CAN-CBM-SIO1
CAN-CBM-SIO4
CAN - RS-232, RS-422, RS-485 or TTY-Interface

CAN-CBM-PLC/331-1
Automation Computer with CAN-Interface

Hardware Manual

N O T E

The information in this document has been carefully checked and is believed to be entirely reliable. **esd** makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors that may appear in this document. **esd** reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance or design.

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

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

esd electronic system design gmbh
Vahrenwalder Str. 205
30165 Hannover
Germany

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

USA / Canada
7667 W. Sample Road
Suite 127
Coral Springs, FL 33065
USA

Phone: +1-800-504-9856
Fax: +1-800-288-8235
E-mail: sales@esd-electronics.com

Document file:	I:\texte\Doku\MANUALS\CAN\Cbm\SIO-331\Englisch\CSIO-20H.en6
Date of print:	18.07.2000

PCB version:	CPU331 Rev. 1.1 SIO3311 Rev. 1.1 SIO4 Rev. 1.0

Changes in the chapters

The changes in the user's manual listed below affect changes in the hardware as well as changes in the description of the facts only.

Chapter	Changes versus previous version
-	Description of CAN-CBM-SIO4 module and CAN-CBM-PLC/331-1 module inserted
-	-

Technical details are subject to change without notice.

Contents

Page

Contents

Page

1. Overview

1.1 Description of the Module

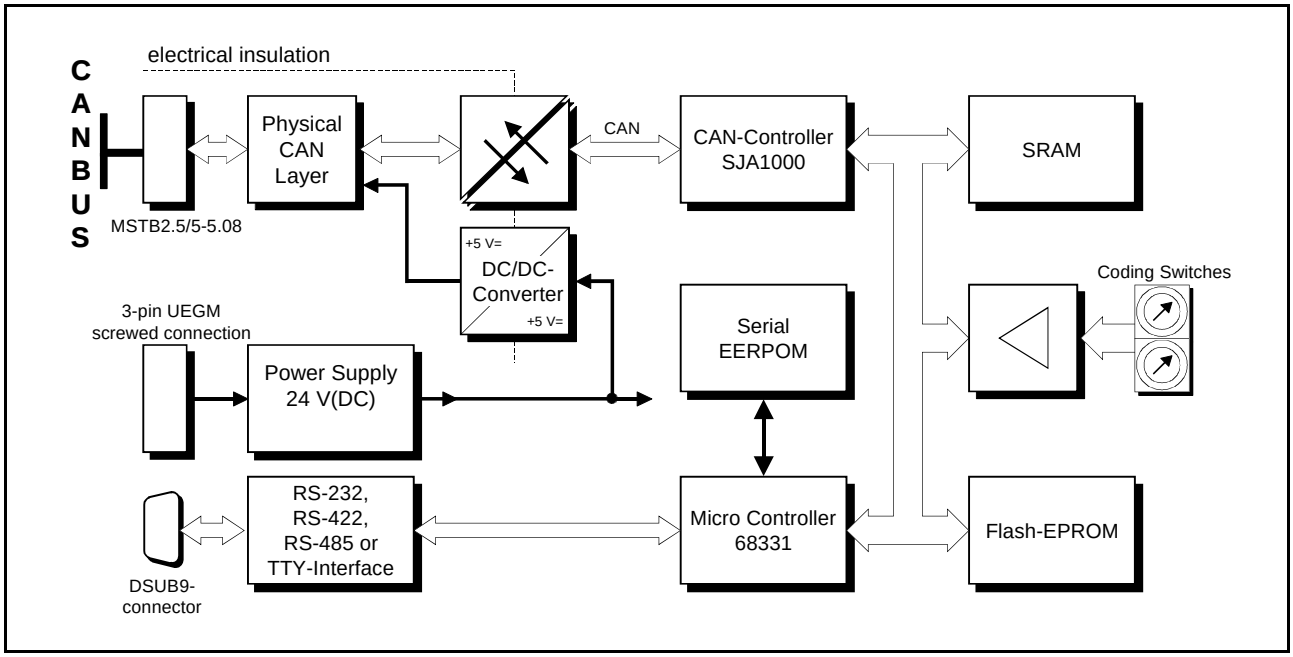


Fig. 1.1: Block-circuit diagram of the CAN-CBM modules

The CAN-CBM-SIO1 and CAN-CBM-PLC/331-1 modules offer the linking of one serial interface with the CAN-net. The CAN-CBM-PLC/331-1 module is configured as SPS controller with the software tool CoDeSys.
The CAN-CBM-SIO4 module is equipped with five serial interfaces. The physical interface of the serial interfaces can be configured like the CAN-CBM-SIO1 module via piggybacks.

The described CAN-CBM modules use a 68331 micro controller, which buffers the CAN-data into a local SRAM. Data security and consistency are guaranteed up to 1 Mbit/s in the CAN-network. The firmware - optional protocols also- is held in the flash.

The ISO 11898-compatible CAN-interface allows a maximum data-transmission rate of 1 Mbit/s. The CAN-interface is electrically insulated by means of optocouplers and DC/DC-converters**.**
The interface is connected via a 5-pin connector with screwed contacts in Combicon style. The module is optionally available with a DeviceNet interface.

The parameters of the serial interface can be configured via CAN - the maximum bit rate is 500 kbit/s. The parameters and the CAN settings are stored into an EEPROM.

Overview

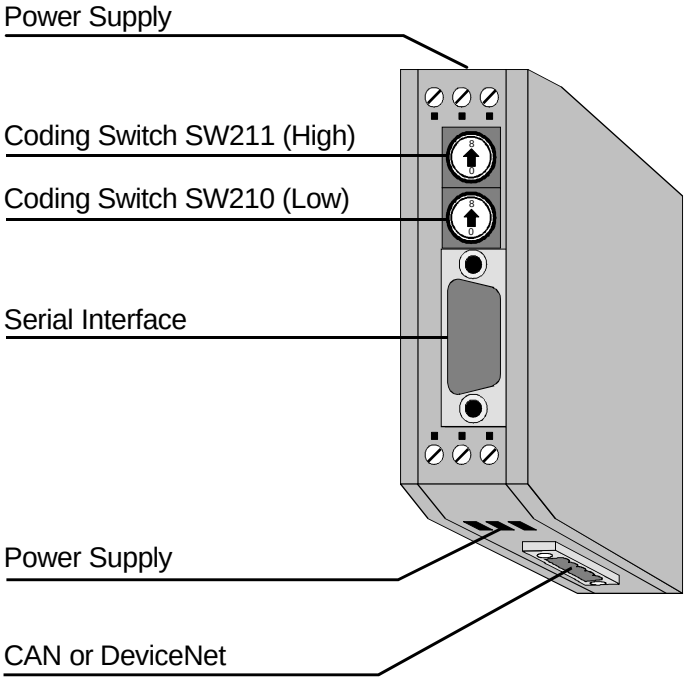
For CAN-CBM-SIO1 and CAN-CBM-SIO4 common protocols like 3964 R, Modbus or also FreePort to the connection of a S7-200 are optionally available. Custom-designed protocols can be made on request or developed with the help of GNU-C surroundings.

By use of the RS-232-interface as modem connection a remote maintenance of the CAN net can be done in remote operation. In addition to RS-232 you can also choose between RS-422, RS-485 or also TTY-20 mA as a physical interface. It is connected via a DSUB9-connector. Beyond that the CAN-CBM-SIO4 is connected via four additional RJ45-sockets.

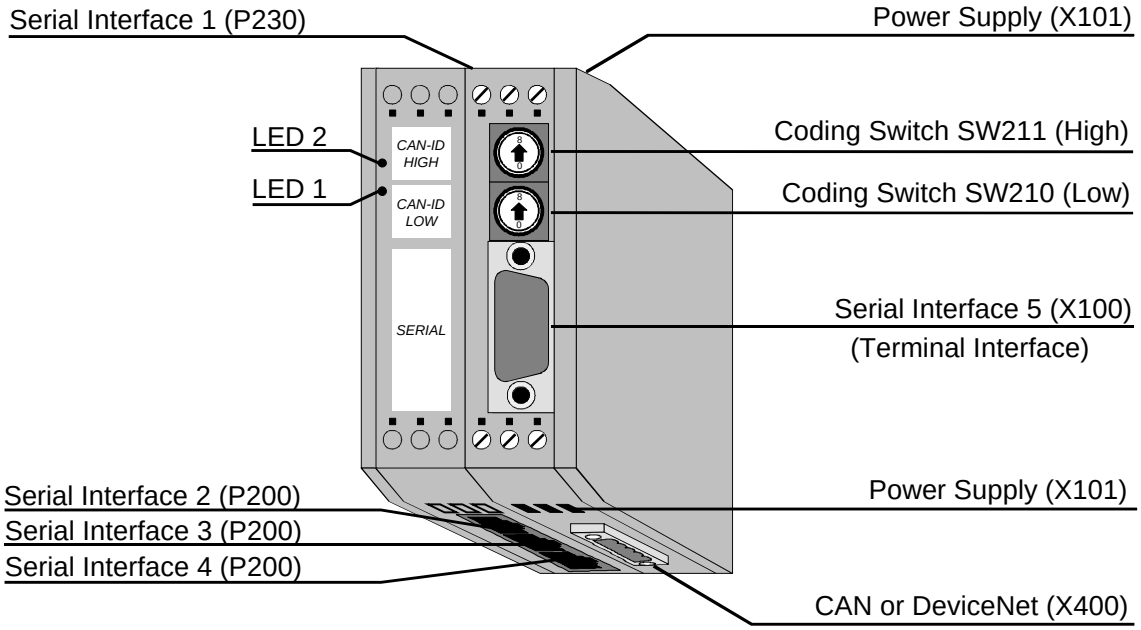
On request the layer-7-protocols CANopen and DeviceNet are supported.

1.2 Front View with Connectors and Coding Switches

1.2.1 CAN-CBM-SIO1 and CAN-CBM-PLC/331-1



1.2.2 CAN-CBM-SIO4



1.3 Summary of Technical Data

1.3.1 General Technical Data

Power supply	nominal voltage: 24 V/DC ±10%, current (at 20°C): max. 70 mA (+20 mA in TTY-operation)
Connectors	X100 (DSUB9, male) - CAN-CBM-SIO1 serial interface 1 CAN-CBM-SIO4 serial interface 5 CAN-CBM-PLC/331-1 serial interface 1 X101 (6-pin screwed connector UEGM) - 24 V-voltage supply X400 (Combicon style, 5-pin MSTB2.5/5-5.08) - CAN or DeviceNet CAN-CBM-SIO4 only: P200 (RJ45-socket) - serial interface 1 P230 (RJ45-socket) - serial interface 2, 3, 4
Temperature range	0...50 /C ambient temperature
Humidity	max. 90%, non-condensing
Case dimensions (B x H x T)	width: 25 mm (CAN-CBM-SIO1, CAN-CBM-PLC/331-1), 40 mm (CAN-CBM-SIO4), height: 85 mm, depth: 83 mm (including hat-rail holder and connector projection DSUB9, without CAN/DeviceNet connector)
Weight	CAN-CBM-SIO1, CAN-CBM-PLC/331-1: ca. 150 g

Table 1.3.1: General data

1.3.2 Micro Controller Unit

Micro controller	68331
Memory	SRAM: 128 k x 16 Bit Flash-EPROM: 128 k x 8 Bit EEPROM: serial I²C-EEPROM
Debug interface	for service and programming

Table 1.3.2: Micro controller unit

1.3.3 CAN/DeviceNet Interface

Number of CAN-interfaces	1 x CAN option: 1 x DeviceNet
CAN-controller	SJA1000, CAN 2.0A/B
Electrical insulation of CAN-interface from other units	via optocouplers and DC/DC-converter
Physical layer CAN	Physical layer in accordance with ISO 11898, transmission rate programmable from 10 kbit/s to 1 Mbit/s
Physical layer DeviceNet (option)	Physical layer in accordance with DeviceNet specification Rev. 2.0, bit rate: 125 kbit/s, 250 kbit/s, 500 kbit/s

Table 1.3.3: Data of CAN-interface

1.3.4 Serial Interface

	Interface at DSUB9 connector	Interface at RJ45 socket (only for CAN-CBM-SIO4)
Channel-assignment for CAN-CBM-SIO1	Channel 1	-
Channel-assignment for CAN-CBM-PLC/331-1	Channel 1	-
Channel-assignment for CAN-CBM-SIO4	Channel 5	Channel 1, 2, 3, 4
Controller	68331	82C684
Interface	standard: RS-232 options: RS-422, RS-485, TTY active / passive	
Connection	9-pin DSUB connector	8-pin RJ45-socket

Table 1.3.4: Data of serial interfaces

1.4 Software Support

The complete EPROM-resident communication firmware for operating the CAN-CBM modules is contained in the product package.

1.4.1 CAN-CBM-SIO1 / CAN-CBM-SIO4

In standard mode without protocol the unit transmits a CAN-frame on the CAN-identifier set before, when receiving 8 ASCII characters - or after receiving a configurable end mark (such as CR, LF or EOT) and after a settable time out expired after no characters had been received anymore.

1.4.2 CAN-CBM-PLC/331-1

The CAN-CBM-PLC/331-1-Module can be configurated with $CoDeSys_{RTOS-UH}$. This is a programming system running under Windows for application control (IEC1131-3) with a run time system under RTOS-UH. The configuration of the CAN-CBM-PLC/331-1 module is described in chapter 5.
The CoDeSys software comes with an online help and a handbook, describing the programming system. Further information on the higher protocol layers can be taken from the CAL/CANopen documentation ‘CiA-Draft Standard 301’.

1.5 Order Information

1.5.1 CAN-CBM-SIO1 / CAN-CBM-SIO4

Type	Features	Order No.
CAN-CBM-SIO	1 x CAN 2.0A/B with RS-232	C.2840.03
CAN-CBM-SIO4	1 x CAN 2.0A/B with (4+1) x RS-232	C.2843.03
Instead of RS-232 with: (please state clearly in order)	RS-422 adaptor RS-485 adaptor TTY-20mA passive TTY-20mA active	X.1930.02 X.1930.04 X.1930.06 X.1930.08
CAN-CBM-SIO	Freeport Protocol	C.2840.42
CAN-CBM-SIO-DvN	DeviceNet Slave	C.2840.13
CAN-CBM-SIO-DvN-M	DeviceNet Master (Scanner)	C.2840.19
CAN-CBM-SIO-Co	CANopen (Slave)	C.2840.18
-	Connection cable 8-pin RJ48 to 8-pin RJ48 Length: 2 m	C.2401.30
-	Adaptor 8-pin RJ45 to 25-pin DSUB/male, Pin arrangement without tools independently configurable	C.2401.34
-	Adaptor 8-pin RJ45 to 25-pin DSUB/female, Pin arrangement without tools independently configurable	C.2401.36
-	Adaptor 8-pin RJ45 to 9-pin DSUB/female, Pin arrangement without tools independently configurable	C.2401.38
-	Adaptor 8-pin RJ45 to 9-pin DSUB/male, Pin arrangement without tools independently configurable	C.2401.40
CAN-CBM-SIO-ME	English manual for C.2840.02 1*)	C.2840.21

1*) If ordered together with the module, the manual is included in the product package.

Table 1.5.1: Order information CAN-CBM-SIO1 and CAN-CBM-SIO4

1.5.2 CAN-CBM-PLC/331-1

Type	Features	Order No.
CAN-CBM-PLC/331-1	1 x CAN 2.0A/B at RS-232	C.2845.03
Instead of RS-232 with: (Please state clearly in order)	RS-422 adaptor RS-485 adaptor TTY-20mA passive TTY-20mA active	X.1930.02 X.1930.04 X.1930.06 X.1930.08
$CoDeSys_{RTOS-UH}$	IEC1131-3 PLC-developing system with 5 program languages; for RTOS-UH; PC-Host	P.4071.02
CAN-CBM-PLC/331-MD	Additional user manual in English [1*)]	C.2845.20

1*) If ordered together with the module, the manual is included in the product package.

Table 1.5.2: Order information for CAN-CBM-PLC/331-1

2. CAN-Identifier

The CAN-CBM-SIO4 module is equipped with one Rx- and one Tx-identifier for each of the five channels. The CAN-CBM-SIO1-module is equipped with one identifier-pair, for the only serial channel.

Module	Physical channel	Receive CAN-Data	Transceiver CAN-data
CAN-CBM-SIO1	Terminal interface on DSUB9 Channel 1	RxID1	TxID1
CAN-CBM-SIO4	Channel 1 Channel 2 Channel 3 Channel 4 Terminal interface on DSUB9 Channel 5	RxID1 RxID2 RxID3 RxID4 RxID5	TxID1 TxID2 TxID3 TxID4 TxID5
CAN-CBM-PLC/331-1	CAN-Identifier must be set by CoDeSys. The CAN-CBM-PLC/331-1 doesn't use the coding switches for any setting.		

Attention: The Rx-Identifier **RxID5** and the Tx-identifier **TxID5** are assigned to terminal-interface (on DSUB9) on CAN-CBM-SIO4 module. On CAN-CBM-SIO1 module with only a single serial interface the Rx-Identifier **RxID1** and the Tx-Identifier **TxID1** are assigned to terminal interface.

Table 2.1: Allocation of serial channels to the identifier of the module

The identifiers are calculated in the default configuration of a base value, which is set by the coding switches, and a fixed offset.

CAN-Identifier

CAN-CBM-SIO4	
Identifier	**Offset (HEX)**
TxID1	0
TxID2	1
TxID3	2
TxID4	3
TxID5	4
RxID1	5
RxID2	6
RxID3	7
RxID4	8
RxID5	9

CAN-CBM-SIO1	
Identifier	**Offset (HEX)**
TxID1	0
RxID1	1

Table 2.2: Offset of the identifier in default setting

Calculation of the base value and the identifier:

base value = 10 x coding switch value
identifier = base value + offset (HEX)

Example:

The coding switches are set to ‘1’. So the setting of the coding switch is $11 and the base value is:

$A x $11 = $AA

The identifier values then arise as follows:

$AA + offset (HEX) = identifier

CAN-CBM-SIO4	
Identifier	**Value (HEX)**
TxID1 TxID2 TxID3 TxID4 TxID5	AA AB AC AD AE
RxID1 RxID2 RxID3 RxID4 RxID5	AF B0 B1 B2 B3

CAN-CBM-SIO1	
Identifier	**Value (HEX)**
TxID1	AA
RxID1	AB

Table 2.3: Example for identifier settings

3. Unit Description

3.1 CAN/DeviceNet Unit

3.1.1 Interface Circuit

The CAN-CBM modules are available with a CAN-interface in accordance with ISO11898 or alternatively with a DeviceNet interface. The same connector is used for both interfaces. The connector assignment is different, however. The following figures represent the two interfaces.

