	$10...FF_h$
Unit	n.a.	n.a.
Default value	$F6_h$	$0A_h$

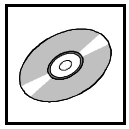
Description: By means of these parameters the handshake function of the serial interface can be set for the data direction serial -> CAN (*this function is not supported in RTS-Modem Mode!* See page 102).

The buffer for the serial data received has a capacity of 256 bytes. In order to prevent the buffer from overflowing and data to be lost, the transmitter of the serial data can be told via the handshake line RTS or via protocol byte <Xoff> to suspend transmission until the buffer has capacity for new data again.

In parameter ***Handshake_On*** the number of bytes received in the Rx-buffer is entered from which the serial data flow is to be suspended. In default setting this value is $F6_h$, i.e. if 250 bytes are in the Rx-buffer, the RTS-signal of the serial interface will be activated. If *XON/XOFF_EN* is enabled by parameter *Tx_Start/Protocol,* the software handshake <Xoff> also signalizes that the buffer is not ready to receive.

In parameter ***Handshake_Off*** the number of bytes in the Rx-buffer is specified from which the reception of serial data will be enabled again. In default setting this value is $0A_h$, i.e. if only 10 bytes are left in the Rx-buffer, the RTS-signal of the serial interface will be deactivated. If *XON/XOFF_EN* is enabled via parameter *Tx_Start/Protocol,* the software handshake <Xon> also signalizes that the buffer is ready to receive again.

The value of parameter *Handshake_Off* must always be smaller than the value of parameter *Handshake_On*!
If the bit *RTS_Mode* = '0', the RTS-signal always has the current handshake, regardless of the bit *XON/XOFF_EN*.



Note:
Parameter *Handshake_On/Off* does not have an effect on the handshake performance in the opposite data direction, i.e. CAN -> serial.

The transmitter of the serial interface is enabled
- if the bit *CTS-Mode* = '1' and the CTS-signal is inactive, or
- if *XON/XOFF_EN* = '1' and <Xoff> has been received.

8.12.2.9 *TxByRxTimeout*

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

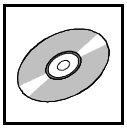
Parameter name	***TxByRxTimeout***
Object index, Sub-index	2800_h, 09_h (COM A)/ 2900_h, 09_h (COM B)
Access mode	rw
Data type	unsigned 8
Value range	$00...FF_h$
Unit	ms
Default value	0

Description: The serial data received is usually only transmitted on CAN after the number of bytes specified in *MinChar* has been received by the serial interface.

For large data rates it is useful to set parameter *Minchar* = '8' in order to get as much data into a CAN frame as possible. It is possible, however, that less bytes than specified in *MinChar* are stored in the buffer of the CAN-CBM-COM2 with the last transmission. This data would then only be transmitted when transmissions will be resumed again. In order to prevent these data from remaining in the buffer for an unspecified period, parameter *TxByRxTimeout* (*Transfer_by_Rx-Timeout)* has been introduced.

If at least one data byte is in the serial Rx-buffer and no further data is received by the serial interface within a specified timeout, the data of the Rx-buffer will be transmitted on CAN even if the number of bytes is still smaller than specified in *MinChar.*

In default setting *TxByRxTimeout* = '0' and therefore the transmission function is blocked. The value for *TxByRxTimeout* is specified in [ms]. Values up to 255 ms are permissible.



8.12.2.10 *End_Char_1, End_Char_2*

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

Parameter name	***End_Char_1***	***End_Char_2***
Object index, Sub-index	2800_h, $0A_h$ (COM A)/ 2900_h, $0A_h$ (COM B)	2800_h, $0B_h$ (COM A)/ 2900_h, $0B_h$ (COM B)
Access mode	rw	rw
Data type	unsigned 8	unsigned 8
Value range	00...FF_h	00...FF_h
Unit	n.a.	n.a.
Default value	$0D_h$	$0A_h$

Description: By means of parameter *MinChar* described above, the transmission of serial data received on CAN can be initiated after the end commands *End_Char_1* or *End_Char_2* have been received.

The characters whose reception is evaluated as end command can be defined by means of parameter *End_Char_1* and *End_Char_2*, i.e. other characters than <CR> or <LF> can be selected. In default setting *End_Char_1* is assigned by <CR>, that is $0D_h$, and *End_Char_2* is assigned by <LF>, that is $0A_h$.