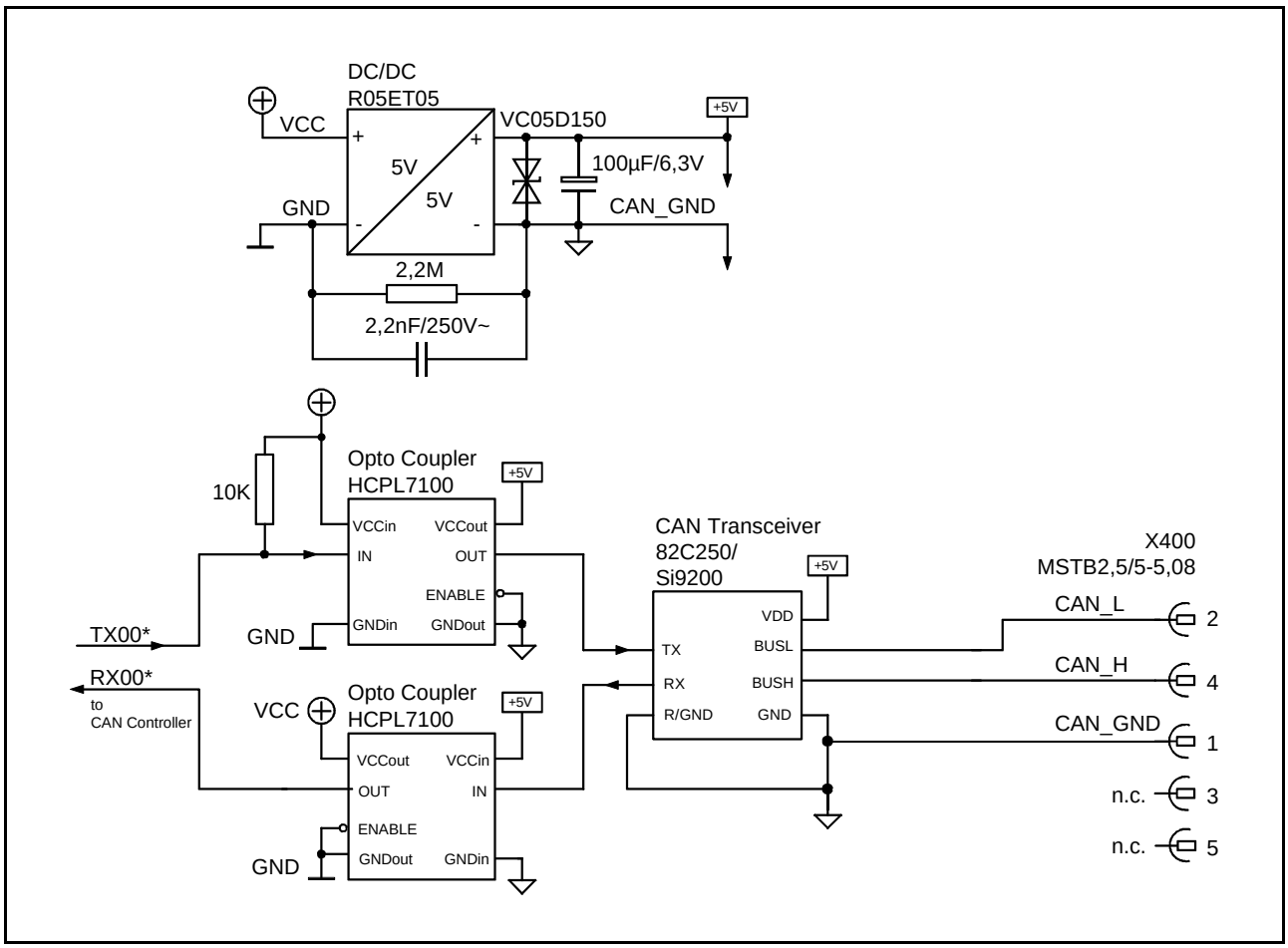


Fig. 3.1.1: Circuit of CAN-interface

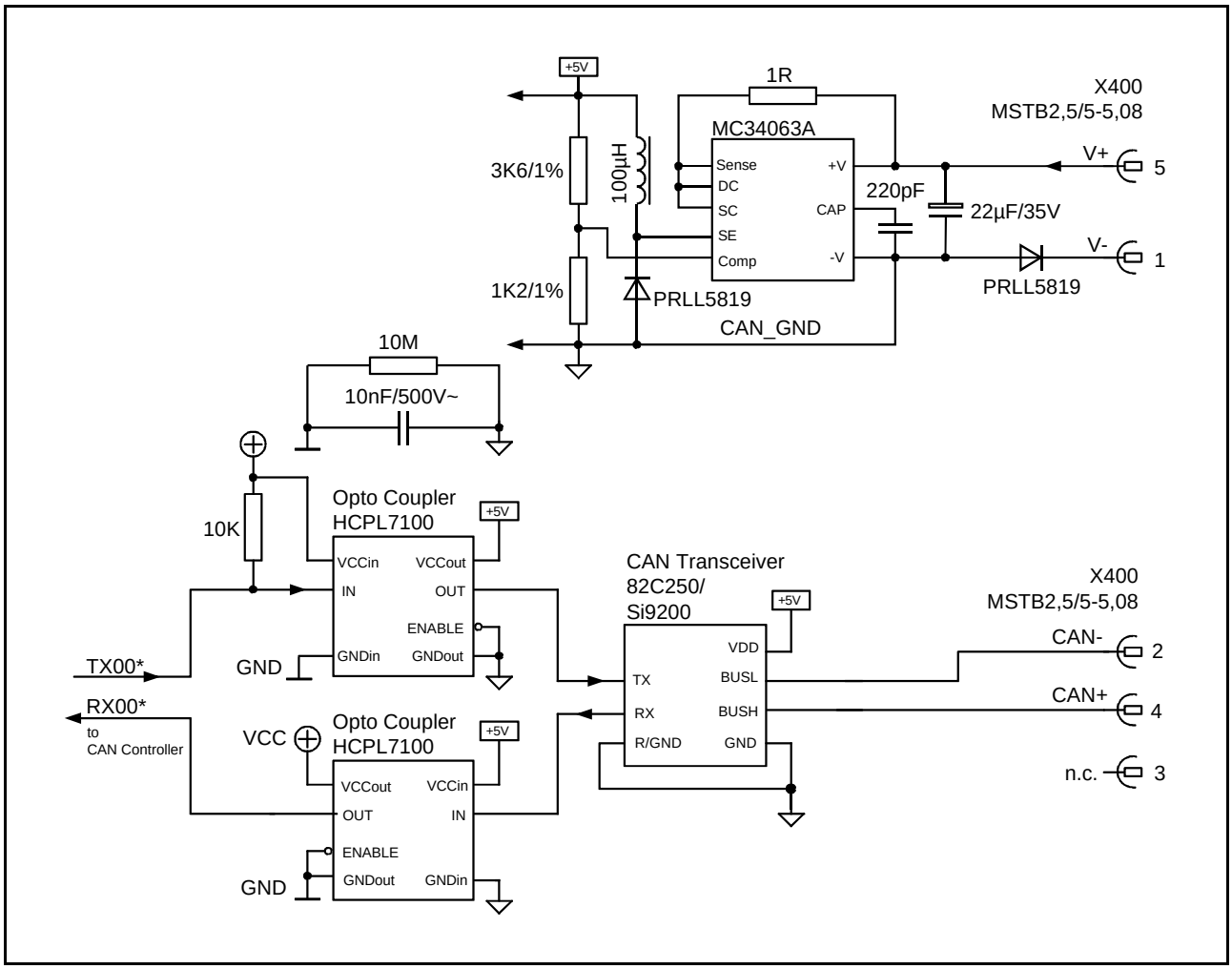


Fig. 3.1.2: Circuit of DeviceNet interface

3.2 Serial Interface X100 (9-pin DSUB/ Male)

3.2.1 Configuration

The physical interface of the serial interface can be configured as an RS-232-, RS-422-, RS-485-, TTY-active- or TTY-passive-interface. For RS-232 operation an RS-232A driver component is used, for the other interfaces piggy backs are used.

The serial interface is controlled by the 68331 controller and by QUART 82C684 . The bit rate of the interface can be parameterized.

The controller QUART 82C684 supports bit rates of up to 230 kbit/s. If the 4 interfaces are run at the same time only 38,4 kbit can be attained.

The controller integrated in the 68331 supports bit rates of up to 500 kbit/s in this application.

Bit rates of over 38.4 kbit/s can only be achieved by means of RS-422 and RS-485 interfaces. With the RS-232 drivers used a maximum of 38.4 kbit/s is possible.

Unit	maximum bit rate
Controller: -68331 -Quart 82C684	 500 kbit/s 230 kbit/s (38,4 kbit/s)
RS-422 interface	500 kbit/s
RS-485 interface	500 kbit/s
RS-232 interface	38.4 kbit/s
TTY-interface	38.4 kbit/s

Table 3.2.1: Attainable bit rates for the different physical interfaces

The following bit rates can be set by means of the software. The values in the second column represent the actual bit rates which result from the 68331 controller-internal conversion.

Bit rate (reference value) [bit/s]	Bit rate (actual value) [bit/s]
500,000 (only 68331)	500,000
38,400	38,462
19,200	19,231
9,600	9,615
4,800	4,808
2,400	2,404
1,200	1,199
600	600.2
300	299.9

Table 3.2.3: Settable bit rates

3.2.2 Connection of the Various Serial Interfaces at DSUB9 Connector

Below the wiring of the serial interfaces is represented for channel 1 (CAN-CBM-SIO1 and CAN-CBM-PLC/331-1) and channel 5 (CAN-CBM-SIO4). The figures help to explain the short terms used in for the signals in the appendix (Connector Assignment). Furthermore the circuit diagrams of the various available piggybacks can be found in the appendix (Circuit Diagrams).
The signal terms are specified exemplary for the connection of the CAN-CBM modules as transmitter (Terminal DTE).

3.2.2.1 The RS-232 Interface

The signals CTS, DSR and DCD aren't evaluated by the CAN-CBM modules.

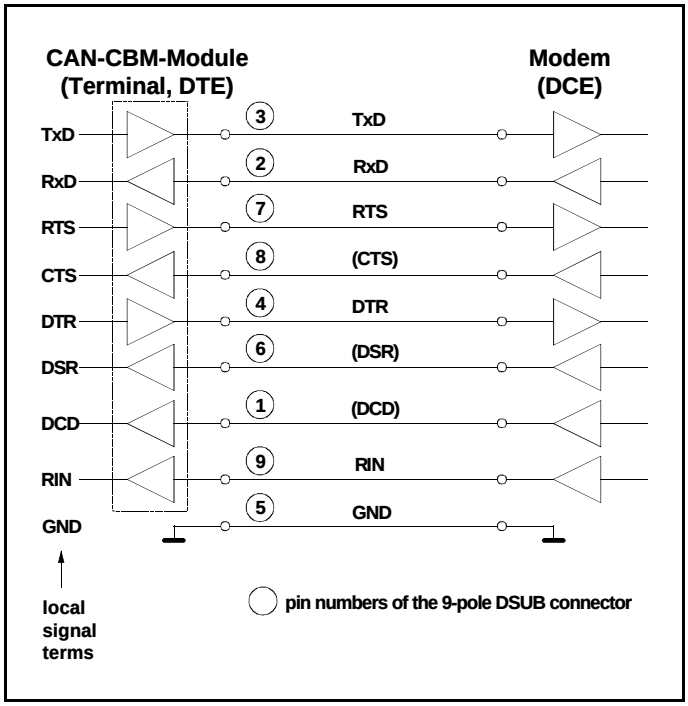


Fig. 3.2.3: Connection diagram for RS-232 operation

3.2.2.2 RS-422 Interface

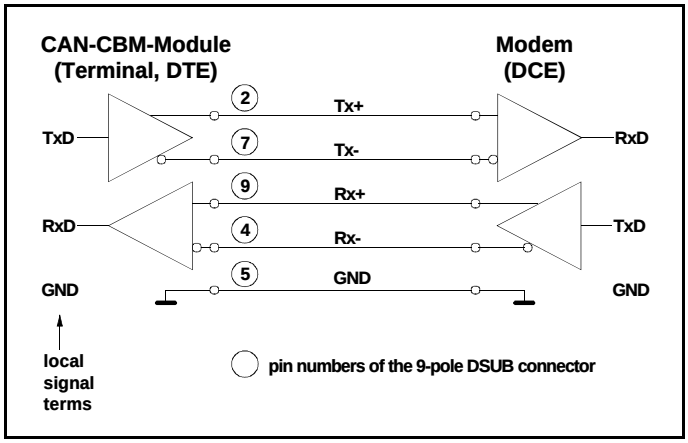


Fig. 3.2.4: Connection diagram for RS-422 operation

3.2.2.3 RS-485 Interface

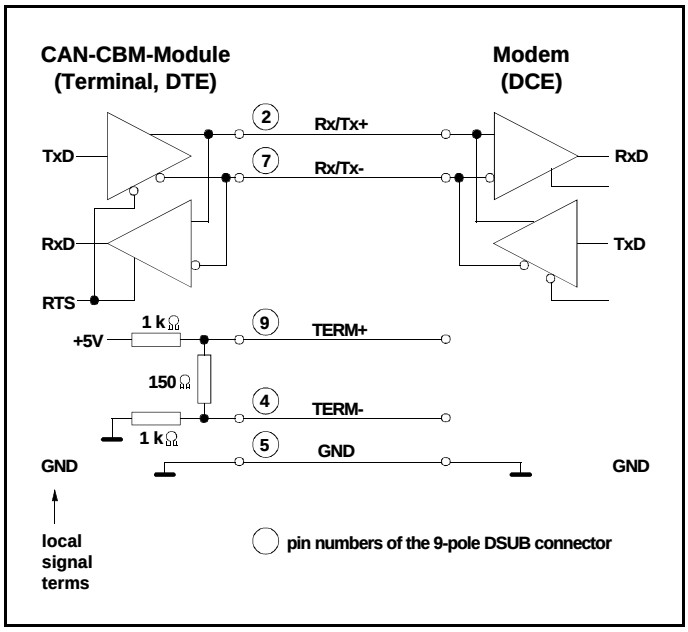


Fig. 3.2.5: Connection diagram for RS-485 operation

In order to activate the terminating-impedance network on the piggyback, you have to connect pins 9 and 2 and pins 4 and 7, e.g. in the DSUB-connector.

3.2.2.4 TTY(20 mA)-Interface

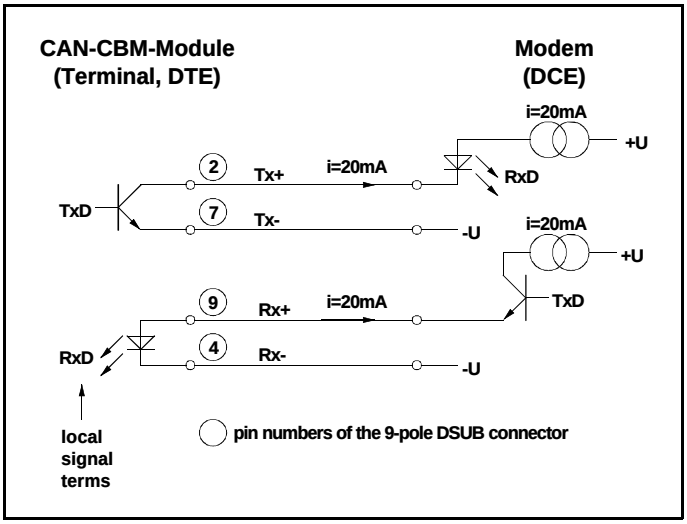


Fig. 3.2.6: Connection diagram for TTY-operation (passive)

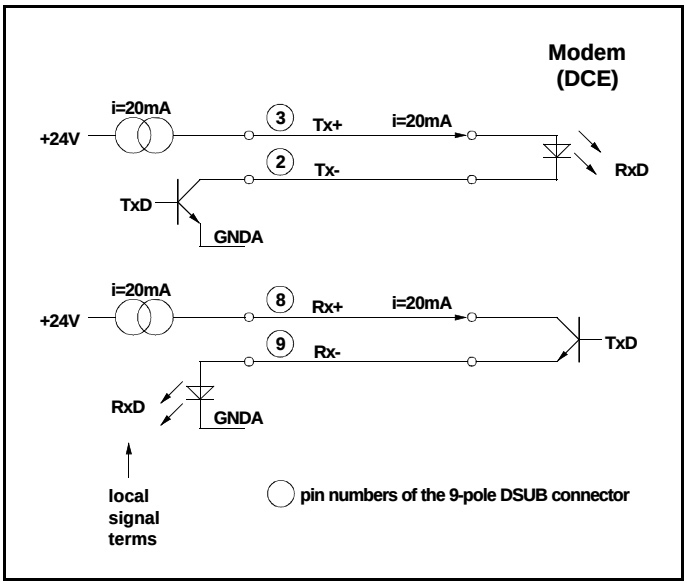


Fig. 3.2.7: Connection diagram for TTY-operation (active)

3.2.3 Connection of the Various Serial Interfaces on RJ45-Sockets

Below the wiring of the serial interfaces of CAN-CBM-SIO4 in relation to the data direction is shown. The figures should explain the short terms used in for the signals in the chapter *Connector Assignment*. Furthermore the circuit diagrams of the various available piggybacks can be found in the chapter *Circuit Diagrams*.

As example for the connection cable the adapter cable RJ48-DSUB9/female has been shown here which is layed out for the RS-232 modem operation (data communication equipment).

The conduction marked by RTS can be programmed as RTS- or DTR-signal in the Controller 82C684. The module-software programs the signal as RTS-input. The RTS wiring can be connected to the DTR pin, if the terminal needs a DTR signal as answer.

3.2.3.1 RS-232-Interface

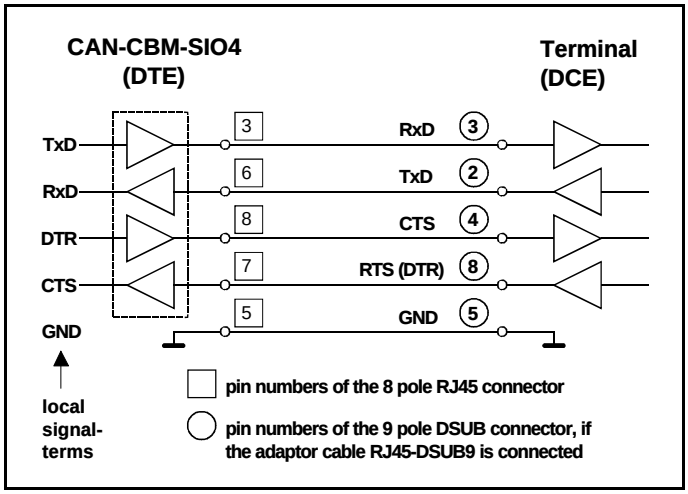


Fig. 3.2.8: Connection-diagram for RS-232 operation

3.2.3.2 RS-422-Interface

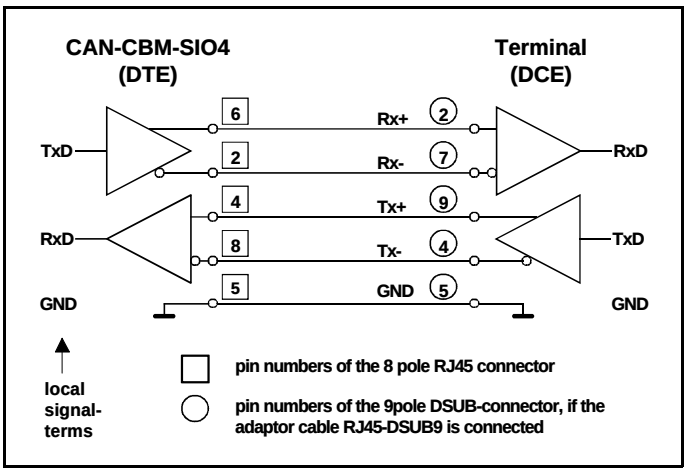


Fig. 3.2.9: Connection diagram for RS-422 operation

3.2.3.3 RS-485 Interface

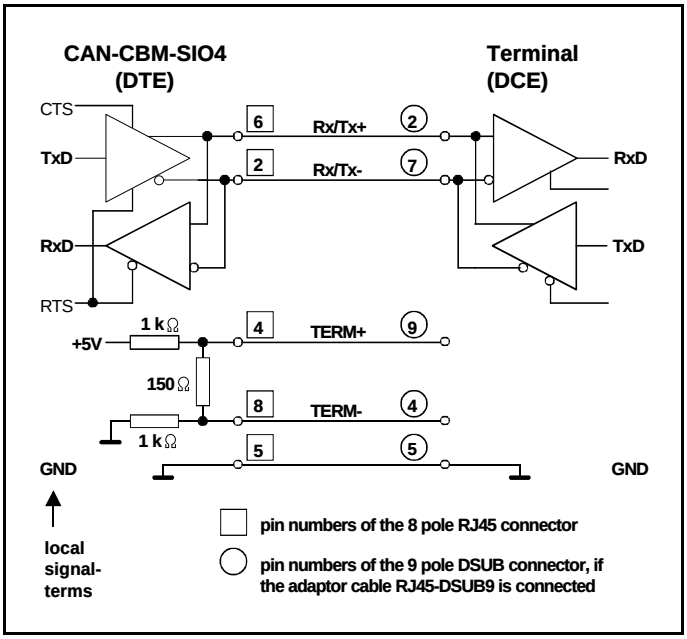


Fig. 3.2.10: Connection diagram for RS-485 operation

Pin 4 and 8 of the RJ45 socket lead in RS-485 operation to a termination resistor, on the piggyback. To activate the termination, the signal Rx/Tx+ has to be connected to TERM+ and the signal Rx/Tx- to TERM-.

3.2.3.4 TTY(20 mA) Interface

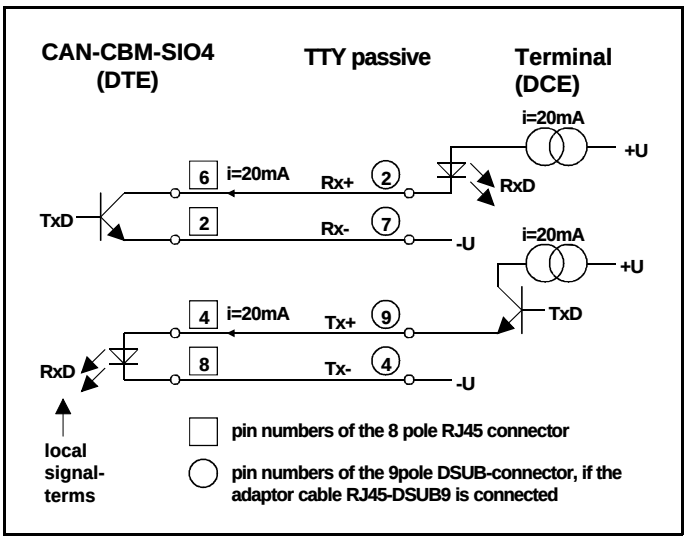


Fig. 3.2.11: Connection diagram for TTY operation (passive)

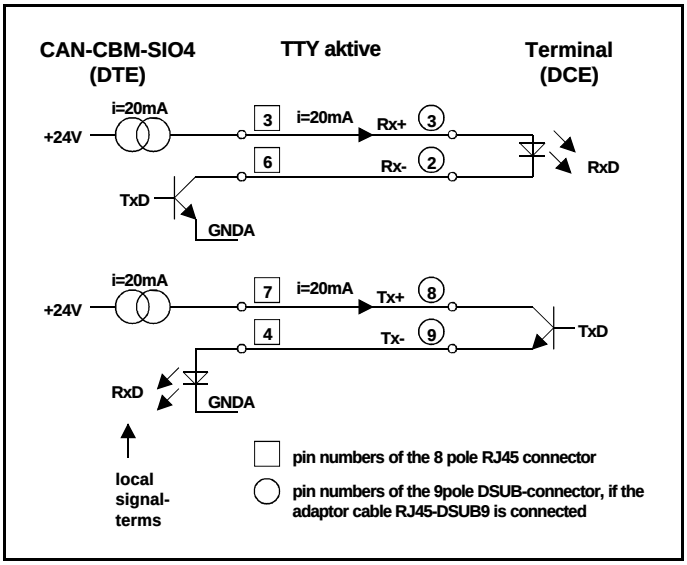
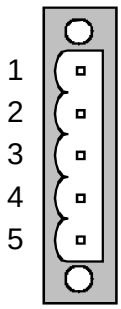


Fig. 3.2.12: Connection diagram for TTY operation (active)

4. Connector Assignments

4.1 CAN (X400, 5 pole Combicon Style)

Pin Position:



Pin Assignment:

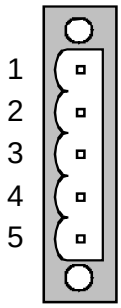
Pin	Signal
1	CAN_GND
2	CAN_L
3	n.c.
4	CAN_H
5	n.c.

Signal Terms:

CAN_L, CAN_H...	CAN-signal lines
CAN_GND ...	reference potential of the local CAN-physical layer
n.c...	not connected

4.2 DeviceNet (X400, 5 pole Combicon Style)

Pin Position:



Pin Assignment:

Pin	Signal
1	V-
2	CAN-
3	n.c.
4	CAN+
5	V+

Signal Terms:

V+...	Voltage supply feed (U_{VCC} = 24 V ± 4%)
V-...	reference potential of V+ and CAN+/CAN-
CAN+, CAN-...	CAN-signal lines
n.c. ...	not connected

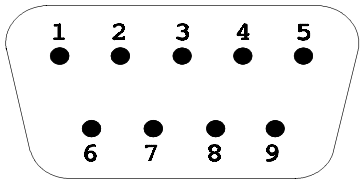
4.3 Assignment of the Serial Interface on DSUB9

Notes to the connection of the serial interfaces can also be taken from the chapter *'Connection of the Various Serial Interfaces at DSUB9 Connector'*. You find the directions of the signals (Rx<->Tx) in the connection diagrams.

4.3.1 RS-232 Interface (X100, 9-pin DSUB / Male)

The signals CTS, DSR and DCD are not evaluated by the CAN-CBM modules!

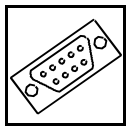
Pin Position:



Pin Assignment:

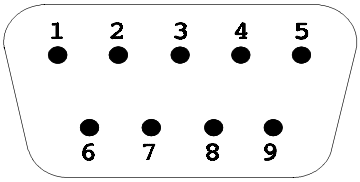
Signal		Pin		Signal	
(DSR)	(input)	6	1	(DCD)	(input)
RTS	(output)	7	2	RxD	(input)
(CTS)	(input)	8	3	TxD	(output)
RIN	(input)	9	4	DTR	(output)
			5	GND	

9-pin DSUB-connector



4.3.2 RS-422 Interface (X100, 9-pin DSUB / Male)

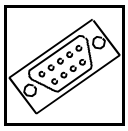
Pin Position:



Pin Assignment:

Signal	Pin		Signal
-	6	1	-
Tx- (output)	7	2	Tx+ (output)
-	8	3	-
Rx+ (input)	9	4	Rx- (input)
		5	GND

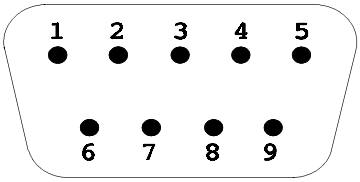
9-pin DSUB-connector



Connector Assignment

4.3.3 RS-485 Interface (X100, 9-pin DSUB / Male)

Pin Position:

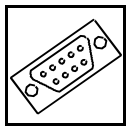


Pin Assignment:

Signal	Pin		Signal
-	6	1	-
Rx/Tx-	7	2	Rx/Tx+
-	8	3	-
Term+ (for Rx/Tx+)	9	4	Term-(for Rx/Tx-)
		5	GND

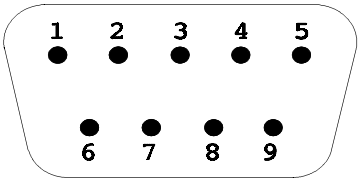
9-pin DSUB-connector

The signals Term+ and Term- are connected to a terminating-impedance network on the board. In order to activate the connection, Term+ has to be connected to the Rx/Tx+ signal and Term- to the Rx/Tx- signal.



4.3.4 TTY-passive-Interface (X100, 9-pin DSUB / Male)

Pin Position:



Pin Assignment:

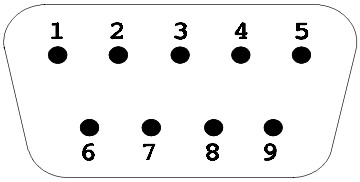
Signal	Pin		Signal
-	6	1	-
Tx- (transmitter)	7	2	Tx+ (transmitter)
(I2+)	8	3	(I1+)
Rx+ (recipient)	9	4	Rx- (recipient)
		5	GND

9-pin DSUB-connector

( ) The signals specified in brackets are assigned, but are not required for operating this physical interface.

4.3.5 TTY-Active Interface (X100, 9-pin DSUB / Male)

Pin Position:



Pin Assignment:

Signal	Pin		Signal
-	6	1	-
(GNDA)	7	2	Tx- (transmitter)
Rx+ (recipient)	8	3	Tx+ (transmitter)
Rx- (recipient)	9	4	(GNDA)
		5	GND

9-pin DSUB-connector

( ) The signals specified in brackets are assigned, but they are not required for operating this physical interface.

4.4 Connector Pin Assignment of the Serial Interface of RJ45 Socket

Only CAN-CBM-SIO4 is mounted with this interface!

Notes to the connection of the serial interfaces can also be taken from the chapter ‘Connection of the Various Serial Interfaces at DSUB9 Connector’. You find the directions of the signals (Rx<->Tx) in the connection diagram.

4.4.1 Serial Interface 2...4 (P200/P230, 8-pin RJ45 Socket)

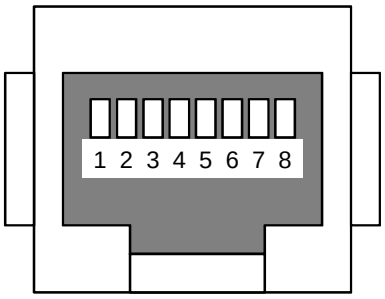


Fig. 4.4.1: Pin assignment of RJ45 socket

4.4.2 Pin Assignment of the 8 Pin RJ45 Sockets (P200/230)

The signal names used in the table below correspond to the physical data directions seen from the CAN-CBM-SIO4, i.e., the TxD signal is an output and has to be connected to the RxD line of the other device.

Connector Pin RJ45	Signal arrangement				
	RS-232	**RS-422**	**RS-485**	**TTY-passive**	**TTY-aktive**
1	-	-	-	-	-
2	-	Tx-	Rx/Tx-	Tx-	[GNDA]
3	TxD Data Output	-	-	[I1+]	Tx+
4	-	Rx+	TERM+ *1)	Rx+	Rx-
5	GND	GND	GND	GND	GND
6	RxD Data Input	Tx+	Rx/Tx+	Tx+	Tx-
7	CTS Handshake Input	GND	GND	[I2+]	Rx+
8	RTS Handshake Output	Rx-	TERM- *1)	Rx-	[GNDA]

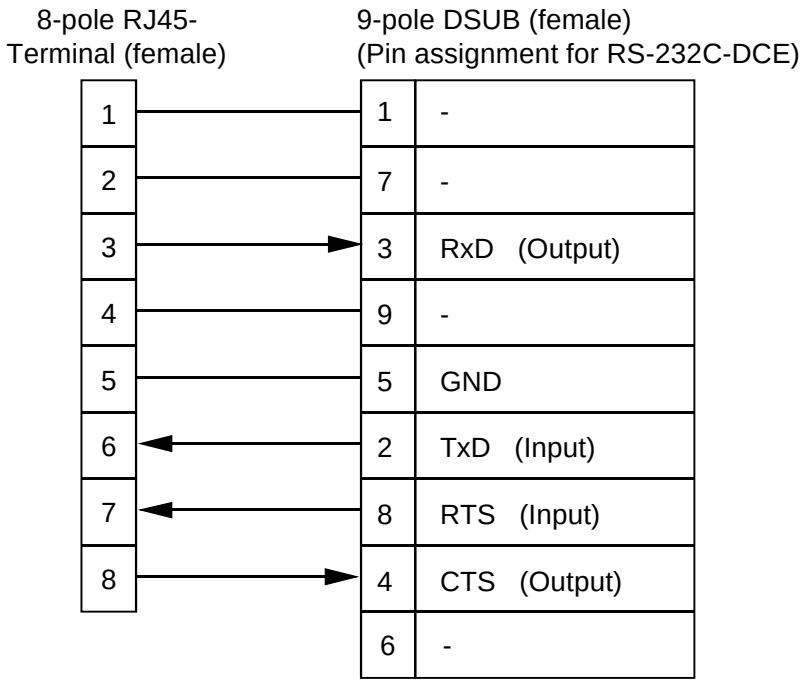
*1) The pins 4 and 8 of the sockets (P200/P230) lead to a terminal resistance which is on the piggyback. To activate the terminal resistance the signal TERM+ has to be connected to Rx/Tx+ and the signal TERM- has to be connected to the signal Rx/Tx-.

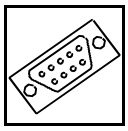
[ ] The signals shown in brackets are arranged but are not necessary for the operation of the interface.

4.4.3 Pin Assignment of the Adaptor Cable RJ45-DSUB9/Female

The adaptors RJ45-DSUB9/male (order no. C.2401.40) and RJ45-DSUB9/female (order no. C.2401.38) can once be configured indepently without tools. The connection between adaptor and CAN-CBM-SIO4 occurs by the connection cable RJ45-RJ45 (order no. C.2401.30).

The adaptor cable (order no. C.2401.30) is layed out for the operation of the CAN-CBM-SIO4 as data communication equipment (receiver, modem). The arrangement for the RS-232 interface reveals itself as follows:





Connector Assignment

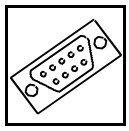
Following table shows the signal arrangement in case the adaptor RJ45-DSUB9/socket is used for the connection of the other interfaces (Signal arrangement seen from Terminal/DCE):

When connecting the TTY lines, following has to be noticed:
The descriptions (out) and (in) show only the direction of the data transmission and *not* the direction of the current. For the connection of the TTY signals the circuit layer in chapter *'Connection of the Various Serial Interfaces at DSUB9 Connector'* will be helpful.

RJ45 socket	DSUB9 socket	Signal arrangement			
		RS-422	RS-485	TTY-passiv	TTY-aktiv
1	1	-	-	-	-
6	2	Rx+ (out)	Rx/Tx+	Rx+ (out)	Rx- (out)
3	3	-	-	[I1+]	Rx+ (out)
8	4	Tx- (in)	TERM- *1)	Tx- (in)	[GNDA]
5	5	GND	GND	GND	GND
-	6	-	-	-	-
2	7	Rx- (out)	Rx/Tx-	Rx- (out)	[GNDA]
7	8	GND	GND	[I2+]	Tx+ (in)
4	9	Tx+ (in)	TERM+ *1)	Tx+ (in)	Tx- (in)

*1) The pins 4 and 8 of the sockets (P200/P230) lead to a terminal resistance which is on the piggyback. To activate the terminal resistance the signal TERM+ has to be connected to Rx/Tx+ and the signal TERM- has to be connected to the signal Rx/Tx-.

[ ] The signals shown in brackets are arranged but are not necessary for the operation of the interface.



4.4.4 Connection of the Adaptor RJ45-DSUB25 Socket

The adaptors RJ45-DSUB25/male (order no. C.2401.34) and RJ45-DSUB25/female (order no. C.2401.36) can once be configured independently without tools. The connection between adaptor and CAN-CBM-SIO4 occurs by the connection cable RJ45-RJ45 (order no. C.2401.30).

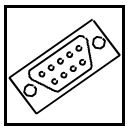
The following table shows the connector Pin Assignment, if the CAN-CBM-SIO4 should work as modem (data communication equipment) in RS-232 operation, i.e. as receiver.
The Pin Assignment of the other serial interfaces (RS-422, RS-485, TTY) results from this.

When connecting the TTY lines, following has to be noticed:
The descriptions (out) and (in) show only the direction of the data transmission and *not* the direction of the current. For the connection of the TTY signals the circuit layer in chapter *'Connection of the Various Serial Interfaces at DSUB9 Connector'* will be helpful.

Connector Pin		**Signal arrangment**				
RJ45 socket	**DSUB25 socket**	**RS-232**	**RS-422**	**RS-485**	**TTY-passive**	**TTY-active**
1	1	-	-	-	-	-
2	14	-	Rx- (out)	Rx/Tx-	Rx- (out)	[GNDA]
3	3	RxD Data Output	-	-	[I1+]	Rx+ (out)
4	16	-	Tx+ (in)	TERM+ *1)	Tx+ (in)	Tx- (in)
5	7	GND	GND	GND	GND	GND
6	2	TxD Data Input	Rx+ (out)	Rx/Tx+	Rx+ (out)	Rx- (out)
7	4	RTS Handshake Input	GND	GND	[I2+]	Tx+ (in)
8	5	CTS Handshake Output	Tx- (in)	TERM- *1)	Tx- (in)	[GNDA]

*1) The pins 4 and 8 of the sockets (P200/P230) lead to a terminal resistance which is on the piggyback. To activate the terminal resistance the signal TERM+ has to be connected to Rx/Tx+ and the signal TERM- has to be connected to the signal Rx/Tx-.

[ ] The signals shown in brackets are arranged but are not necessary for the operation of the interface.



Connector Assignment

The following table shows the connector Pin Assignment of the25 pole DSUB/male, if the CAN-CBM-SIO4 should work as terminal in RS-232 operation, i.e. as transmitter.
The Pin Assignment of the other serial interfaces (RS-422, RS-485, TTY) results from this.

Attention: This pin assignment is *not compatible* to the pin assignment of the serial interfaces of the previous table. It is *only* for the RS-232-signals compatible (DTE-DCE-connection).

Connector Pin		**Signal arrangement**				
RJ45 socket	**DSUB25 socket**	**RS-232**	**RS-422**	**RS-485**	**TTY-passive**	**TTY-active**
1	1	-	-	-	-	-
2	14	-	Tx- (out)	Rx/Tx-	Tx- (out)	[GNDA]
3	2	TxD Data Output	-	-	[I1+]	Tx+ (out)
4	16	-	Rx+ (in)	TERM+ *1)	Rx+ (in)	Rx- (in)
5	7	GND	GND	GND	GND	GND
6	3	RxD Data Input	Tx+ (out)	Rx/Tx+	Tx+ (out)	Tx- (out)
7	5	CTS Handshake Input	GND	GND	[I2+]	Rx+ (in)
8	4	RTS *2) Handshake Output	Rx- (in)	TERM- *1)	Rx- (in)	[GNDA]

*1) The pins 4 and 8 of the sockets (P200/P230) lead to a terminal resistance which is on the piggyback. To activate the terminal resistance the signal TERM+ has to be connected to Rx/Tx+ and the signal TERM- has to be connected to the signal Rx/Tx-.

[ ] The signals shown in brackets are arranged but are not necessary for the operation of the interface.

*2)... An DTR signal is needed by some modems (Data from CBM-SIO4 -> terminal). If this is the case, the DTR signal can be created by bridgeing the RTS signal in the connector on the DTR pin. With a 25-pin DSUB-connector pin 4 has to be bridged to pin 20 in this case.

4.5 Voltage Feed (X101, UEGM)

The voltage is fed by means of the UEGM-screwed connectors integrated in the case. They can be used for cables with a cross section of up to 2.5 mm².

The assignment of the connectors is the same at both sides of the case. The connectors can be used alternatively. The contact in the middle is designed for +24V and the two outer contacts are designed for GND.

Attention: It is **not** permissible to feed through the 24V supply voltage, i.e. using one side as a 24V input and the other side as a 24V output to supply other devices !

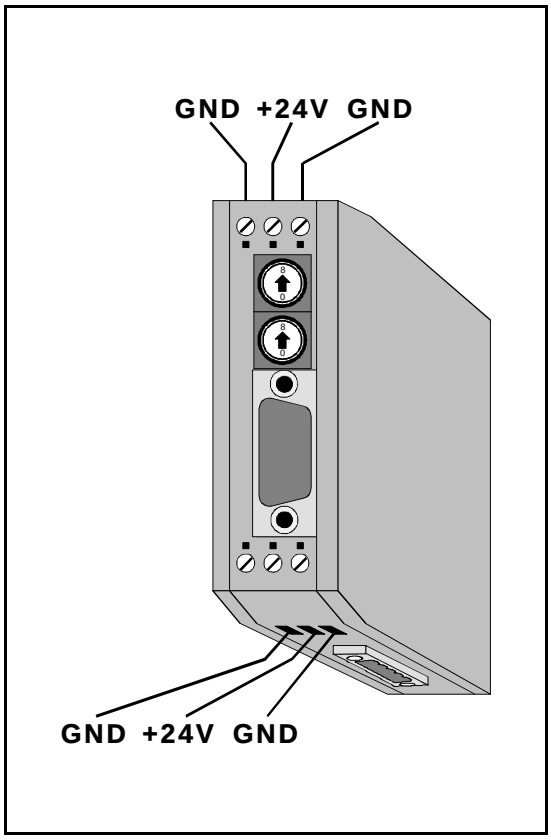


Fig. 4.5.1: Voltage feed CAN-CBM-SIO1-module and CAN-CBM-PLC/331-1-module

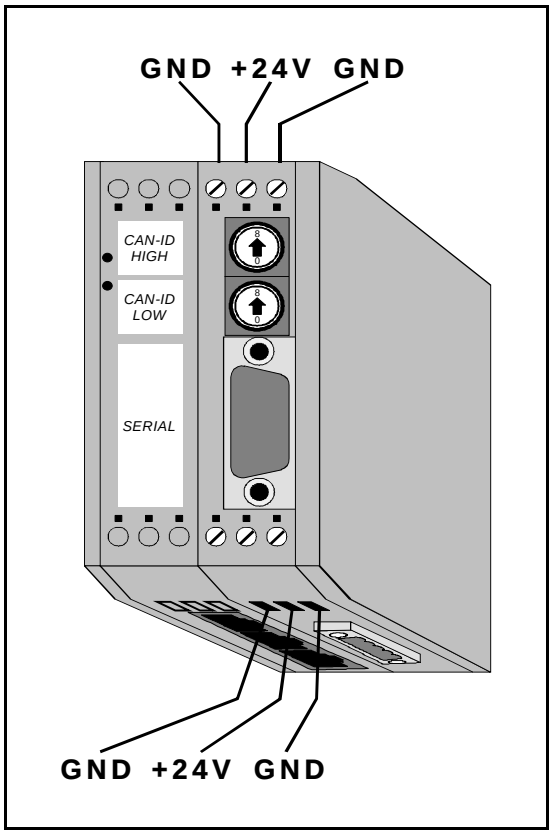


Fig. 4.5.2: Voltage feed CAN-CBM-SIO4-module

5. Configuration of the CAN-CBM-PLC/331-1/-2 Module

This chapter describes how to configure the CAN-CBM-PLC/331-1/-2 module and take it into operation by means of the CoDeSys programming environment.
The CoDeSys software is shipped with an online help which describes the various possibilities of CoDeSys.
Further information about CANopen can be found in the CANopen CiA Draft Standard 301 specification.

In order to configure the CAN-CBM-PLC/331-1/2 module you have to follow the steps below:

1. Import the various Files:

Install the Target Support Package by means of the installation program ***Install Target.exe***. Check, whether the EDS files of the desired modules are available in the subdirectory: %CoDeSys%\Targets\ESD\ESD_CAN-Module\ of the library directory. Import the desired EDS files, if required.

2. Start the CoDeSys Development Environment.

3. Configuration: Select ***New*** in the ***File*** menu. The dialog box ***Target Settings***, as shown in the figure below, appears.

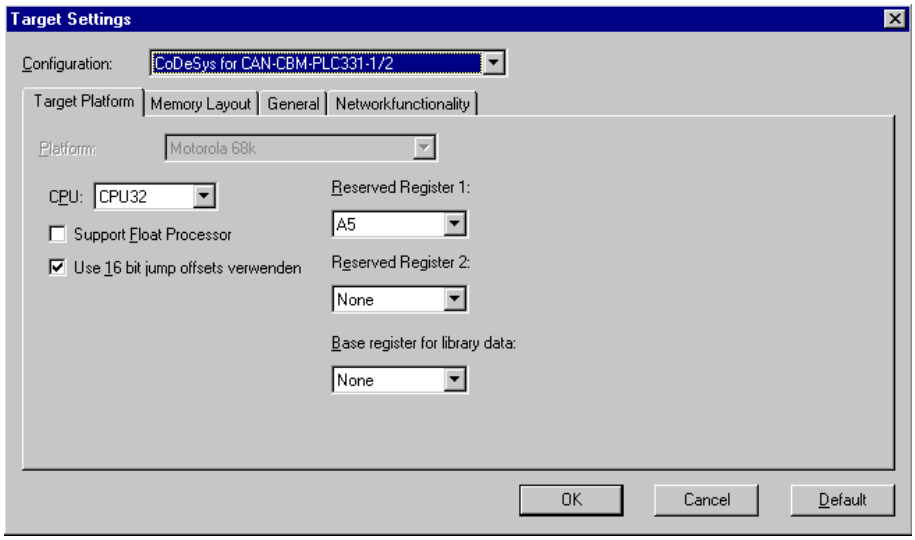


Fig.1: Settings of the target platform

The ***Configuration*** has to be set to 'CoDeSys for CAN-CBM-PLC/331-1/2'. By selecting this target the platform-specific basis configuration is loaded.

Set ***CPU*** to'CPU32'. Acknowledge by ***OK***.