8.12.2.11 *Special_Baudrates*

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

Parameter name	***Special_Baudrates***
Object index, Sub-index	2800_h, $0C_h$ (COM A)/ 2900_h, $0C_h$ (COM B)
Access mode	rw
Data type	unsigned 16
Value range	$0001...07FF_h$
Unit	n.a.
Default value	0034_h

Description: The baud rate of the serial interface can be set in 14 steps by means of parameter *Serial_Baudrate*. Further baud rates can be set with the parameter *Special_Baudrates* if *Serial Baudrate* is set to EE_h.

In *Special_Baudrates* an integer value 'N' is specified with a value range between:

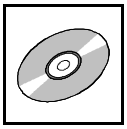
$N = 1....07FF_h$, i.e. $1....2047_d$.

The baud rates that can be set are determined by means of the following equation:

$$\text{Baud rate [Baud]} = \frac{8 \cdot 10^6}{16 \cdot N} = \frac{500\,000}{N}$$

Baud rates can be set from 244 Baud to 500 000 Baud.

Because *N* has to be an integer, it is not possible to attain all standard baud rates exactly!



8.12.2.12 *RTR_T_Cycle_Time*

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

Parameter name	***RTR_T_Cycle_Time***
Object index, Sub-index	2800_h, $0D_h$ (COM A)/ 2900_h, $0C_h$ (COM B)
Access mode	rw
Data type	unsigned 16
Value range	$0000...FFFF_h$
Unit	ms
Default value	1388_h => 5000 ms

Description: The *RTR_T_Cycle_Time* is only important, if the CAN-CBM-COM2 module runs in RTR-handshake mode. The bit *RTR* in parameter *Protocol* (see page 106) enables the RTR-handshake mode.

The CAN-CBM-COM2 module responds in the RTR-handshake mode with an RTR-frame to the Rx-data (data CAN -> serial) received by a CAN master if there is still enough space left in the buffer for at least one further CAN frame. The RTR-frame is transmitted via the same identifier on which the data are received (RxPDO1).

The module can send RTR-frames cyclically if the data of the CAN master fail to appear. For this option the value of the parameter *RTR_T_Cycle* must be higher than '0' and the buffer of the CAN-CBM-COM2 module must be empty.

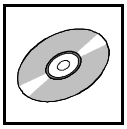
8.12.2.13 *Tx_Cycle_Time*

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

Parameter name	***Tx_Cycle_Time***
Object index, Sub-index	2800_h, $0E_h$ (COM A) / 2900_h, $0E_h$ (COM B)
Access mode	rw
Data type	unsigned 16
Value range	0000...$FFFF_h$
Unit	ms
Default value	0

Description: For recurring transmission orders with the same content, as e.g. used for the initialization of gauges on the serial bus, the CAN-CBM-COM2 module offers a function for cyclic transmission of a programmable character string. The function relieves the CAN bus because the character string must be sent to the CAN-CBM-COM2 module only once.

The time for cyclic transmissions of the character string on the serial bus is set via the parameter *Tx_Cycle_Time*. For Tx_Cycle_Time = '0' no transmission will be carried out.
The content of the character string is transmitted via the parameter *Tx_Cycle_String* (object 2810_h/2910_h, see page 117) .



8.12.2.14 *Break_Code*

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

Parameter name	***Break_Code***
Object index, Sub-index	2800_h, $0F_h$ (COM A)/ 2900_h, $0F_h$ (COM B)
Access mode	rw
Data type	unsigned 16
Value range	0000...$FFFF_h$
Default value	0

Description: The parameter *Break_Code* contains the coding of *Break.*

Break_Code	
Tx_Break_Code 15 8	*Rx_Break_Code* 7 0
MSB	LSB

8.12.2.15 *Break_Time_Factor*

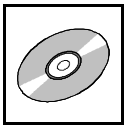
An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

Parameter name	***Break_Time_Factor***
Object index, Sub-index	2800_h, 10_h (COM A)/ 2900_h, 10_h (COM B)
Access mode	rw
Data type	unsigned 8
Value range	$00...FF_h$
Unit	bit times
Default value	11

Description: The parameter *Break_Time_Factor* defines the duration of the *Break* signal in bit times.