Check the path names of the Compilation Files and Libraries.
Select ***Options*** in the menu ***Project***, and further ***Directories***. Check the path of the ***Libraries*** (e.g.:C:\codesys\library) and of the ***Compilation Files*** (e.g.:C:\CoDeSys)

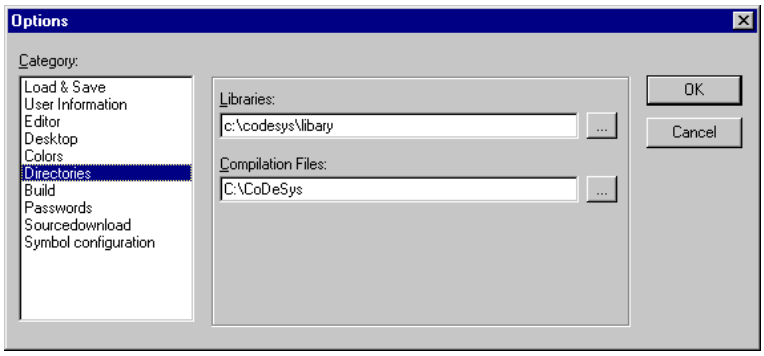


Fig.2: Check libraries and compilation files

4. New POU: When you acknowledge your selection in ***Target Platform*** with ***OK***, the dialog box ***New POU*** opens:

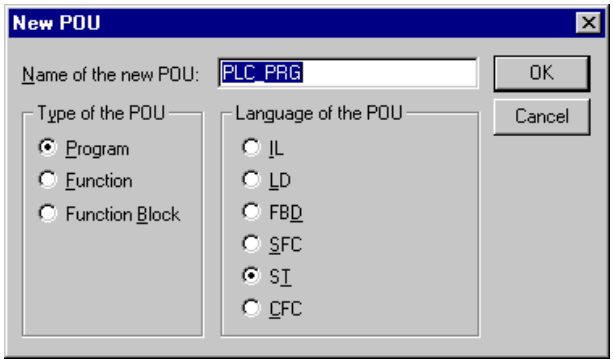


Fig.3: Dialog box ***New POU***

The PLC_PRG unit has been specially predefined and is automatically installed for every new project. It must not be deleted or renamed (does not apply for the use of task configuration). You can find further information on this in the CoDeSys online help.

Acknowledge the settings without further changes by ***OK***.

5. Selecting the CAN Master:

Change to ***Resources*** register (register, lower left screen corner).

Select menu point ***PLC Configuration***.

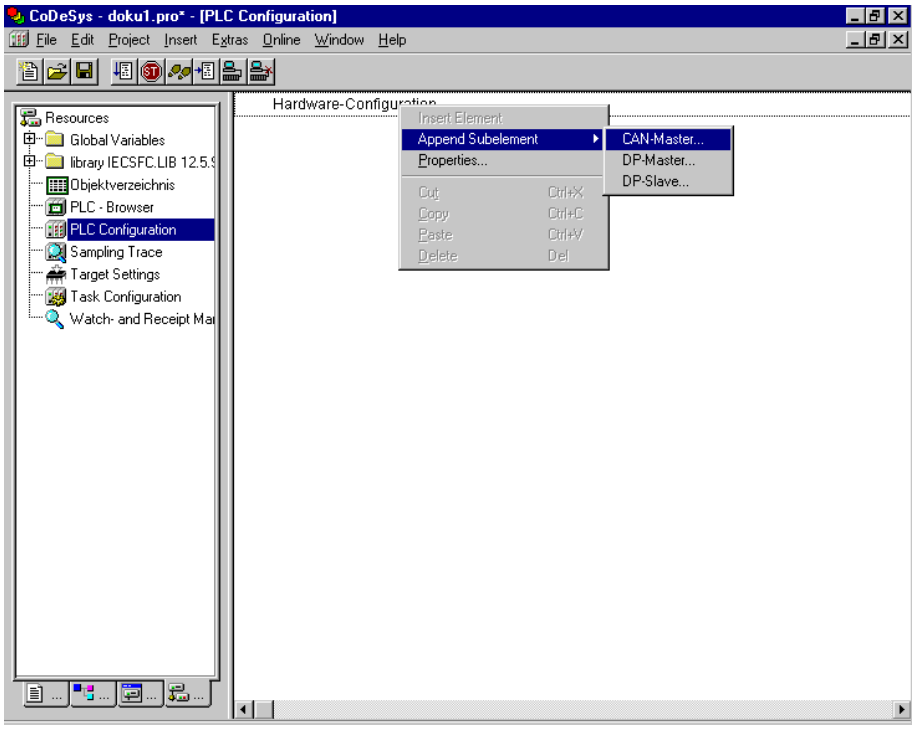


Fig.4: Select CAN-master

The ***PLC Configuration*** field appears on screen. Click the field ***Hardware-Configuration*** with the right mouse key and select the menu point ***Append Subelement*** and then ***CAN-Master***.

6. Select CAN Properties:

In the dialog box which then appears you can now specify the desired CAN properties:

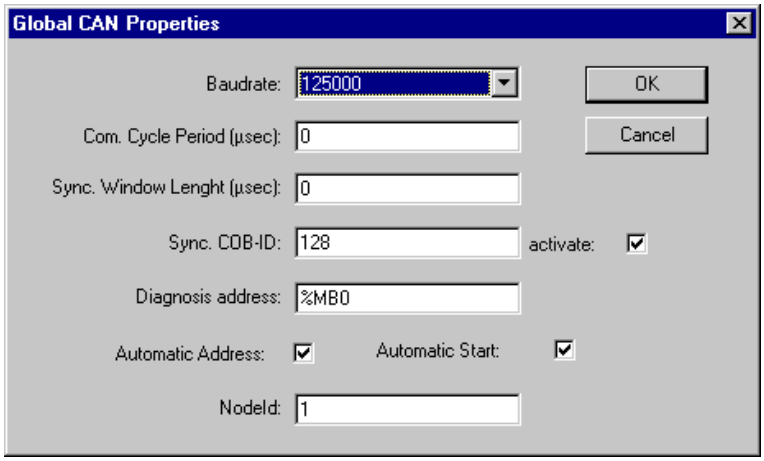


Fig.5: Set global CAN properties

The settings of the parameters listed depend on the respective application. Further information can be found in the CoDeSys online help.

***Baudrate*:** Specify the baud rate desired for transmission (here 125 kbaud).

***Com. Cycle Period*:** Cycle period for Sync. telegram, i.e. the period between the transmission of two SYNC telegrams by the SYNC master.
The ***Com. Cycle Period*** depends on slaves, bus speed and internal data processing rate.

Attention: ***Com.Cycle Period*** must be larger than ***Sync. Window Length*** to make sure that all SYNC consuming devices have received the synchronous PDOs (Process Data Objects). See also Fig.6.

Sync.Windows Length*:** Shows the time which passes from the transmission of a SYNC until all synchronous PDOs have been transmitted. Since it is smaller than the ***Com.Cycle Period the transmission of all requested data is guaranteed before a new SYNC telegram can be started. See Fig.6.

Attention! If the fields ***Com. Cycle Period*** and ***Sync.Windows Length*** have been assigned with ‘0’, no SYNC telegrams will be transmitted.

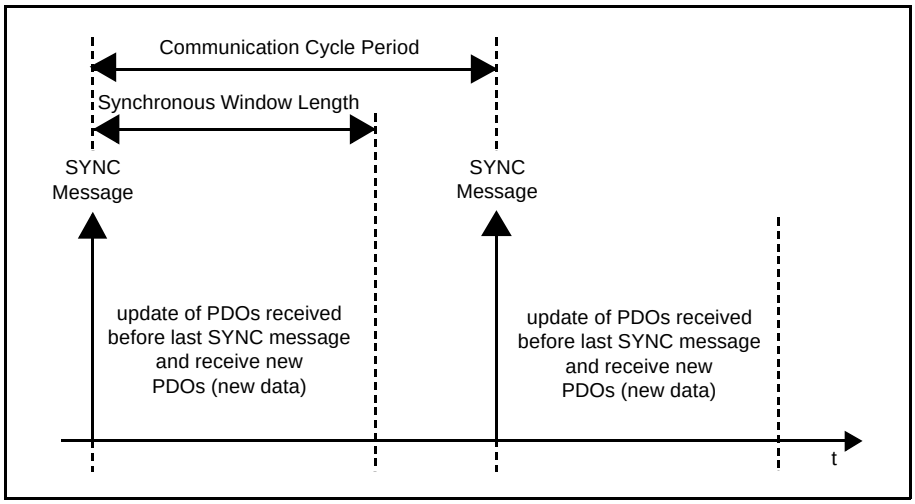


Fig.6: Bus synchronisation

***Sync. COB-ID*:** Identifier under which SYNC telegrams are transmitted and received.

***Diagnose Address*:** Here you have to specify a pointer under which the diagnose data is stored.

***Node ID*:** Identifier of the CAN-CBM-PLC/331-1/2 (between 1 and 127, decimal specification).

If you acknowledge your selection with ***OK***, the CAN-master in the ***PLC Configuration*** field is included into the configuration scheme under hardware configuration (see Fig. 7 'Append subelements').

Further information and details can be found in the CANopen specification 'CANopen CiA Draft Standard 301' chap. 9.3.1.

7. Append Subelement:

After the master has been configured the remaining CAN network is assembled and configured. In order to include further elements you have to click on the included CAN-master with the right mouse key to get to a selection of modules via menu point ***Append Subelements***.

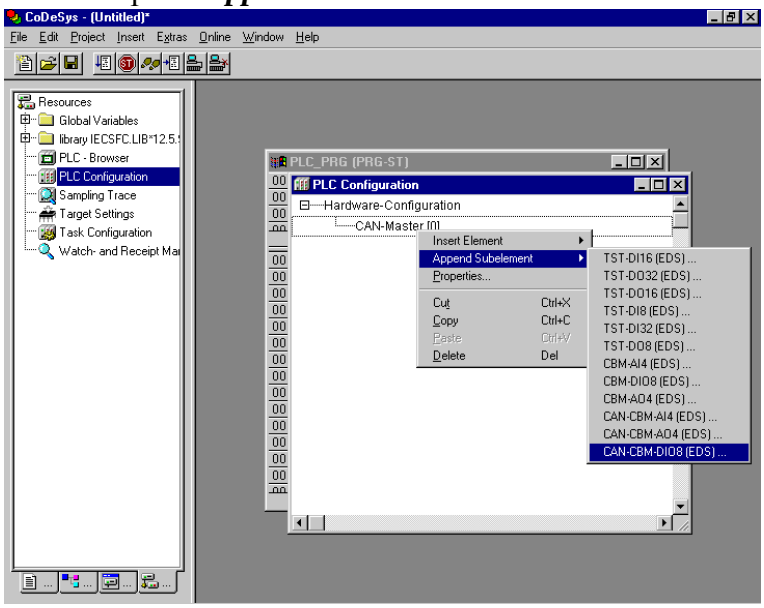


Fig.7: Append subelements

If you click the desired module (here CBM-DIO8) with the left mouse key, a dialog box (see Fig. 8) will open in which you can specify the desired properties of the device selected.

If no entry or EDS file is available for the device, you can substitute the unavailable EDS file by a TST file. TST files are EDS files configured for simple applications. The ending of the test file name (TST files) explains the respective function of the file:

TST-*xxyyzz*

trunk of name:	TST-	
following letter:	*xx*...	D,A (Digital, Analog)
following letters:	*yy*...	I,O, IO (Input, Output, In/Output)
following numbers:	*zz*...	length of the transmitted data in bits, e.g.:8, 16, 32 or 64

Example:

TST-DI8 (EDS)	digital input, 8 bits
TST-DO32 (EDS)	digital output, 32 bits

8. Basis Parameters:

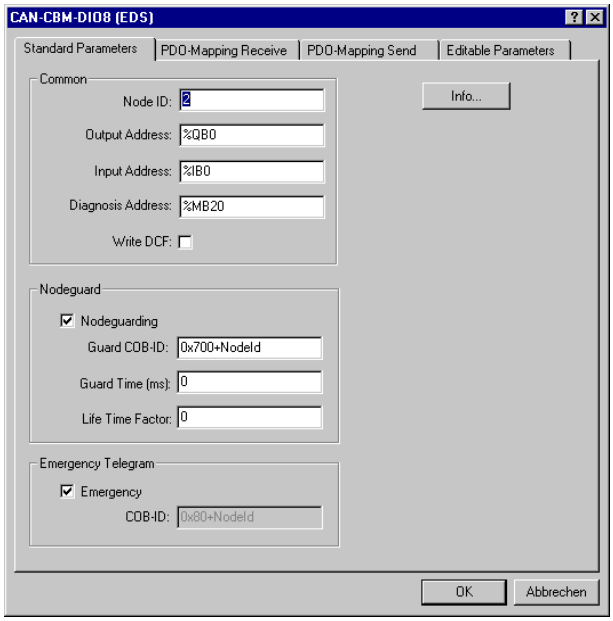


Fig.8: Set basis parameters of subelements

Specify the following parameters according to your application.
Please refer to the chapter 'Basis Parameters of a CAN Module' of the CoDeSys online help for more details about the parameters.

Node ID: Identifier of CAN-slave

Input Address: Address under which the module is accessed by the application program.

Diagnose Address: Address under which the diagnose data is stored.

Write DCF: Creating a DCF file after an EDS file has been included, if activated.

All process data in the CAN network are read or written via input and output address range of the CAN-CBM-PLC/331-1/2 module. (Via, e.g.: %IB4 ... input byte 4, %QB6 ... output byte 6).
CAN-specific data such as: identifier, RTR...are not used in the application program itself.

If the options ***Nodeguarding*** and ***Emergency Telegram*** are desired to monitor the device, activate them. Further details, also about the menu points ***PDO Mapping Receive***, ***PDO Mapping Send*** and ***Editable Parameters*** can be found in the CoDeSys online help.

Acknowledge your selection with ***OK***

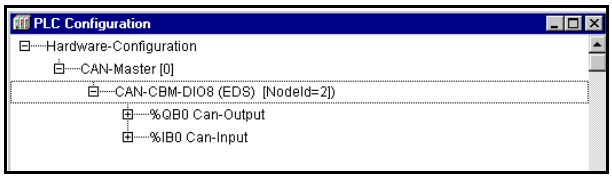


Fig.9: PLC configuration

The selected module (here CAN-CBM-DIO8) now appears in the window ***PLC Configuration*** in the configuration scheme as subelement. By clicking the preceding plus sign with the left mouse key you get more information about the respective element, such as input and output address.

9. Add further Modules:

In order to add further modules you have to repeat the steps described under 7. and 8.

Example: The module CBM-AO4 (EDS) is selected as further module as described under **7. Append Subelements**.

The dialog box properties ***CAN-CBM-AO4*** opens:

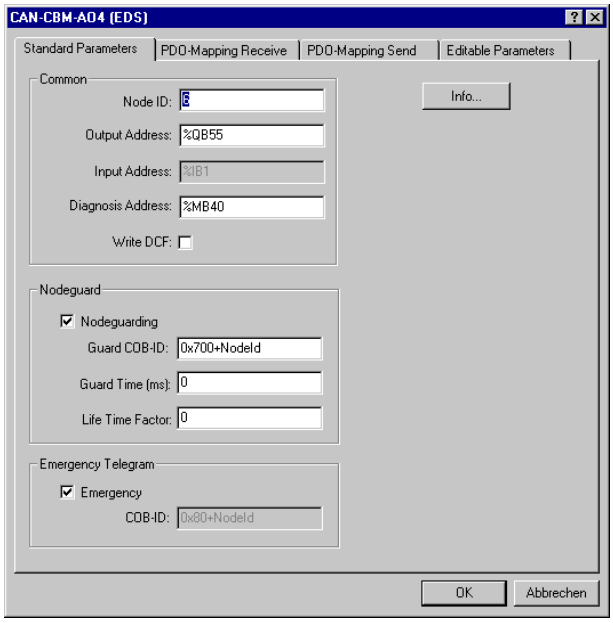


Fig.10: Example CAN-CBM-AO4

The identifier of the desired module has got the node ID = 6,
the output address: %QB55 (the output byte 55),
the diagnose address: %MB40 (the byte at the address of the pointer 40),
the guard COB-ID results from 0x700+Node ID (here 6)

The following window will open, when you acknowledge with OK:

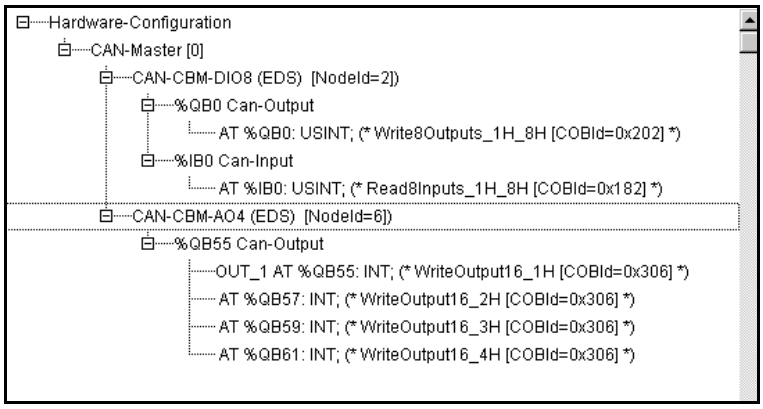


Fig.11: Configuration example

The module CAN-CBM-AO4 with node ID = 6 has now been added. By clicking the plus sign you can get further information about the configuration. The output address of the first channel is: %QB55
The following function description appears for each channel of the selected module:

Function description	*name* AT*%address*:data type;(*comment*)
Example	OUT_1 AT%QB55:INT;(*WriteOutput16_1H [COBId0x306]*)

name: If another output module is added before the CAN-CBM-AO4 module, the output address is automatically increased according to the number of outputs of the added module. In order to prevent a change for all programs in which this address appears, a name can be assigned to the global variable. By clicking AT a small input window appears. Here you can enter a name for the global variable, which is on address 55, here. This name can now be used for all programming.

AT%*address*: selected output address, %QB55

Data type: here of integer type

(*Comment*): WriteOutput16_1H: Output 16 bits on channel 1(1H), under the COB-ID = 0x306 the process data is transmitted in the CAN network.

6. Correctly Wiring Electrically Insulated CAN Networks

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

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

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

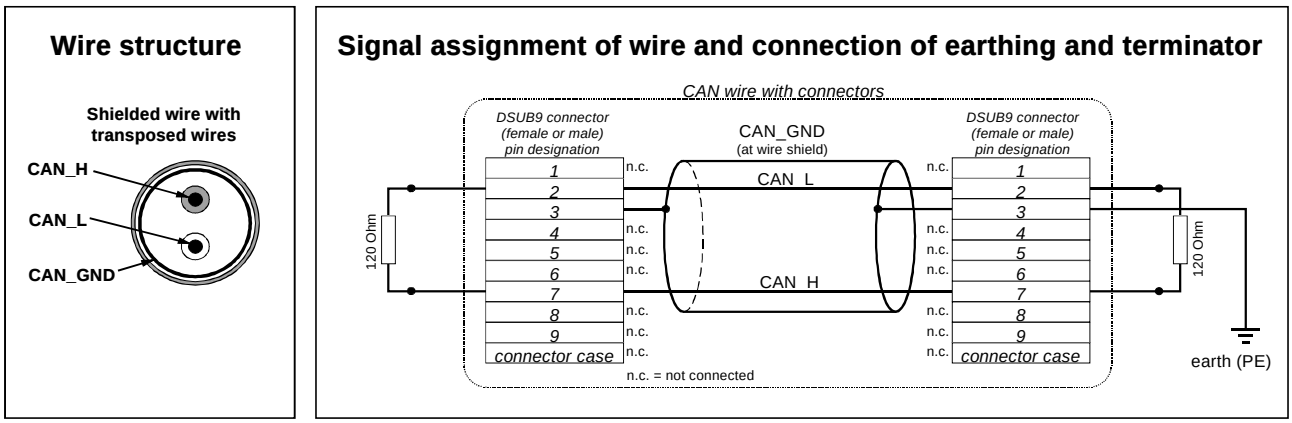


Figure: Structure and connection of wire

Cabling

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

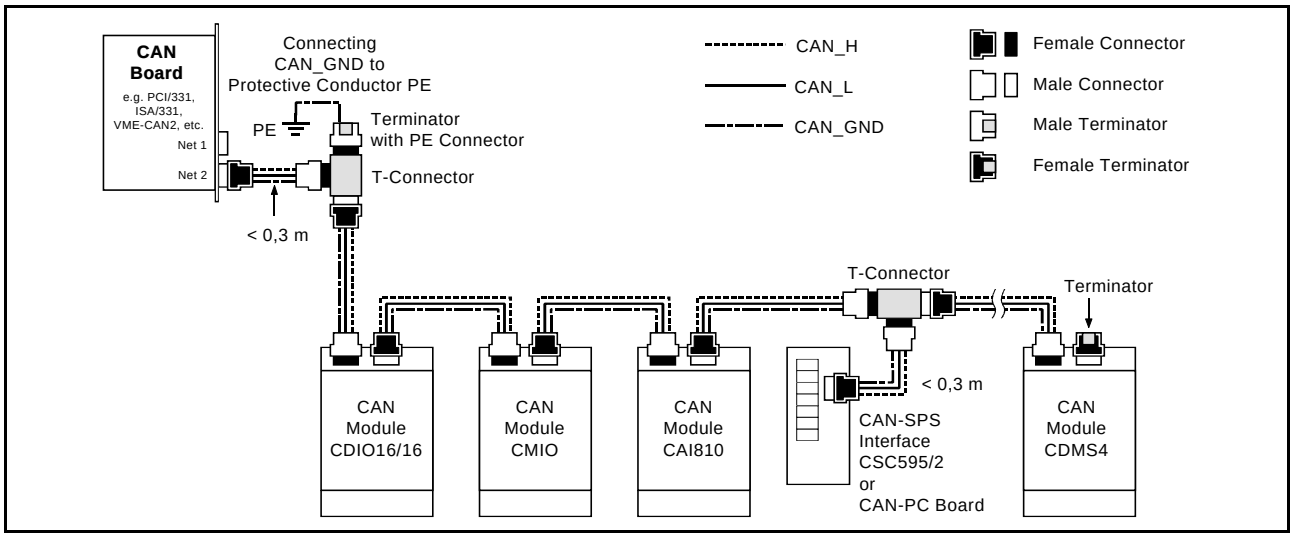


Figure: Example for correct wiring (when using single shielded wires)

Terminal Resistance

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

Earthing

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

Wire Length

- Optical couplers are delaying the CAN signals. By using fast optical couplers and testing each board at 1 Mbit/s, however, esd CAN guarantee a reachable length of 37 m at 1 Mbit/s for most esd CAN modules within a closed net without impedance disturbances like e.g. longer dead-end feeders. (Exception: CANbloc-Mini-DIO8, -AI4 and AO4 (these modules work only up to 10 m with 1 Mbit/s))

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

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

Examples for Suitable Types of Wire

Manufacturer	Type of wire
U.I. LAPP GmbH & Co. KG Schulze-Delitzsch-Straße 25 70565 Stuttgart	UNITRONIC ®-BUS LD, UNITRONIC ®-BUS FD P LD
metrofunk KABEL-UNION GmbH Postfach 410109 12111 Berlin	LiYCY 2 x 0,38 mm², LiYCY 2 x 0,5 mm², LiYCY 2 x 0,75 mm², LiYCY 2 x 1,0 mm², 1P x AWG 22 C, 1P x AWG 20 C

Manufacturer	Type of wire
Alcatel Kabelmetal Kabelkamp 20 30179 Hannover	DUE 4401, DUE 4001, DUE 4402
ConCab Kabel GmbH Äußerer Eichwald 74535 Mainhardt	1 x 2 x 0,22 mm² Best-Nr. 93022016 (UL approved)

7. Circuit Diagrams

CAN-CBM-SIO
CAN-CBM-SIO4

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

Manual of the Module-Specific Software

N O T E

The information in this document has been carefully checked and is believed to be entirely reliable. **esd** makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors that may appear in this document. **esd** reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance or design.

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

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

esd electronic system design gmbh
Vahrenwalder Str. 205
D-30165 Hannover
Germany

Tel: +49-511-372-980
Fax: +49-511-372-981-98
Email: info@esd-electronics.com
Internet: http://www.esd-electronics.com

Manual file:	I:\TEXTE\DOKU\MANUALS\CAN\CBM\SIO-331\CSIO-10S.EN6
Date of setting copy:	14.07.1999

Software version:	sio4_V1.0aE0

Changes in the chapters

The changes in the user's manual listed below affect changes in the firmware as well as changes in the description of the facts only.

Manual Rev.	Chapter	Changes versus previous version
-	-	First revision
	-	-

Technical details are subject to change without notice.

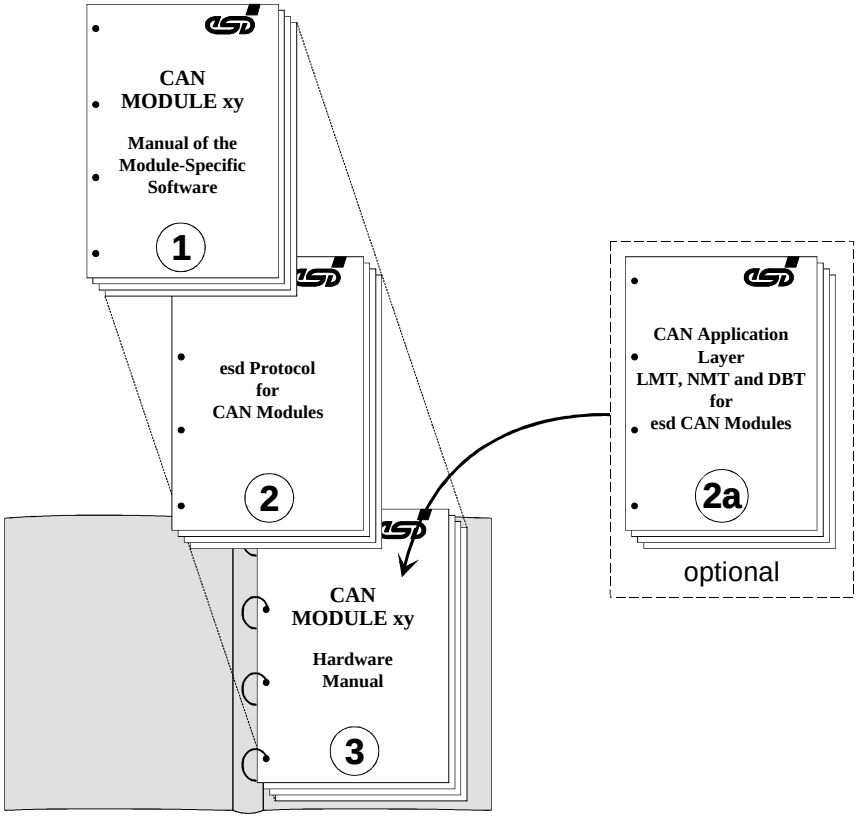
Content

Page

1. Overview

1.1 Which is Where?

The description of esd-CAN modules has been divided into three manuals, which are delivered together in one ring binder.



The first manual deals with software properties and parameters which are module-specific. This manual can therefore be used regardless of the CAN protocol you chose:

CAN-CBM-SIO CAN-CBM-SIO4 **Serial Interfaces** **Manual of the Module-Specific Software**

This manual, for example, looks into the functions of the type-specific firmware, the identifier assignment and the assignment of user parameters.

In this manual the esd-CAN modules CAN-CBM-SIO4 and CAN-CBM-SIO will be described. Both manuals are generally the same, in contrast to the CAN-CBM-SIO4 module, however, the CAN-CBM-SIO module only has got one serial interface, which is at channel 1. Due to this, it only has got two CAN identifiers or two COB-IDs.
In the following the CAN-CBM-SIO4 module will be generally described. Differences to the CAN-CBM-SIO module will be described at the adequate points.

The second manual contains general software descriptions which are valid for all esd-CAN modules operated by the same protocol.

Two different protocols are available for the modules: The esd-CAN protocol and the CMS protocol. The protocols are independent from each other and are used alternatively. Depending on the implemented protocol, therefore, one of the following two manuals is valid for the module:

The esd-CAN protocol is described in the manual:

esd Protocol for CAN Modules

The protocol allows the user to set the esd-CAN modules by means of an initialisation identifier ($700). By means of this protocol identifiers can be assigned to the modules, user parameters can be set and watchdog functions can be activated.

Alternatively, the modules can be controlled via the CMS protocol. If this protocol has been implemented, you will have to consult the manual

CAN Application Layer LMT, NMT and DBT in esd Modules

for the CMS option. This manual explains the CMS services of
the Layer Management (LMT),
the Network Management (NMT) and
the Identifier Distributor (DBT)
in esd-CAN modules.

The third manual contains the hardware description of the module. It explains general as well as module-specific characteristics of the hardware. Here you can find subjects such as installation notes and connector assignments.

CAN-CBM-SIO CAN-CBM-SIO4
CAN - RS-232, RS-422, RS-485 or TTY Interface
Hardware Manual

1.2 Default Settings

The default settings of the manual are active, when one or more of the following conditions apply:

- A default RESET had been triggered on the module via the esd-CAN protocol.
- The data of the I²C-EEPROM are not OK (e.g. EEPROM is not equipped).
- The position of the coding switches after a RESET or power-on had been set to ‘00' and had then be changed to another value.

Individual parameters can be changed without influencing the default setting of other parameters. Changes in parameters are only retained after a RESET, if they had been stored in the EEPROM.

Default values when operating the module with the esd-CAN protocol	
INIT-Id.	in all operating modes $700
Rx-identifier, Tx-identifier	RxId1, TxId1 -▸ serial channel 1 RxId2, TxId2 -▸ serial channel 2 RxId3, TxId3 -▸ serial channel 3 RxId4, TxId4 -▸ serial channel 4 RxId5, TxId5 -▸ serial channel 5 The default values of the identifiers correspond to the settings via the coding switches. Please refer to the hardware manual for a detailed description of the settings. (*)
Module No.	= setting of the coding switches
CAN-bit rate	= 125 kbit/s

(*) Please note that the CAN-CBM-SIO4 module covers five Tx- and five Rx-identifiers. Therefore, the identifiers of the following modules have to be selected with an offset of at least +10, because otherwise the identifiers would clash!

Table 1.2.1: Default settings of the module when operated with the esd protocol

Attention: The terminal interface (at DSUB9) on the CAN-CBM-SIO4 module has been assigned with Rx-identifier RxId**5** and Tx-identifier TxId**5**. On the CAN-CBM-SIO module with only one serial interface, however, the terminal interface has been assigned with Rx-identifier RxId**1** and Tx-identifier TxId**1**.

Default values when operating the module with the CAL protocol	
Manufacturer name	ASCII 'esd_han'
Product name	has not been defined yet
Module-ID	= setting of coding switches
Module name	has not been defined yet
CAN-bit rate	125 KBIT/s
After a default RESET a *Configuration Download* to the module via the NMT protocol is absolutely necessary!	

Table 1.2.2: Default settings of the module when operated with CAL

Default values of user parameters (regardless of protocol used)		
First Tx-activate delay	10.000 msec	
CAN-Tx mode (all channels)	$1411, i.e.	'MinChar', 'MaxChar' = 1, 'Inhibit-Time' = 20 ms
Serial mode (all channels)	$2273, ie.	CTS* active (terminal interface: no CTS), 9600 baud, 2 stop bit (terminal interface: 1 stop bit) no parity, 8 bit/character

Table 1.2.3: Default settings of parameters of the module

Explanations of the terms for the user parameters in table 1.2.3:

First Tx-activate delay... Delay after a RESET before the module starts transmitting messages to the CAN or the serial interfaces.

CAN-Tx mode... 'MinChar' and 'MaxChar' determine the minimum and maximum number of data bytes which are to be transmitted within the CAN frame on the CAN-bus. ‘Inhibit time’ shows the delay of the CAN controller between the last successful transmission on the CAN-bus and the start of the following transmissions.

Serial mode... By means of serial mode the serial interfaces are set.

2. Description of the Data Transfer

The serial data are buffered between the CAN-bus and the serial interfaces in both data directions and by means of a 256 bytes sized toroidal-core store for each channel. The data being received first are also transmitted again first.

If the toroidal-core store is full and further data is received, it will be lost. Therefore, it is important to match the transmission rates of CAN-bus and serial interfaces with each other.

2.1 Serial Interfaces Transmit Data

The amount of CAN-bus data to be stored in the toroidal-core store per interval, has to be smaller than the amount of data transmitted by the serial interfaces.

The amount of received data per interval depends on the amount of data bytes transmitted, the frequency of transmissions, the bit rate of the CAN-bus and the assigned bus enabling of the CAN-bus.

If data is lost during operation, the intervals between transmission are to be prolonged or/and the amount of transmitted bytes has to be decreased.

The data is transmitted to the module by the CAN-bus via the Rx-identifiers or via COBs 1 to 4. The user is free to select the amount of bytes to be transmitted.

esd Prot.	CAL	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
RxId	COB								
RxId 1	COB 1	Data for serial interface channel 1							
RxId 2	COB 2	Data for serial interface channel 2							
RxId 3	COB 3	Data for serial interface channel 3							
RxId 4	COB 4	Data for serial interface channel 4							
RxId 5	COB 5	Data for serial interface channel 5							

Table 2.1.1: Receiving the data to be transmitted via Rx-identifiers or COB 1 to 5 (CAN-CBM-SIO4)

Attention:
The terminal interface (at DSUB9) on the CAN-CBM-SIO4 module has been assigned with Rx-identifier RxId**5** and Tx-identifier TxId**5**. On the CAN-CBM-SIO module with only one serial interface, however, the terminal interface has been assigned with Rx-identifier RxId**1** and Tx-identifier TxId**1**.

Data Transfer

When operating with the esd protocol, the data output on the serial interfaces can be stopped by the supervisor command 'Suspend Module'. All four channels are stopped simultaneously. By means of the command 'Continue' the channels are started again simultaneously.

If the module is in 'Suspended' status, data being received from the CAN-bus and the serial interfaces will be stored in the toroidal-core store.

The data transfer from CAN-bus to serial interfaces principally has the following chronological course:

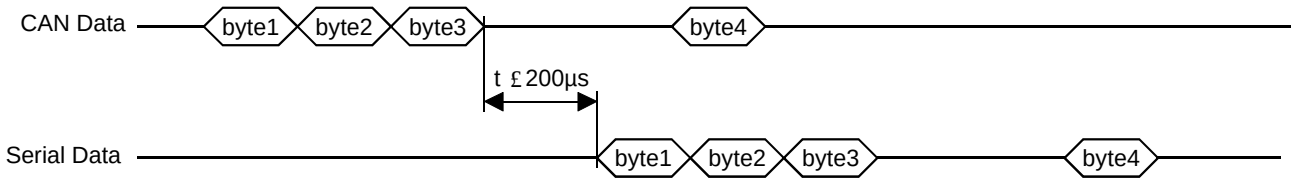


Fig. 2.1.1: Data transfer CAN-bus -> serial interface

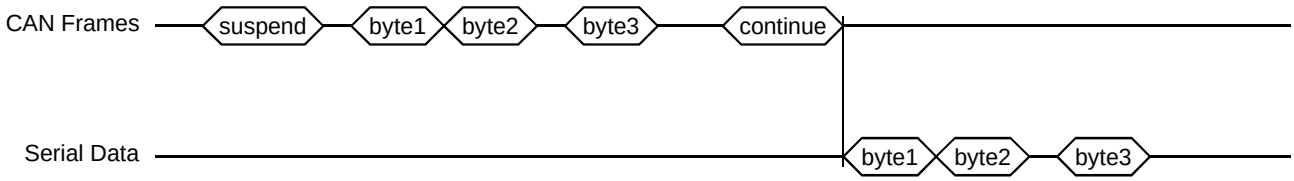


Fig. 2.1.2: Data transfer CAN-bus -> serial interface controlled by means of 'Suspend' and 'Continue'

2.2 Serial Interfaces Receive Data

The amount of serial interface data which is stored in the toroidal-core store per interval has to be smaller than the amount of data transmitted from the CAN-bus.

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

By means of parameters 'Inhibit-Time', 'MaxChar' and 'MinChar' the module offers the possibility to influence the last two factors. These parameters have been combined in the user parameter 'CAN-Tx-Mode'.

Via 'Inhibit-Time' the delay between two transmissions on the CAN-bus is determined.

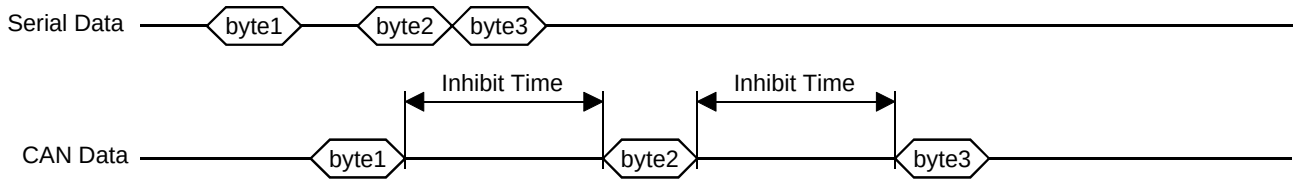


Fig. 2.2.1: Function of 'Inhibit-Time'

'MinChar' and 'MaxChar' determine the minimum and maximum number of data bytes which are to be transmitted on the CAN-bus within a CAN frame.

The data is transmitted by the module via Tx-identifiers or COBs 5 to 8 on the CAN-bus.

esd Prot.	CAL	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
TxId	COB								
TxId 1	COB 6	Data of serial interface channel 1							
TxId 2	COB 7	Data of serial interface channel 2							
TxId 3	COB 8	Data of serial interface channel 3							
TxId 4	COB 9	Data of serial interface channel 4							
TxId 5	COB 10	Data of serial interface channel 5							

Table 2.2.1: Transmission of data received on the serial interfaces via Tx-identifiers or COBs (CAN-CBM-SIO4)

The CAN-CBM-SIO module has only one Tx-identifier TxId1 or the COB 1.

3. User Parameters of the CAN-CBM-SIO and CAN-CBM-SIO4 Modules

By means of the user parameters the parameters for the serial interfaces and the parameters ‘Inhibit Time’, ‘MaxChar’ and ‘MinChar’ (CAN-Tx mode) for controlling the CAN-bus transmission rate are specified on the module.

If the module is operated with the esd protocol, the user parameters will be specified by the command 'Set User Parameters' ($86) on bytes 5 and 6 of the INIT-Id ($700).
All user parameters always have to be transmitted as 16-bit value with byte 5 as MSB!

If the CMS protocol has been implemented, the user parameters will be set via a configuration download (NMT).

The following table gives an overview of the user parameters of the CAN-CBM-SIO4 module. Only parameters $00...$02 are required for the CAN-CBM-SIO module.

User parameter No.	Parameter	Value range	Default settings
$00	First Tx-activate delay	$0000...$FFFF (0...65535 msec)	10.000 msec
$01	CAN-Tx mode channel 1	$0011...$FF88	$1411
$02	Serial mode channel 1	$0000...$88FF	$2273
$03	reserved	-	-
$04...$08	not assigned	-	-
$09	CAN-Tx mode channel 2	$0011...$FF88	$1411
$0A	Serial mode channel 2	$0000...$88FF	$2273
$0B	reserved	-	-
$0C...$10	not assigned	-	-
$11	CAN-Tx mode channel 3	$0011...$FF88	$1411
$12	Serial mode channel 3	$0000...$88FF	$2273
$13	reserved	-	-
$14...$18	not assigned	-	-
$19	CAN-Tx mode channel 4	$0011...$FF88	$1411
$1A	Serial mode channel 4	$0000...$88FF	$2273
$1B	reserved	-	-
$14...$20	not assigned	-	-
$21	CAN-Tx mode channel 5	$0011...$FF88	$1411
$22	Serial mode channel 5	$0000...$88FF	$2273
$23	reserved	-	-

Table 3.1.1: User parameters of the CAN-CBM-SIO/CBM-SIO4 module

3.1 First Tx-activate Delay (Parameter 0)

Parameter 0 specifies the delay before the module starts transmitting data to the CAN-bus and the serial interfaces after a RESET.

This delay is to secure that all modules operate stable on the CAN-bus before the module starts transmitting.

User parameter No. (= sub-command No.)	Parameter	Value range	Default setting
$00	First Tx-activate delay	$0000...$FFFF (0...65535 msec)	10.000 msec

Table 3.1.2: User parameter 0

3.2 CAN-Tx-Mode (Parameters 1, 9, 11, 19, 21)

By means of these parameters the 'Inhibit-Time' and parameters 'MaxChar' and 'MinChar' are specified.

Via 'Inhibit-Time' the delay between two transmissions on the CAN-bus is determined.

'MinChar' and 'MaxChar' determine the maximum and minimum amount of data bytes which are to be transmitted on the CAN-bus within a CAN frame. Starting from software-rev. '17.7e' the function os parameter ‘MinChar’ has been extended. A truncation status can now also be specified here.

User parameter No. (sub-command No.)	Parameter	Value range	Default setting
$01	CAN-Tx-mode chan. 1	$0011..$FF88	$1411
$09	CAN-Tx-mode chan. 2		
$11	CAN-Tx-mode chan. 3		
$19	CAN-Tx-mode chan. 4		
$21	CAN-Tx-mode chan. 5		

Table 3.2.1: User parameters CAN-Tx-mode

The two bytes of parameter CAN-Tx-mode are structured as follows:

	CAN-Tx-mode		
	Byte 5 of INIT-Id $700	Byte 6 of INIT-Id $700	
Parameter	Inhibit-Time	MaxChar	MinChar
Value range	$00...$FF	$1...$8	$1...$8

Table 3.2.2: Structure of parameter CAN-Tx-mode

Inhibit-Time..... The parameter Inhibit-Time specifies the time the CAN-controller waits after the last successful transmission of CAN-bus data, before it starts a new transmission. The entry is made in [ms].
The default setting is $14 (= 20 ms).

MinChar, MaxChar... In these two parameters the number of bytes from which a transmission to the CAN-bus is to be started, and the maximum amount of data bytes to be transmitted in a CAN frame are specified. The default setting is $11, i.e. each received byte is transmitted individually in one frame.