Example: baud rate: 9600 Baud
Break_Time_Factor: 11

$$\text{Break Time} = 11 \times \frac{1}{9600 Baud} = 1.146 ms$$



8.12.3 *Tx_Cycle_String* 2810_h, 2910_h

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

The parameters of the objects 2810_h and 2910_h are described together in this chapter because they are identically structured.
The *Tx_Cycle_String* parameters of object 2810_h are used for interface COM A and the *Tx_Cycle_String* parameters of object 2910_h are used for the interface COM B.

Index	Sub-index	Description	Value range	Default value	Data type	Access mode	Save to EEPROM
2810_h / 2910_h	0	*Length of string*	0...F_h	-	unsigned 8	rw	yes
	1	*string_1*	0...FF_h	00	unsigned 8	rw	yes
	:	:	:	:	:	:	:
	F_h	*string_15*	0...FF_h	00	unsigned 8	rw	yes

In this parameter character strings can be programmed which can be transmitted cyclically with the cycle time of *Tx_Cycle_Time* (object 2800_h, D_h, COM A / object 2900_h, D_h, COM B) on the serial bus (see page 114).

Every character string can contain up to 15 bytes.

8.12.4 *Dummy_Rx_Object* 2880_h, 2980_h

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

The parameters of object 2880h and object 2980_h are described together in this chapter, because they are identically structured.
The *dummy-rx-objects* are used as placeholders for the serial data in the Rx-PDOs.

The *dummy_rx_object* (object 2880_h) is used for the interface COM A and the *dummy_rx_object* (object 2980_h) is used for interface COM B.

Index	Sub-index	Description	Value range	Default value	Data Typ	Access mode	Save to EEPROM
2880_h / 2980_h	0	*dummy_rx_object*	-	-	unsigned 8	rw	yes

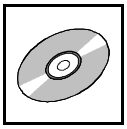
8.12.5 *Dummy_Tx_Object* 2881_h, 2981_h

An overview of the communication parameters 2800_h and 2900_h is shown on page 95.

The parameters of object 2880h and object 2980_h are described together in this chapter, because they are identically structured.
The *dummy-tx-objects* are used as placeholders for the serial data in the Tx-PDOs.

The *dummy_tx_object* (object 2881_h) is used for the interface COM A and the *dummy_tx_object* (object 2981_h) is used for interface COM B.

Index	Sub-index	Description	Value range	Default value	Data type	Access mode	Save to EEPROM
2881_h / 2981_h	0	*dummy_tx_object*	-	-	unsigned 8	rw	yes



8.13 Firmware Management via CANopen Objects

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

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

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

In normal operation mode the objects $1F50_h$ and $1F52_h$ can not be accessed.
The object $1F51_h$ is also available in normal operation mode.
For further information about the objects and the firmware-update please refer to [2].

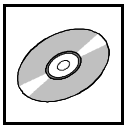
Index	Sub-index	Description	Data type	Access mode
$1F51_h$	1	Boot-Loader: FLASH command	unsigned 8	rw
$1F52_h$	0,1,2	Boot-Loader: Firmware date	unsigned 32	ro

8.13.1 Download Control via Object $1F51_h$

INDEX	**$1F51_h$**
Name	Program Control

Index	Sub-index	Description	Value range	Default value	Data type	Access mode
$1F51_h$	0	*number of programs*	1	1	unsigned 8	ro
	1	*program_number_1*	0...2	1	unsigned 8	rw

Note:
The value range of this object in the implementation for the CAN-CBX-COM2 differs from the value range specified in CiA 302 [2].
For further information about object $1F51_h$ and the firmware-update please refer to [2].



8.13.2 Download Control via Object $1F52_h$

INDEX	**$1F52_h$**
Name	Verify Application Software

In this object the date and the time of the generation of the application software can be stored to check later whether the configuration complies with the expected configuration or not.

Index	Sub-index	Description	Value range	Default-Value	Data type	Access mode
$1F52_h$	0	*number of entries*	2	2	unsigned 8	ro
	1	*application_software_date*	0...FFFF $FFFF_h$	-	unsigned 32	ro
	2	*application_software_time*	0...FFFF $FFFF_h$	-	unsigned 32	ro

Meaning of the variables:

application_software_date — Date of the last application software of the module, specified in days since 01.01.1984.
Example: 0000 $196D_h$ -> 26.10.2001

application_software_time — Time is the number of milliseconds since midnight of the day of the generation of the last application software.
Example: 02A0 $AB8E_h$ -> approx. 12:14 pm

9. References

[1] CiA 301 CANopen Application Layer and Communication Profile V4.2.0 (2011-02)
CAN in Automation (CiA) e. V.