MinChar is only determined as described above, if values between $1...$8 are specified.
Values between $9 and $F change the functionality of the parameter: In this case the local software evaluates the entries of flags 'Carriage Return' ($0D) or 'Line Feed' ($0A) as transmission statuses. If one or both characters will be recognized, the data having been received by the serial interface will be transmitted to the CAN-bus. If eight bytes have been received before the flags have been received, the transmission will automatically start.
For the transmission you can also choose whether the flags are to be transmitted together with the data on the CAN-bus or not. The following table shows the various options:

MinChar [HEX]	Trans. status	Flag transmission to CAN	Comments
1...8	-	-	At least 1 to 8 bytes are always transmitted. Parameter MaxChar is also evaluated.
9	< Cr >	with < Cr >	Data is transmitted, if < Cr > had been received. < Cr > is also transmitted.
A	< Lf >	with < Lf >	As above, but with < Lf >.
B	< Cr >	without < Cr >	As above, but < Cr > is not transmitted.
C	< Lf >	without < Lf >	As above, but without < Lf >.
D	< Cr >	without < Cr > and without <Lf >	< Cr > and < Lf > can be at the end of the message. The data is transmitted after < Cr > had been received. Neither < Cr > nor < Lf > are transmitted with the data.
E	< Lf >	without < Cr > and without < Lf >	As above, but with < Lf >.
F	-	-	reserved

Table 3.2.3: Selection of transmission status via parameter MinChar

Examples:

1. For MinChar value $B has been selected. Via the serial interface the data 'abcd<Cr>' are received. The CAN-bus would transmit the data ‘abcd’ after having received <Cr>.

2. For MinChar value $E has been selected. Via the serial interface the data ‘abcd <Cr> <Lf>’ are received. The CAN-bus would transmit the data ‘abcd’ after having received <Lf>.

3.3 Serial Mode (Parameters 2, A, 12, 1A, 22)

The 'Serial Mode' user parameters set the bit rate, the stop bits, the number of bits/character and the CTS* locking and determine the parity evaluation of the serial channels.

User parameter No. (sub-command No.)	Parameter	Value range	Default setting
$02	Serial mode channel 1	$0000..$88FF	$2273
$0A	Serial mode channel 2		
$12	Serial mode channel 3		
$1A	Serial mode channel 4		
$22	Serial mode channel 5		

Table 3.3.1: User parameter Serial Mode

The two bytes of the parameter Serial Mode are structured as follows:

	Serial Mode		
	Byte 5 of INIT-Id $700		Byte 6 of INIT-Id $700
Parameter	Rx-Baudrate	Tx-Baudrate	Mode
Value range	$0...$8	$0...$8	$00...$7F

Table 3.3.2: Structure of parameter Serial Mode

Rx-bit rate,
Tx-bit rate... 4 bits each determine the bit rate with which data is transmitted (Tx) or received (Rx) on the serial interfaces. The default setting for Tx- and Rx-bit rate is 9600 KBIT/s. The physically attainable bit rate is limited by the hardware to a maximum of 38.4 kbit/s, when at the same time using all four channels.

Parameter Rx-(Tx) bit rate [HEX]	Bit rate [Bit/s]
0	38400
1	19200
2	**9600**
3	4800
4	2400
5	1200
6	600
7	300
8	150
9	7200
A	14400
B	28800
C	(57600)
D	(115200)
E	(2304000)
F	(76800)

Table 3.3.3: Setting the bit rate of the serial interfaces

Mode..... The bits of parameter Mode are assigned with the following functions:

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

Table 3.3.4: Assignment of parameter Mode

Explanations of bits of parameter Mode

RTS-Mode..... Via this bit the RTS-modem mode for RS485-interfaces can be selected.

RTS-mode	Evaluation
0	RTS on Rx (default setting)
1	RTS-modem (RS485)

Table 3.3.5: Evaluation of RTS-mode bits

CTS* enable... Via this bit the CTS*-function of the serial controllers is enabled.
If the bit is '0', the CTS* signal will not be evaluated and the controller transmits the available data at once (provided the module is not in ‘Suspend’ status).
The evaluation of bits can be taken from the following table.

CTS* enable bit	CTS* -evaluation
0	CTS* -input is ignored (always at terminal interface)
1	CTS* -input is evaluated: hardware handshake (default setting for channel 1...4)

Table 3.3.6: Evaluation of 'CTS* enable' bits

The RS232 driver used activate the CTS* signal automatically, if the CTS* line of the serial interface is not connected.

At the terminal interface (DSUB9) the CTS input is not evaluated. The CTS* enable bit is insignificant for this interface, therefore.

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

Stop bit	Number of stop bits
0	1 stop bit (always at terminal interface)
1	2 stop bits (default setting for channel 1...4)

Table 3.3.7: Number of stop bits

The terminal interface (DSUB9) always operates with only one stop bit. The setting ‘1', therefore, is insignificant for this interface.

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

Parity mode		Evaluation
Bit 4	Bit 3	
0	0	parity evaluation when receiving data and transmitting the parity bit
0	1	reserved
1	0	no parity evaluation, no parity transmission (default setting)
1	1	reserved

Table 3.3.8: Parity evaluation

Parity Type... The polarity of the parity bit is determined by the parameter bit ‘Parity Type’:

Parity Type	Polarity
0	'even' (default setting)
1	'odd'

Table 3.3.9: Setting the polarity

Bits per Character..... Via these two bits the number of bits/character is selected:

Bits per Character		Number of transmitted bits
Bit 1	Bit 0	
0	0	5 (not at terminal interface)
0	1	6 (not at terminal interface)
1	0	7
1	1	8 (default setting)

Table 3.3.10: Number of bits/character

For the terminal interface only the following combinations of ‘Bits per Character’ and ‘Parity Mode’ are permissible:

7 data bits +	1 parity bit	
8 data bits	no parity	(default setting)
8 data bits +	1 parity bit	

4. Examples

In this chapter the operation and initialization of a module which is operated with the ***esd-CAN-Protocol*** will be explained by means of some examples.

4.1 Operation with Default Parameters

4.1.1 Basic Conditions, Objective

A device is to be connected to channel 1 of the CAN-CBM-SIO4 which corresponds in parameters to the default setting of the CAN-CBM-SIO4:

- 9600 Baud
- 8 Bit/Character
- 2 Stop-Bits
- no Parity

The module has not been initialized yet.

The desired Tx-identifier for transmitting data to the serial interface is to be $15E. The Rx-identifier is to be $15F.

4.1.2 Procedure

4.1.2.1 Set Identifiers

The identifier is determined, for example, by the coding switches. It corresponds to the tenfold of the value set at the coding switches plus the identifier offset (note: The selection of any value is therefore not possible via the coding switches. If you wish to do so, the setting has to be made by means of the esd-CAN protocol).

With the specification for the identifier, given above, the coding switches have to be set to a value corresponding to $15E/$A.

$15E / $A = $23

At coding switch ‘HIGH’ the value ‘2’, and at coding switch ‘LOW’ the value ‘3’ is now set.

The offset results from the desired serial channel: TxId-offset = 0, RxId-offset=1

The position and function of the coding switches is described in the hardware manual of the module.

4.1.2.2 Transmitting the Data to the Serial Interface

Via Rx-identifier RxId1 data is transmitted to the module. The number of data bytes transmitted can be between 0 and 8. In this example the following 5 bytes are to be transmitted:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5
$11	$22	$33	$44	$55

On the CAN-bus the bytes are transmitted on the Rx-identifier as follows:

RxId1	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$15F	$11	$22	$33	$44	$55	-	-	-

Table 4.1.2: Transmission of data to be transmitted via Rx-identifier RxId1

Bytes 6...8 are not required.

4.1.2.3 Receiving Data from the Serial Interface

Via Tx-identifier TxId1 the data received by the serial interface is sent from the module to the CAN-bus. The module is operated with the default parameters and therefore each byte received is immediately transmitted. The delay between initiating the individual transmissions by the CAN-controller is 20 ms.

In this example the device connected at channel 1 is to transmit a block of 8 bytes with the following contents:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$FF	$EE	$DD	$CC	$BB	$AA	$99	$88

The module now transmits the individual bytes of this block on the Tx-identifier with delays of about 20 ms. If the CAN-bus is occupied with messages of a higher priority, the delay between the transmissions can be longer.

TxId1	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$15E	$FF	-	-	-	-	-	-	-

:
t ≥ 20ms
:

TxId1	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$15E	$EE	-	-	-	-	-	-	-

:
t ≥ 20ms
:

TxId1	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$15E	$DD	-	-	-	-	-	-	-

:
...
:
t ≥ 20ms
:

TxId1	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$15E	$88	-	-	-	-	-	-	-

Table 4.1.3: Transmitting received serial data to the CAN-bus

Bytes 2...8 of the individual Tx-transfers are not transmitted, because value ‘1' is set for ‘MaxChar’ in default setting.

4.2 Changing the Bit Rate

The bit rates of channel 1 are to be increased from 9600 baud (default setting) to 19200 baud for receive and transmission data.

The module No. of the CAN-CBM-SIO4 corresponds to the default setting of the coding switch setting and is therefore $23.

The bit rate is changed in the cells 'Rx-Baudrate' and 'Tx-Baudrate' of user parameter 'Serial-Mode'. These two cells are assigned to the first byte of the user parameter (at transmission = byte 5). For channel 1 this parameter is selected by means of sub-command $02.

The assignment of bit rate 19200 results in the values ‘$1' for each nibble of byte 5 of the user parameter.

Byte 6 of the parameter (Serial-Mode) should not be changed, has to be transmitted as well, however, as have all user parameters! Therefore, the default value $73 is entered here.

CAN-Id	Byte 1 (command)	Byte 2 (sub-command)	Byte 3 (always $00)	Byte 4 (module No.)	Byte 5 (bit rate)	Byte 6 (serial mode)
$700 =INIT-Id	$86	$02	$00	$23	$11	$73

Table 4.2.1: Changing the bit rates of serial channel 1

Byte 7 and 8 are not required for this command.

The changed bit rate is always active immediately after the user parameter has been received.

esd CAN Protocol
for
CAN-CBM-SIO1/4

Manual File:	I:\texte\Doku\MANUALS\CAN\Cbm\SIO-331\Englisch\ESD_SIO1.EN6
Date of Print:	11.04.2001

Described Software Revision:	
CAN kernel :	from revision '1.f' (HEX)
esd protocol :	from revision '0' (HEX)
Module specific implementation:	refer to manual of the module specific software

N O T E

The information in this document has been carefully checked and is believed to be entirely reliable. **esd** makes no warranty of any kind with regard to the material in this document, and assumes no responsibility for any errors that may appear in this document. **esd** reserves the right to make changes without notice to this, or any of its products, to improve reliability, performance or design.

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

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

esd electronic system design gmbh
Vahrenwalder Str. 205
30165 Hannover
Germany

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

USA / Canada
7667 W. Sample Road
Suite 127
Coral Springs, FL 33065
USA

Phone: +1-800-504-9856
Fax: +1-800-288-8235
E-mail: sales@esd-electronics.com

Content

Page

1. Introduction

1.1 Notes to this Manual

This manual describes the '*esd*-CAN protocol' for *esd*-CAN modules, here specially for the CAN-CBM-SIO modules. By this protocol it is possible to set the CAN parameters of the modules, as for example, the Rx and Tx identifiers or the baudrate. Apart from the CAN parameters it is also possible to set and play back module-specific parameters (user parameters) with the help of the protocol.

In the chapter 'Overview of the implemented commands and parameters' all commands supported at the moment are shown in a tabular summary.

You can find detailed information about the module-specific User-Parameters in the description of the module-specific software.

1.2 Specification of the *esd* Protocol

On the basis of its problem-oriented structure the *esd* protocol cannot be categorized clearly into a layer of the ISO layer model: It offers services which range from fundamental functions, as e.g., the inquiry of the status of the CAN hardware controller up to application-specific adjustments, as, e.g., the setting of so-called 'user parameters'.

The *esd* protocol offers functions which are comparable, e.g., with the LMT (layer management) in the CAL (CAN application layer). Because the identifier allocation occurs by the *esd* protocol, too, its functionalities are partly similar to those of the DBT (identifier distributor) in the CAL, as well.

But the *esd* protocol is operated totally independently and has got no interface to the CAL!

1.3 General Notes on Data Transmission

In the following descriptions the transmission direction of data is looked at, if not other wisely stated, from the module. The module receives data on the 'Rx identifier' and transmits data on the 'Tx identifier'.

The data bytes are counted from 1 to 8 and are always transmitted in the order byte 1...byte 8. The number of transmitted bytes can vary from 0 to 8. The data transmission has to start with byte 1 and progress in continuous order (e.g., byte 1, byte 2, byte 3 - not possible, e.g., byte 1, byte 7, byte 8).

Generally only those bytes are overwritten which are received by the module. The data of not recorded bytes remain unchanged.

1.4 General Hardware Functions

To use the esd CAN protocol at the CAN modules at each module the following hardware circuits are necessary:

CAN controller SJA1000 (or compatible)

The controller has a internal RAM, that is used as a working memory. In this RAM the dynamical parameters are stored. The RAM is deleted with each RESET.

I²C EEPROM

The I²C EEPROM is used for storing the configuration parameters. The data will remain stored in power off condition or after a RESET.

Coding switch

Via the coding switch, e.g. the default value of the module no. is set. Valid module no's are from $00 up to $FE. The value $FF is reserved for the 'Advanced Configuration'.

Further descriptions can be read in the hardware manual of the module.

1.5 Parameters after a RESET

The module offers various possibilities to trigger a RESET:

A power-on RESET and a RESET by the general command 'RESET module' reset the local components.
Furthermore the module is able to trigger a RESET independently when the hardware watchdog has expired. This RESET also resets the local components without changing the stored parameters.

The listed RESETs only change the parameters of the module filed in the I²C EEPROM, if the conditions apply which are listed in the table below.

The module also supports the command 'default RESET'. By this command a local RESET is triggered and the parameters of the module are <u>always</u> overwritten with the default parameters.

The used parameters with which the module operates after a power-on RESET or a RESET by the general command 'RESET module', depend mainly on three factors:

The switch position of the coding switches, the availability of the I²C EEPROM data and the module number (parameter 'module no.') stored in the I²C EEPROM.

The following table should give an overview of the resulting parameters. It does not contain the 'default RESET', because this does always lead to the activation of the default parameters.

	Actual position of the coding switches after RESET	I²C EEPROM status	I²C EEPROM module no.	CAN identifier (CAN Id.), parameter, module no.
1a	$00	x	x	CAN Id. = f{Coding switches} Parameter = default I²C EEPROM mod no. = in this case the previous module no. is deleted at power-on active mod no. = $00
1b	≠ $00	ERROR	x	CAN Id. = f{Coding switches} Parameter = default I²C EEPROM mod no. =$00 active mod no. = Coding switch no.
2	≠ $00	OK	$00	CAN Id. = f{Coding switches} Parameter = default I²C EEPROM mod no. = $00 active mod no. = Coding switch no.
3	≠ $00	OK	≠ $00	CAN Id. = f{I²C EEPROM} Parameter = f{I²C EEPROM} active mod no. = I²C EEPROM mod no.

(x) This value or status is of no importance in this case.

Table 1.5.1: Parameter after a RESET

Explanations to Table 1.5.1:

Some of the terms from the table above will be described in detail in following sections of this manual. For a general understanding of the table, a short explanation of the terms:

Module no. ...	Serial number (1...254) which can be allocated to the module by the user independently from the module type.
active mod no. ...	The module no. with which the module is selected by the initialisation identifier (INIT Id) during the parameter interchange.
I²C EEPROM mod no. ...	The module no. which was stored in the local I²C EEPROM of the module. If no change in this number had been made after the storing, the actual mod no. is identical with the I²C EEPROM module no..

Below the combinations of factors, shown in table 1.4.1, which are decisive for the selection of the default parameters will be explained:

Combination 1a

If the positions of the coding switches are $00 when the module starts after a RESET, the I²C EEPROM data, previously stored, will be overwritten at the moment in which the adjustment is changed from $00 to any other value. The firmware needs about 10 seconds for this. Afterwards a local RESET is triggered by the firmware. During the RESET both LEDs are flashing fast.

The CAN identifier corresponds to the value adjusted at the coding switches.

The module operates with the default parameters.

Combination 1b

The second listed combination occurs if no I²C EEPROM is available or the I²C EEPROM data are faulty. If this is the case, the local software would use the default parameters.

The CAN identifier corresponds to the value adjusted at the coding switches. The actual module no. corresponds to the value which is adjusted on the coding switches.

Combination 2

In the third combination the module no. $00 is stored. The modules works with the default parameters.

Combination 3

In this combination of the factors listed above, the module operates after a RESET with the previously changed and in the I²C EEPROM stored parameters.

Requirements for this are I²C EEPROM status=OK, a coding switch position unequal $00 and a module no., stored in the I²C EEPROM, which has got a value unequal $00.

The CAN identifier with which the module operates and all used parameters are taken from the I²C EEPROM.

The actual module no. with which the module is selected in the initialisation phase corresponds to the module no. stored in the I²C EEPROM.

1.6 Advanced Configuration

With the *Advanced Configuration* you can set up to 15 additional parameters with the coding switches. To set the value of the parameters you have to execute the below described step-by-step instruction.

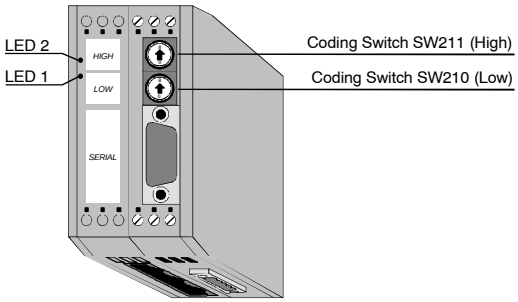


Fig. 1.6.1: Names of LEDs and coding switches

1.6.1 Setting Instruction

Step	State of LEDs		Action/Setting of Coding Switches
	LED 1	LED 2	
1	OFF	OFF	- set both coding switches to 'F' - switch on the power supply of the module
2	flashes fast	flashes slow	- now you have 10 seconds to set the switch HIGH to the parameter number and the switch LOW to the parameter value (parameters see table below) Only, if previous step was step 4: - if do not want to change more parameters, set both coding switches to 'F' to write the changed parameter(s) to the EEPROM
3	lights continuously	flashes slow	- the local firmware verifies your settings of step 2 in 2 seconds
4	flashes fast	lights continuously	- if this light state appears, the changed parameter value is accepted - after 2 seconds the firmware continues with step 2, except you have set both coding switches to 'F', than the firmware continuous with step 5
5	flashes fast	flashes fast	- the LEDs are flashing for approx. 10 seconds, then the module is automatically reset; within this time you have to set the module no. at the coding switches - after the reset the module starts with the changed parameters and the set module no.

1.6.2 Parameter Numbers and Values

Coding Switch		Meaning
HIGH (parameter number)	LOW	
0	baud (0...F)	setting the CAN baudrate index (values see table at page 11)
1-E	reserved	-
F	F	write data to EEPROM

1.7 Summary of LED States

State of LEDs		Meaning
LED 1	LED 2	
OFF	OFF	- power supply off
lights continuously	OFF	- module is working with stored parameters
OFF	lights continuously	- module is working with default parameters
lights continuously	flash slow	- local firmware verifies parameter settings of coding switches (see page 6)
flash fast	flash slow	- time to set parameter value by coding switches (see page 6)
flash fast	lights continuously	- the changed parameter value is accepted (see page 6)
flash fast	flash fast	- microcontroller writes parameter value to EEPROM (see page 6)

1.8 Calling the Commands and Setting the Parameters

If the module is in original condition (default condition at delivery), it operates only by the CAN identifiers on the CAN (see hardware manual) adjusted by hardware.

To report the initialisation parameters to the module nevertheless, a special CAN identifier (INIT Id) has been reserved which is the same for all *esd*-CAN modules.

In spite of the identifier adjusted by the coding switches, the module receives and processes every CAN frame transmitted on the INIT Id.

The INIT Id determined by *esd* has got the value:

$700

(valid since software version V0.8, subject to alterations)

The identifier $700 is reserved for the initialisation, i.e., if this identifier should wrongly be allocated to other functions, the transmitted data will be interpreted as initialisation parameter, nevertheless!

Generally not all modules should be initialized with the same parameters. To distinguish the modules the fourth byte of the six INIT Id bytes has to have the 'actual Module no.' of the wanted module at the initialisation.

The module no. is a characteristic number in the area of $00...$FE which can be freely defined by the user. It is possible, e.g., to number all existing modules (max. 254 modules), independent from the type, continuously.

In a CAN net the same module no. should only be used once. The module no. $00 should not be used, because it is used during the initialisation sequence with global commands, which are valid for all modules.

The actual module no. is identical with the number adjusted at the coding switches, when the module is operated with the default parameters.

But it is also possible to program the module no. freely during the initialisation.

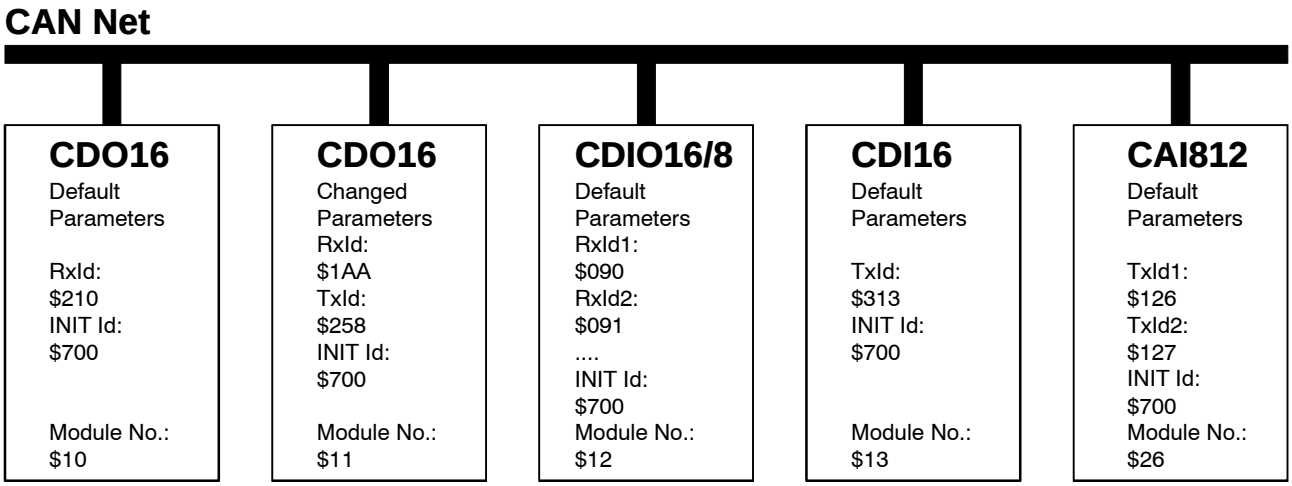


Fig. 1.6.1: Examples for the module no. and the CAN identifier

6 bytes are necessary for the initialisation of a module. The transmitted structure has to have the correct length and the functions corresponding to the module type of board. If this is not the case, the module would not react to the initialisation.

The table below shows the construction of the INIT Id. In the command byte and the sub command bytes the function of the respective initialisation level is determined. The module no. selects the wanted module. In the cells parameter 1 and 2 the wanted parameters are transmitted.

Byte no.	Function	Value range
1	Command	$00...$FF
2	Sub command 1	$00...$FF
3	Sub command 2 1*)	$00...$FF 1*)
4	Module no.	$00, $01...$FF
5	Parameter 1	$00...$FF
6	Parameter 2	$00...$FF
7	Not used	-
8		

(*) Byte 3 (sub command 2) is not needed for the majority of commands and parameters. In this case it should always be allocated with $00.

Table 1.6.1: Data bytes of the INIT Id ($700)

The parameter interchange can lead to the setting of parameters, to the reply of already adjusted parameters or to the execution of a command. At the command interchange and the setting of parameters the highest bit of the command byte is always '1', at the request of parameters it is always '0'.

The requested reply of the module is only sent once by the module. The identifier (CTxId) which had been allocated to the module for this transmission is not stored on the module. If another transmission is desired an INIT Id with the corresponding parameters has to be transmitted to the module again.

If the module processes faultlessly and the CAN is free, it transmits the reply within 200µs (max. 10 ms).

Normally, at the reply of the actual parameter condition of the modules, in the first byte the contents of the received command byte and in the second byte the contents of the sub command byte is given.

This is not the case if the second byte is used for the display of other parameters or an unknown command or sub command byte had been sent to the module. In the second case a message containing only one byte with the contents $FF is given back to the CAN.

The given value ranges of the parameters have to be kept, because otherwise the faultless execution of commands is not guaranteed! No error message occurs after wrong entry of the parameter values.

Below the parameters and commands will be specified. The examples added subsequently to the specification should clarify the function course further.

The description of the bytes of the INIT Id restricts to those which are relevant for the corresponding mode.

Byte 3 (sub command 2) should, if it is not needed, always be recorded with $00. In byte 4 the actual module no. has to be entered, as already mentioned above.

2. Overview of the Implemented Commands and Parameters

The two following tables give a complete summary of all bytes implemented until now and the parameters given back by the module. The individual designations of the commands and parameters will not be explained in detail in the tables in favour of the clarity. The descriptions of the used expressions can be taken from the following chapters in which the individual commands will be described.

The value ranges of the parameters cover all possible entries which are evaluated correctly by the software. Here it has to be noted that partly special functions (e.g. 'function switched off') have been allocated to individual values of the parameters (e.g. $00). The allocation of values which are outside of the given limits is not permissible, because otherwise the perfect function of the addressed module will not be guaranteed anymore.

2.1 Overview of the Commands

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command	don't care	Module No.	Parameter 1	Parameter 2	not used	not used
Request Configuration $00	$00 - Module Type $01 - Active Switch $02 - ASCII Id $03 - Software Rev. $04 - reserved $05 - Serial No.	Please write always $00.	selected Module No. $01...$FF	CTxId $0000...$07FF		These Bytes are not used.	

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command 1	Sub-Command 2	Module No.	Parameter 1	Parameter 2	not used	not used
System Parameter: $81	$00 - Save Parameter			selected Module No. (= Byte 4)	-		
$81 - set	$01 - Module No			new Module No. $00...$FF	-	▲	
$01 - request	$01 - Saved Module No			CTxId $0000...$07FF			
$81 - set	$02 - CAN-Status-Byte	▲ Please write always $00.	▲ selected Module No. $01...$FF	cstat $00	-	These Bytes are not used.	
$01 - request				CTxId $0000...$07FF			
$81 - set	$03 - Bitrate	▼	▼	bust 0 (=BTR0) (see Controller-Manual)	bust 1 (=BTR1) (see Controller-Manual)	▼	
$01 - request				CTxId $0000...$07FF			
$81 - set	$04 - Watchdog Tx Identifier			WTxId $0000...$07FF			
$01 - request				CTxId $0000...$07FF			
$81 - set	$05 - Watchdog Time	WDLife TimeFactor		WDtime $0000...$FFFF [ms]			
$01 - request		-		CTxId $0000...$07FF			

Overview

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command	don't care	Module No.	Parameter 1	Parameter 2	not used	not used
TxIds $82 - set	$00 - TxId1 $01 - TxId2 $02 - TxId3 : n - TxId(n+1)	Please write always $00.	selected Module No. $01...$FF	TxId $0000...$07FF		These Bytes are not used.	
$02 - request				CTxId $0000...$07FF			

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command	don't care	Module No.	Parameter 1	Parameter 2	not used	not used
RxIds $83 - set	$00 - RxId1 $01 - RxId2 $02 - RxId3 : n - RxId(n+1)	Please write always $00.	selected Module No. $01...$FF	RxId $0000...$07FF		These Bytes are not used.	
$03 - request				CTxId $0000...$07FF			

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command	don't care	Module No.	Parameter 1	Parameter 2	not used	not used
Tx-Activate-Time $84 - set	$00 - act.-T. TxId1 $01 - act.-T. TxId2 $02 - act.-T. TxId3 : n - act.-T. TxId(n+1)	Please write always $00.	selected Module No. $01...$FF	tx_act $0000...$FFFF [ms]		These Bytes are not used.	
$04 - request				CTxId $0000...$07FF			

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command	don't care	Module No.	Parameter 1	Parameter 2	not used	not used
User Parameter $86 - set	User Parameter No. $00...$7F	Please write always $00.	selected Module No. $01...$FF	Para $0000...$FFFF		These Bytes are not used.	
$06 - request				CTxId $0000...$07FF			

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command 1	Sub-Command 2	Module No.	Parameter 1	Parameter 2	not used	not used
Service Request $7F	Module No_LOW $00...$FF	Module No_HIGH $00...$FF	Please write always $00.	CTxId $0000...$07FF		These Bytes are not used.	

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Command	Sub-Command	don't care	Module No.	Parameter 1	Parameter 2	not used	not used
Supervisor Commands $FF	$00 - RESET Module	Please write always $00.	$00 - all Modules $01...$FF - selected Module	$AAAA - RESET		These Bytes are not used.	
	$01 - reserved			-			
	$02 - Supervisor Watchdog			-			
	$03 - Default RESET			$AAAA - Default-Reset			
	$04 - Suspend/ Continue Module			$5A5A - suspend $A5A5 - continue			
	$05 - RESET CAN-Status			-			

2.2 Overview of the Returned Parameter Values

Command: Request...	Sub-Command	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Configuration $00	$00-Module Type	$00	$00	type	iomode	-	-	-	-
	$01-Active Switch	$00	$01	switch	-	-	-	-	-
	$02- ASCII Id	$00	a (ASCII)	b (ASCII)	c (ASCII)	d (ASCII)	e (ASCII)	Module No.-H (ASCII)	Module No.-L (ASCII)
	$03- Software Revision	$00	'V' in ASCII	level H	'.' in ASCII	level L	revision	esd/cms	protocol-revision
	$04- reserved	$FF	-	-		-			
	$05- Serial No.	$00	$05	u (ASCII)	v (ASCII)	w (ASCII)	x (ASCII)	y (ASCII)	z (ASCII)

Command: Request...	Sub-Command	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
System Parameter: $01	$00-Saved Module No.	$01	$00	saved Module No.	-	-	-	-	-
	$01-Active Module No.	$01	$01	active Module No.	-	-	-	-	-
	$02 - CAN-Status Byte	$01	$02	cstat	-	-	-	-	-
	$03 - Saved Bitrate	$01	$03	bust 0	bust 1	-	-	-	-
	$04 - WD-Tx-Id	$01	$04	WTxId		-	-	-	-
	$05 - WD-Time	$01	$05	WDtime [ms]		WDLifeTime Factor	-	-	-

Command: Request...	Sub-Command	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
TxIds $02	$00-TxId1	$02	$00	TxId1		-	-	-	-
	$01-TxId2	$02	$01	TxId2		-	-	-	-
	$02-TxId3	$02	$02	TxId3		-	-	-	-
	:	$02	:	:		-	-	-	-
	n -TxId(n+1)	$02	n	TxId(n+1)		-	-	-	-

Command: Request...	Sub-Command	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
RxIds $03	$00-RxId1	$03	$00	RxId1S		RxId1E		-	-
	$01-RxId2	$03	$01	RxId2S		RxId2E		-	-
	$02-RxId3	$03	$02	RxId3S		RxId3E		-	-
	:	$03	:	:		:		-	-
	n -RxId(n+1)	$03	n	RxId(n+1)S		RxId(n+1)E		-	-

Command: Request...	Sub-Command	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
Tx Activate Time $04	$00- Time TxId1	$04	$00	tx-act1 [ms]		-	-	-	-
	$01- Time TxId2	$04	$01	tx-act2 [ms]		-	-	-	-
	$02- Time TxId3	$04	$02	tx-act3 [ms]		-	-	-	-
	:	$04	:	:		-	-	-	-
	n-Time TxId(n+1)	$04	n	tx-act(n+1)		-	-	-	-

Command: Request...	Sub-Command	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
User Parameter $06	$00-Para 0	$06	$00	Para0		-	-	-	-
	$01-Para 1	$06	$01	Para1		-	-	-	-
	...	$06	...	...		-	-	-	-
	$7F-Para127	$06	$7F	Para7F		-	-	-	-

(-) This Byte is not transmitted.

3. Description of the Commands and Parameters

3.1 Configuration Reply

The transmission is called by transmitting a 'request configuration' command. The identifier on which the module should transmit the information on the CAN is reported to it by byte 5 and 6.

Contrary to the other commands only parameters are called with this command.

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Function	Command	Sub command	Not used	Module no.	Parameter 1	Parameter 2
Value	$00	$00...$05	Always write $00	Selected module no. $01...$FF	CTxId $0000...$07FF	

Table 3.1.1: Bytes of the command 'request configuration'

Explanation of the Bytes Transmitted to the Module:

Command... The command 'request configuration' requests the transmission of the actual parameters of the module.

Sub command... The sub command determines the configuration bytes which should reply:

Sub command	Reply of the parameters
$00	Module type
$01	Active switch
$02	ASCII Id
$03	Software rev.
$04	reserved
$05	Serial number

Table 3.1.2: Selection of the configuration reply by sub command

Parameter 1 and 2... With these parameters it is reported to the module to which CAN Id. it should transmit the requested reply.
The module does only transmit once on this identifier. The identifier is not stored on the module.
If another transmission is desired another INIT Id with the corresponding parameters has to be transmitted to the module.

Transmission of the adjusted configuration by the module:

Decisive for the selection of the message to be transmitted is the value of the sub command received by the module. The following table shows the information which is transmitted to the CAN by the module.

Sub command	Reply of the parameter	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$00	Module type	$00	$00	type	iomode	-	-	-	-
$01	Active switch	$00	$01	switch	-	-	-	-	-
$02	ASCII Id	$00	a	b	c	d	e	Mod No ASCIIH	Mod No ASCIIL
$03	Software rev.	$00	'V' ASCII	level H	'.' ASCII	level L	rev.	esd/cms	prot-rev
$04	reserved	$00	not defined						
$05	Serial number	$00	$05	u	v	w	x	y	z

(-) Byte is not transmitted.

Table 3.1.3: Transmitting the adjusted configuration

Explanation of the bytes transmitted by the module:

Sub command $00 --▸ module type

Byte 1... The first byte replies the contents of the received command byte (here always $00).

Byte 2... The second byte replies the contents of the received sub command (here always $00).

Byte 3:
type... In this cell the type of the *esd*-CAN module is coded. The following table shows examples of the existing type designations:

type	Modules (examples)
$00	reserved
$01	CDO16
$02	CDI16
$03	CAI810
$04	CAO812
$05	CDIO16/16
$06	CAI812
$07	reserved
$08	CREL8
$09	CSC595
$0A	CPIO16/8
$0B	CCOM4
$0C	CCOM1
$0D	PTIDAC
$0E	SPS16
$0F	CMIO
$10	reserved
$11	CTERM
$12	CBIP
$13	CANSAT
$14	AIS16
$15	CI488
$16	CAN-PT100/DMS4
$17	CDMS4I
$18	CAN-PCC
$19	CCOM1
$1A	XMIO4
$1B	reserved
:	:
$1F	reserved
$20	LasCon I/O
:	:
$30	SIO
$31	SIO4
:	:
$FE	reserved
$FF	reserved

Table 3.1.4 Examples for module type designations

Byte 4:
iomode...

The byte 'iomode' contains, broken down into 6 bits, information about the operating mode of the addressed module. A '1' of the respective bit signalizes the operating mode possible for this module:

Bit	Function	Example: CAN-CMIO
0	Output	1
1	Input	1
2	Digital	1
3	Analog	1
4	Controller	0
5	Serial	0
6	Reserved 1*)	0
7	Reserved 1*)	0

1*) These bits are read as '0'.

Table 3.1.5: Bits in parameter 'iomode'

Example: At a CAN-CMIO module the value $0F would be replied for 'iomode'. The value for a CAN-CBM-SIO is not defined at the moment.

Sub command $01 --▸ active switch

Byte1,
Byte2...

See sub command $00.

Byte 3:
switch...

The byte 'switch' replies the number adjusted on the coding switches.

If the module no. filed in the EEPROM has got the value $00 or if the EEPROM data are not OK, the actual module no. by which the module is addressed during the initialisation corresponds to this coding switch number.

Sub command $02 --▸ ASCII Id

Byte 1... See sub command $00.

Byte 2 - Byte 6
a, b, c, d, e... These bytes describe the module type in ASCII code.

Byte 7, Byte 8:
mod no. ASCII... These two bytes describe the module name and the active module number in ASCII code. The following table gives an example for the display of these bytes in ASCII code. It is a CBM-SIO4 module with the module no. $99.

Byte	2	3	4	5	6	7	8
Parameter	a	b	c	d	e	Module no. H	Module no. L
ASCII Id SIO4	S	I	O	4	1	9	9

Table 3.1.6: Example for an ASCII Id (the exact value for the SIO4 is not defined at the moment)

Sub command $03 --▸ software revision

Byte 1... See sub command $00.

Byte 2 - Byte 5
'V', level,
'.', rev.... In ASCII code these bytes describe the revision number of the firmware of the CAN core used on the module.
Byte 2 and byte 4 are permanently allocated with the ASCII symbols 'V'($56) or '.'($2E).
In byte 3 and byte 5 the actual revision number is described.
Byte 6 contains a letter which stands for the revision of the module-specific firmware.
In byte 7 is returned, which protocol is implemented (esd CAN protocol or CMS protocol). An 'E' means, that the esd CAN protocol is implemented and a 'C' means, that the CMS protocol is implemented.
Byte 8 returns the actual revision number of the used protocol (e.g. of the esd CAN protocol).

Byte	2	3	4	5	6	7	8
Parameter	'V'	level H	'.'	level L	rev.	esd/cms	prot-rev
ASCII display	V	1	.	0	a	E	0

Table 3.1.7: Example for the ASCII software rev no. 'V1.0aE0'

Sub command $04 --▸ reserved

Sub-Command $05 --▸ serial-number

Byte 1, Byte 2 ... refer to sub command $00.

Byte 3...8 ... Hardware serial number of the CAN module.

Byte	3	4	5	6	7	8
Parameter	u	v	w	x	y	z
ASCII display	A	A	0	0	1	2

Table 3.1.7: Example for the ASCII serial no. 'AA0012'

3.2 System Parameters

With the command $81 the parameters described below are set. By command $01 and the according sub command the module is lead to reply the actual parameters.

If the command 'store parameter' (sub command $00) is transmitted to the module, all previously interchanged parameters are stored in the local I²C EEPROM.
If the module is reset (RESET) or the supply voltage is switched off, the entered parameters are lost if this command has not been entered before.

After a RESET command or a power-on RESET the module operates with the stored parameters (identifiers etc.). If the programming has been unsuccessful (e.g. no I²C EEPROM or defect) the module uses the standard parameters (e.g. identifier =▸ coding switches).

The module no. with which the module is selected at the parameter interchange corresponds to the adjustment of the coding switches (provided no modification had been made so far). By the sub command $01 it is possible to allocate another module no. to the module. The new number is active immediately after the setting and the number adjusted by the coding switches is ignored.

The CAN-status byte offers various information about the condition of the module: It is shown if the module had previously not been connected to the CAN, if the default parameters had been activated after a RESET, if the last RESET had been caused by a power-on cycle, etc.

By the sub command $03 it is possible to change the CAN bitrate of the module which was adjusted by the configuration jumper of the module. The new bitrate is only activated after the parameters have been stored by sub command $00 and a RESET has been triggered.

Between the modules and a supervisor (master) a mutual function control by a watchdog protocol similar to the CMS specification can occur. By the sub commands $04 and $05 it is possible to interchange a watchdog identifier and a watchdog time.

<table>
<tr><th></th><th>Byte 1</th><th>Byte 2</th><th>Byte 3</th><th>Byte 4</th><th>Byte 5</th><th>Byte 6</th></tr>
<tr><th>Function</th><th>Command</th><th>Sub command</th><th>Not used</th><th>Module no.</th><th>Parameter 1</th><th>Parameter 2</th></tr>
<tr><td rowspan="8">Value</td><td>Request: $01</td><td>$00...$03</td><td rowspan="6">Always allocate $00</td><td rowspan="8">Selected module no. $01..$FF

= active module no.</td><td colspan="2">CTxId
$0000...$07FF</td></tr>
<tr><td rowspan="7">Set: $81</td><td>$00 store parameter</td><td>active mod no.</td><td>-</td></tr>
<tr><td>$01 module no.</td><td>new mod no.</td><td>-</td></tr>
<tr><td>$02 status</td><td>cstat</td><td>-</td></tr>
<tr><td>$03 bitrate</td><td>bust 0</td><td>bust 1</td></tr>
<tr><td>$04 watchdog Id</td><td colspan="2">WTxId
$0000...$07FF</td></tr>
<tr><td>$05 watchdog time</td><td>WDLife TimeFactor
$00...$FF</td><td colspan="2">WDtime
$0000...$FFFF</td></tr>
</table>

Table 3.2.1: Bytes of the commands 'store parameter, module no., bitrate and watchdog'

<u>Setting the Commands and Parameters (Command $81):</u>

Command... The command $81 leads to the setting of parameters or to the activation of the commands.
The command $01 leads to the reply of the actual parameters.

Sub command... The sub command selects the desired parameter interchange or the command to be executed.

Parameter1, Parameter2 Following sub commands are implemented:

Sub command	Function
$00	Storing the actual parameter
$01	Setting a new module no.
$02	Setting the CAN-status byte
$03	Setting a new bitrate
$04	Setting the watchdog Tx identifier
$05	Setting the watchdog time

Table 3.2.2: Function of the sub commands

The kind of the parameters interchanged to the module depends on the selected sub command:

Sub command $00 --▸ store parameter

active
mod. no... To store all interchanged parameters (also those of other commands) the actual module no. has to be entered into byte 5 when calling this sub command.
The actual module no. is either the number adjusted by the coding switches (default parameter active) or the number changed by sub command 'set module no.'.

Sub command $01 --▸ set module no.

new. mod. no.. Here the desired new module no. is entered. The module is addressed immediately after this command by the new module no. The module no. adjusted by the coding switches is ignored.
If the new module no. should remain active after a RESET, the parameters have to be stored by the sub command 'store parameters' before a power down or a RESET.

Sub command $02 --▸ set CAN-status byte

cstat... A 'set' access onto the CAN-status byte sets the bits 2 and 7 of the byte to '0'. All other bits of the status byte remain unchanged, because they serve as read only information (see also 'requesting the actual parameters').

Bit 2 shows that a CAN error has been detected by the module. The bit serves the documentation of errors which do not cause a 'standstill' for the CAN, but remain only for a short period. Error arising for a short time can easily be overlooked, because they make themselves visible only by a temporal limited flashing of the status LED.
The bit is set to '1' when an error has been detected. A 'set' access with any data (recommended: byte 5 = $00) onto the status byte or each RESET reset the bit back to '0'.

The CAN-error bit can also be reset by the supervisor command 'reset CAN error' (sub command $05). The other bits remain unchanged by this command.

Notes on the internal management of the CAN-error bit:

The CAN-error bit is set by the local software if the status bit of the CAN controller 'error status' is activated. (see constat).
After the first recognition of a CAN error the controller at first tries to transmit or receive data repeatedly. If it recognizes after several attempts that the error does not occur anymore, it does not take back its error bit at once. First further successful transmissions have to take place to count back the internal error counter again. Supervised Tx transfers which are addressed to other modules are also counted as successful transmissions. But if only one module and one CAN master are installed on the bus, the master possibly has to transmit some messages first to reset the error counter and therefore reset the controller status bit.
Therefore it is possible that the CAN-error bit of the CAN-status byte is still active after only one reset, because the error bit of the controller is still active.