[2] CiA Draft Standard Proposal 302 CANopen Additional Application Layer Functions
Part 3: Configuration and program download V4.1.0 (2010-04)
CAN in Automation (CiA) e. V.

[3] CiA 303 Recommendation
Part 3: Indicator specification V1.4.0 (2012-04)
CAN in Automation (CiA) e. V.

[4] Phoenix Contact GmbH & Co. KG, Blomberg.
Technical data is taken from the Phoenix Contact website:
https://www.phoenixcontact.com/online/portal/de;
PCB plug connector - FKCT-2,5/4-ST KMGY - 1921900, downloaded 2013-05-30

[5] Phoenix Contact GmbH & Co. KG, Blomberg.,
Technical data is taken from the Phoenix Contact website:
https://www.phoenixcontact.com/online/portal/de;
PCB plug connector - FK-MCP 1,5/ 5-STF-3,81 - 1851261, downloaded 2013-05-30

[6] Phoenix Contact GmbH & Co. KG, Blomberg.,
Technical data is taken from the Phoenix Contact website:
https://www.phoenixcontact.com/online/portal/de;
PCB plug connector -FK-MCP- 1,5/7-ST-3,5 - 1939960, downloaded 2013-05-30

10. Declaration of Conformity

EU-KONFORMITÄTSERKLÄRUNG
EU DECLARATION OF CONFORMITY

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

esd erklärt, dass das Produkt *esd declares, that the product*	Typ, Modell, Artikel-Nr. *Type, Model, Article No.*
CAN-CBX-COM2 2xRS232	**C.3055.02**
CAN-CBX-COM2 1xRS232 1xRS485	**C.3055.03**
die Anforderungen der Normen *fulfills the requirements of the standards*	**EN 61000-6-2:2005,** **EN 61000-6-4:2007+A1:2011**
gemäß folgendem Prüfbericht erfüllt. *according to test certificate.*	**H-K00-0473-13**
Das Produkt entspricht damit der EU-Richtlinie „EMV" *Therefore the product corresponds to the EU Directive 'EMC'*	**2014/30/EU**
Das Produkt entspricht der EU-Richtlinie „RoHS" *The product corresponds to the EU Directive 'RoHS'*	**2011/65/EU**

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

Name / *Name*	T. Ramm
Funktion / *Title*	CE-Koordinator / *CE Coordinator*
Datum / *Date*	Hannover, 2014-12-09

Thorsten Ramm

Rechtsgültige Unterschrift / *authorized signature*

I:\Texte\Doku\MANUALS\CAN\CBX\CAN-CBX-COM2\Konformitaet\CAN-CBX-COM2_Konformitaetserklaerung_2014_12_09.odt

11. Order Information

Type	Features	Order No.
CAN-CBX-COM2 2xRS-232	CAN-CBX-COM2 with 2 serial RS-232 interfaces, including 1x CAN-CBX-TBUS (C.3000.01)	C.3055.02
CAN-CBX-COM2 1xRS-232 1xRS-485	CAN-CBX-COM2 with 1x RS-232 and 1x RS-485 interfaces, including 1x CAN-CBX-TBUS (C.3000.01)	C.3055.03
Accessories		
CAN-CBX-TBUS 	Mounting-rail bus connector of the CBX-InRailBus for CAN-CBX-modules, (one bus connector is included in delivery of the CAN-CBX-module)	C.3000.01
CAN-CBX-TBUS-Connector 	Terminal plug of the CBX-InRailBus for the connection of the +24 V power supply voltage and the CAN interface Female type	C.3000.02
CAN-CBX-TBUS-Connection adapter 	Terminal plug of the CBX-InRailBus for the connection of the +24 V power supply voltage and the CAN interface Male type	C.3000.03

Table 32: Order information

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

Manuals		**Order No.**
CAN-CBX-COM2-ME	Manual in English	C.3055.21
CAN-CBX-COM2-MD	Manual in German	C.3055.20

Table 33: Available manuals

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