Bit 7 of the byte has got the designation 'new on bus' and shows if the module processes for the first time on the CAN:
A CAN master is able to evaluate and set the bit to zero to document on the module that it noted the presence of the module on the CAN. If the bit has got the value '1' at the reading, in this application of the bit the module had not been found by a master so far after the last RESET.

A 'set' access with any data (recommended: byte 5 = $00) onto the status byte sets the bit onto '0'.

Sub command $03 --▸ set bitrate

bust0, bust1.. These two bytes set the contents of the registers BTR0 and BTR1 of the CAN controller SJA1000 which determine the bitrate of the CAN interface.

An allocation of the register contents to the bit rates can be taken from the table below.

Contrary to the other commands this parameters only become active after the actual parameter set was stored in the EEPROM (store parameters) and a RESET was triggered on the module.

If no communication is possible with the module, this is often due to a wrong adjustment of the bitrate.
If the local software discovers a malfunction on the CAN, the bitrate is adjusted again to the default value (adjustment at the configuration jumper of the module), but without changing the other parameters.

The specified typical line lengths base on experimental values from experience. The minimum reachable line lengths result from the 'worst case' delay times of the used components.

baud [hex]	CAN controller register		Bit rate [kBit/s]	Typical values of the reachable line length l_{max} [m]	Minimum values of the reachable line length l_{min} [m]
	BTR0 [HEX]	**BTR1 [HEX]**			
0	**00**	**14**	1000	37	20
1	**00**	**16**	800	59	42
2	**00**	**18**	666.6	80	65
3	**00**	**1C**	500	130	110
4	**01**	**18**	333.3	180	160
5	**01**	**1C**	250	270	250
6	**02**	**1C**	166	420	400
7	**03**	**1C**	125	570	550
8	**04**	**1C**	100	710	700
9	**45**	**2F**	66.6	1000	980
A	**09**	**1C**	50	1400	1400
B	**4B**	**2F**	33.3	2000	2000
C	**18**	**1C**	20	3600	3600
D	**5F**	**2F**	12.5	5400	5400
E	**31**	**1C**	10	7300	7300

The specifications in the table base on the limit values of the bit timing of the CAN protocol, the runtime of the local CAN interface and the runtime of the cable. The runtime of the cable is assumed with about 5.5 ns/m. Further influences, e.g., by the terminal resistances, the specific resistance, the geometry of the cable of outer interference effects at the transmission have not been included in the specifications!

Table 3.2.3: Allocation of the bitrate to the registers of the controller SJA1000

Sub command $04 --► set watchdog Tx identifier Sub command $05 --► set watchdog time (guard time)

The watchdog protocol functions with following scheme:

1. After a RESET, the watchdog is inactive. The 'master' interchanges with these sub commands a Tx identifier, a watchdog time (Guard Time) and a life time factor to the module. Setting the watchdog time to values > '0' and setting the life time factor to values > '0' enables the local watchdog on the module. The watchdog is not activated at that moment! The watchdog time (WDtime) and the watchdog life time factor (WDLifeTimeFactor) are ‘0’.

2. The watchdog Tx identifier has to be saved in the EEPROM. The watchdog time (WDtime) and the watchdog life time factor (WDLifeTimeFactor) are ‘0’ after RESET.

3. Now a remote request has to be received for this Tx identifier or the command 'Supervisor Watchdog' ($FF) has to be received, before the watchdog time is counted down for the first time. After this the master has to send a RTR or a supervisor command at the given Tx identifier within the time (WDtime x WDLifeTimeFactor), otherwise a RESET is triggered at the module. Receiving the RTR or the command shows the module that the master is still active.

4. If the remote request or the supervisor command arrives within the given time, the module transmits a one byte containing message back on the Tx identifier. The byte is constructed as follows:

Bit	7	6	5	4	3	2	1	0
Contents	Toggle bit	x	x	x	x	CMS State		

Table 3.2.4: Watchdog reply of the modules

The toggle bit changes its condition with each transmission. In normal position, i.e. before the first transmission, it has got the value '0'. The bits 6 to 3 are always transmitted as '1', if the module is in the state ‘preoperational’ and are always transmittet as ‘0’, if the module is in the state ‘operational’. The module turns into the state ‘operational’ after the ‘First Tx-activate Delay’. The bits 2 to 0 contain a CMS status message which is permanently programmed on '101' with all modules.
Possible replies therefore are in preoperational state: **$7D** and **$FD**.
Possible replies therefore are in operational state: **$05** and **$85**.

5. The master can recognize from the reply that the module is still active. After a RESET that is generated by the watchdog the module will not answer to receiving RTR frames of the master (first the watchdog time and the life time factor has to be set).

The interchanged parameters of the sub command $04 and $05 have the following meaning:

WTxId ... In this word the Tx identifier is interchanged on which the master transmits the remote request and on which the module transmits the response.

WDtime... The watchdog time after that a local RESET is generated results from the time set in WDtime multiplied by the life time factor, that is set in WDLifeTimeFactor. **The product has to be greater than '0' to enable the watchdog!**

WDtime [HEX]	Watchdog time in [ms]
0000	Watchdog disable (default adjustment)
0001 0002 0003 .. FFFF	1 2 3 ... 65.535

Table 3.2.5: Allocation of the parameters to the watchdog time

WDLifeTime Factor... Multiplier for the watchdog time (function is described at WDtime).
value: $00...$FF

Requesting the actual parameters by the module (command $01):

Command... The command $01 leads to the reply of the actual parameters.

Sub command... The sub command determines the configuration bytes which should be returned:

Sub command	Reply of the parameter
$00	Saved module no.
$01	Active module no.
$02	CAN-status byte
$03	Saved bitrate
$04	Watchdog Tx identifier
$05	Watchdog time

Table 3.2.6: Definition of the reply by sub command

Parameter 1 and 2... With these parameters it is reported to the module onto which CAN Id it should transmit the requested reply.

Decisive for the selection of the requested message is the sub command value received by the module.

The following table shows the information which the module transmits onto the CAN if a request command has been received.

Sub command	Reply of the parameter	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$00	Saved mod. no.	$01	$00	saved mod no	-	-	-	-	-
$01	Active mod. no.	$01	$01	active mod no	-	-	-	-	-
$02	CAN status	$01	$02	cstat	-	-	-	-	-
$03	Saved bitrate	$01	$03	bust0	bust1	-	-	-	-
$04	Watchdog TxId	$01	$04	WTxId		-	-	-	-
$05	Watchdog time	$01	$05	WDtime		WDLife TimeFactor	-	-	-

(-) Byte is not transmitted

Table 3.2.7: Transmitting the actual parameters module no. and bitrate

Explanation of the bytes transmitted by the module:

Sub command $00 --▸ stored module no.

Byte 1... The first byte returns the contents of the received command byte (here always $01).

Byte 2... The second byte returns the contents of the received sub command byte (here always $00).

Byte 3: saved mod no... In this cell the module no. is returned which is actually stored in the EEPROM. After a power-on RESET or a RESET command this module no. is the number with which the module is selected if no other module no. had been stored after a RESET.
If the module operates with the default parameters, then the value $00 is returned here always.

Sub command $01 --▸ actual module no.

Byte1, Byte2... See sub command $00.

Byte 3: active mod no... Here the module no. is returned which is active at the moment. This number is identical with the number of this INIT Id entered into byte 4.

Sub command $02 --▸ CAN-status byte

Byte1,
Byte2... See sub command $00.

cstat... Byte 3 describes status information about the condition of the CAN components of the selected module (see the following table).

Bit	Designation
0	Power-down RESET
1	Suspend bit
2	(Error on CAN)
3	(I²C error)
4	EEPROM size
5	(I²C busy)
6	Default wake up
7	New on bus

Table 3.2.8: Bits of the parameter 'cstat'

Explanation of the bits of parameter 'cstat':

Power-down RESET (bit 0)
Bit 0 shows if the last RESET on the module had been triggered by a power-down RESET. If the bit has got the value '1', the last RESET had been triggered by removing the supply voltage. For all other RESET causes (RESET by supervisor command, RESET by EMERGENCY STOP) the bit is '0'.

Suspend bit (bit 1)
In bit 1 is described if the module is in condition 'suspended' or in active condition:

By the supervisor command 'suspend module' it is possible to put individual or all modules of the CAN into a delay condition. In this operating mode the actual condition of the outputs of the selected module remains unchanged (if available) and cannot be changed anymore until the module is enabled again. Parameters and commands are received and filed by the module but not evaluated: It is possible, e.g., to set raw data on the module without changing the normal conditions at once.

In condition 'suspended' the module does not transmit data onto the CAN. This is valid for each kind of Tx transfer with the exception of this configuration reply.
If the module is activated again by the command 'continue', possibly existing outputs are set at once corresponding to the last received data and the module operates with the last received parameters.

If bit 1 has got the value '1', the module is in condition 'suspended'. In normal operating mode (continue) bit 1 has got the value '0'.

Error on CAN (bit 2)
This bit is always ‘0’ at the CAN-CBM-SIO module.

I²C error (bit 3)
This bit is always ‘0’ at the CAN-CBM-SIO module.

EEPROM size (bit 4)
If bit 4 has got the value '1', the module has got an EEPROM whose memory capacity is bigger than 128 bytes. An EEPROM of this size is prerequisite for the allocation of pin names for each individual connection pin, because the names are filed in the EEPROM (see also chapter 'Allocation of Pin Names').

I²C busy (bit 5)
This bit is always ‘0’ at the CAN-CBM-SIO module.

Default wake up (bit 6)
If the bit 'default start' has the value '1', the module 'awoke' with the default parameters after the last RESET.

New on bus (bit 7)
The bit 'new on bus' can be evaluated by the CAN master to recognize if it already found this module after the last RESET or if the module is new on the bus. The bit is set to the value '1' after each RESET which corresponds to condition 'new on bus'. The master can reset the bit to show that it found the module. Apart from 'error on CAN' it is the only bit of this byte which can be reset.

Sub command $03 --▸ stored bitrate

Byte1,
Byte2... See sub command $00.

Byte 3, 4:
bust 0,
bust1 ... These two bytes describe the contents of the registers BTR0 and BTR1 of the CAN controller SJA1000 which determine the bitrate of the CAN interface. The allocation of the register contents to the bit rates has already been explained in the description of setting these registers.

Sub command $04 --▸ watchdog Tx identifier

Byte1,
Byte2... See sub command $00.

Byte 3, 4:
WTxId-H,
WTxId-L ... These two bytes return Tx identifiers on which the module transmits the watchdog reply.

Sub command $05 --▸ watchdog time

Byte1,
Byte2... See sub command $00.

Byte 3, 4:
WDtime... Here the watchdog time is returned (value range see 'Setting the watchdog time').
The value is only returned, if the value of WTxId is valid.

Byte 5:
WDLifeTime
Factor... Return of the Life Time Factor ($00...$FF).
The value is only returned, if the value of WTxId is valid.

3.3 Process TxIds

By this command one or more Tx identifiers are allocated to the module, depending on the module type, by which it is able to transmit the data onto the CAN (set TxIds).

The new Tx identifier replaces immediately after he had been received by the module the default identifier adjusted by the coding switches. If the new Tx identifier should remain after a RESET or a power-down, it has to be filed into the local EEPROM by command 'store parameter'.

Apart from that the module can be forced to transmit a reply about the Tx identifiers actually used by it onto the CAN by the Tx identifier CTxId (request TxIds).

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Function	Command	Sub command	Not used	Module no.	Parameter 1	Parameter 2
Value	Request: $02	$00...n	Always allocate with $00	Selected module no.	CTxId $0000...$07FF	
	Set: $82			$01...$FF	TxId $0000...$07FF	

Table 3.3.1: Bytes of the command set/request TxIds

Command... By the command $02 the actual Tx identifiers are requested by the module. After this request the module transmits the information on the identifier 'CTxId', entered into byte 5 and 6.
By the command $82 new Tx identifiers (TxId) to transmit I/O data are allocated to the module corresponding to the sub command. The new Tx identifiers are interchanged by byte 5 and 6.

Sub command... By the sub command it is selected which Tx identifier should be interchanged. Depending on the module type a different number of Tx identifiers can be set.

Parameter 1 and 2... With these parameters it is reported to the module at a request command onto which CAN Id (CTxId) it should transmit the requested reply.
The module only transmits once on this identifier. The identifier is not stored on the module. If another transmission is wanted, another INIT Id with the corresponding parameters has to be transmitted to the module again.

At a set command the 11 bit wide Tx identifier on which it should transmit the data is interchanged to the module by the parameters.

By the sub command it is selected which one of the possible Tx identifiers should be set or reset:

Sub command	Tx identifiers on parameters 1 and 2
$00	TxId1
$01	TxId2
$02	TxId3
:	:
n	TxId(n+1)

Table 3.3.2: Selection of the identifiers by the sub command

The following table shows how the bits of Tx identifiers have to be allocated to the bytes 5 and 6:

	Byte 5								Byte 6							
	Parameter 1								Parameter 2							
Bits in byte 5 and 6	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	TxId-H								TxId-L							
Bits of the TxId	-	-	-	-	-	id11	id10	id9	id8	id7	id6	id5	id4	id3	id2	id1

(-) These bits are not evaluated.

Table 3.3.3: Bits of the Tx identifiers in byte 5 and byte 6

Reply of the modules to the command 'request TxIds':

The reply of a Tx identifier depends on the value of the received sub command.

The following table shows the information which the module transmits onto the CAN.

Sub command	Reply of the parameter	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$00	TxId 1	$02	$00	TxId1		-	-	-	-
$01	TxId 2	$02	$01	TxId2		-	-	-	-
$02	TxId 3	$02	$02	TxId3		-	-	-	-
:	:	$02	:	:		-	-	-	-
n	TxId (n+1)	$02	n	TxId(n+1)		-	-	-	-

(-) Byte is not transmitted

Table 3.3.4: Transmitting the actual Tx identifiers

Explanation of the bytes transmitted by the module:

Byte 1... The first byte returns the contents of the received command byte (here always $02).

Byte 2... The second byte returns the contents of the received sub command byte. The sub command designates the selected identifier.

Byte 3,
Byte 4... In these two cells the actual Tx identifier is returned. The bits of the identifier are described in byte 3 and 4 and returned in that form as they have to be entered at setting in byte 5 and 6. (See also table 'Bits of the Tx identifier in byte 5 and 6'.)

3.4 Process RxIds

Depending on the module type one or more Rx identifiers are allocated to the module on which it is able to receive data to the CAN (set RxIds).

The new Rx identifier replaces the default identifier adjusted by the coding switches immediately after it had been received by the module. If the new Rx identifier should remain after a RESET or a power-down, it has to be filed into the local EEPROM by the command 'store parameter'.

Apart from this it is possible to make the module transmit a reply onto the CAN by the Tx identifier CTxId (request RxIds) about the Rx identifiers actually used by the module.

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Function	Command	Sub command	Not used	Module no.	Parameter 1	Parameter 2
Value	Request: $03	$00...n	Always allocate with $00	Selected module no.	CTxId $0000...$07FF	
	Set: $83			$01...$FF	RxId $0000...$07FF	

Table 3.4.1: Bytes of the command set/request RxIds

Command... By the command $03 the actual Rx identifiers are requested by the module. The module transmits the information on the identifier 'CTxId' entered in byte 5 and byte 6 after this request.

By the command $83 new Rx identifiers for the transmission of I/O data are allocated to the module. The new Rx identifier are interchanged by byte 5 and byte 6.

Sub command... By the sub command it is selected which Rx identifier should be interchanged. Depending on the module type a different number of Rx identifiers can be set.

Parameter 1 and 2... With these parameters it is reported to the module at a request command onto which CAN Id (CTxId) it should transmit the requested reply.
The module only transmits once on this identifier. The identifier is not stored on the module. If another transmission is wanted, another INIT Id with the corresponding parameters has to be transmitted to the module again.

At a set command the Rx identifier is interchanged to the module by these parameters on which it should transmit the data.

By the sub command it is selected which one of the possible Rx identifiers should be set or reset:

Sub command	Rx identifiers on parameter 1 and 2
$00	RxId1
$01	RxId2
:	:
n	RxId(n+1)

Table 3.4.2: Selection of the Rx identifiers by the sub command

The following table shows how the bits of Rx identifiers have to be allocated to the bytes 5 and 6:

	Byte 5								Byte 6							
	Parameter 1								Parameter 2							
Bits in byte 5 and 6	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	RxId-H								RxId-L							
Bits of the RxId	-	-	-	-	-	id11	id10	id9	id8	id7	id6	id5	id4	id3	id2	id1

(-) These bits are not evaluated.

Table 3.4.3: Bits of the Rx identifier in byte 5 and byte 6

Reply of the modules to the command 'request RxIds':

The reply of the Rx identifiers depends on the value of the received sub command.

The following table shows the information which the module transmits to the CAN.

Sub command	Reply of the parameter	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$00	RxId 1	$03	$00	Rx1S		Rx1E		-	-
$01	RxId 2	$03	$01	Rx2S		Rx2E		-	-
$02	RxId 3	$03	$02	Rx3S		Rx3E		-	-
:	:	$03	:	:		:		-	-
n	RxId (n+1)	$03	n	Rx(n+1)S		Rx(n+1)E		-	-

(-) Byte is not transmitted

Table 3.4.4: Transmitting the actual Rx identifiers

Explanation of the bytes transmitted by the module:

Byte 1... The first byte returns the contents of the received command byte (here always $03).

Byte 2... The second byte returns the contents of the received sub command byte. The sub command designates the selected identifier.

Byte 3-
Byte 6... In these cells the actual Rx identifier is returned. The bits of the identifier are described in byte 3 and 4 (or byte 5 and 6) and returned in that form as they have to be entered at setting in byte 5 and 6. (See also table 'Bits of the Rx identifier in byte 5 and 6'.)

Apart from the allocation of Rx identifiers described above there is the possibility to allocate, depending on the module type, an own identifier to each output bit, byte or word (see chapter 'Rx Block Mode'). If this block mode is activated, the first (Rxm_S) and the last (Rxm_E) identifier of the block are returned at an RxId request command (m=1,2,3,4...).

If the module is not in Rx block mode, the returned identifier in byte 3 and byte 4 is identical with the one in byte 5 and byte 6.

3.5 Cyclic Tx Transfers (Tx Activate Time)

The interchange of the parameter 'Tx activate time' makes the module transmit I/O data onto the CAN in regular periods of time.

This parameter has no effort on the CAN-CBM-SIO module !

The data are transmitted on the actual Tx identifier (see also chapter 'Setting/Reading TxIds', and 'Parameters after RESET').

Apart from that the module can be forced to transmit a reply by the Tx identifier CTxId onto the CAN about the 'activate times' it uses at the moment.

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Function	Command	Sub command	Not used	Module no.	Parameter 1	Parameter 2
Value	Request: $04	$00...n	Always allocate with $00	Selected module no.	CTxId $0000...$07FF	
	Set: $84			$01...$FF	tx_act $0000...$FFFF	

Table 3.5.1: Bytes of the command set/request TxIds

Command... By the command $04 the actual Tx activate times are requested by the module. After this request the module transmits the information on the identifier 'CTxId' of byte 5 and 6.
By the command $84 new Tx activate times to transmit I/O data are allocated to the module. The new period time is interchanged by byte 5 and byte 6.

Sub command... By the parameter sub command it is selected for which Tx identifier the new Tx activate time should be interchanged. Depending on the module type it is possible to allocate an activate time for a different number of Tx identifiers.

Parameter 1 and 2... By these parameters it is reported to the module at a request command onto which CAN Id (CTxId) it should transmit the requested reply.
The module transmits only once on this identifier. The identifier is not stored on the module. If another transmission is wanted, another INIT Id with the corresponding parameters has to be transmitted to the module again.

At a set command the activate time is interchanged to the module by these parameters.

By the sub command it is selected to which one of the four possible Tx identifiers the transmitted activate time should be allocated.

Sub command	Tx activate time (parameters 1 and 2)
$00	tx_act 1
$01	tx_act 2
$02	tx_act 3
:	:
n	tx_act (n+1)

Table 3.5.2: Selection of the Tx activate time by the sub command

The parameters of the activate time show the period of time from the last data transmission, after which the module automatically starts a new transmission.
The last data transmission could have been the previous Tx transfer of this cycle, but also every other Tx transfer transmitted by the module (e.g. a transmission initiated by a 'remote request').

tx_act [HEX]	Activate time in [ms]
0000	Function switched off
0001	1
0002	2
0003	3
..	...
FFFF	65.535

Table 3.5.3: Allocation of the parameters to the Tx activate time

Reply of the modules to the command 'request Tx activate time':

The reply of the activate time depends on the value of the received sub command.

The following table shows the information which the module transmits onto the CAN.

Sub command	Activate time for TxId	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$00	TxId 1	$04	$00	tx_act1		-	-	-	-
$01	TxId 2	$04	$01	tx_act2		-	-	-	-
$02	TxId 3	$04	$02	tx_act3		-	-	-	-
:	:	$04	:	:		-	-	-	-
n	TxId (n+1)	$04	n	tx_act(n+1)		-	-	-	-

(-) Byte is not transmitted

Table 3.5.4: Transmitting the actual Tx activate times

Explanation of the bytes transmitted by the module:

Byte 1... The first byte always returns the contents of the received command byte (here always $04).

Byte 2... The second byte returns the contents of the received sub command. The sub command designates the selected identifier.

Byte 3,
Byte 4... In these two cells the actual Tx activate time is returned (see also 'parameter 1 and 2').

3.6 Process User Parameters

The user parameters are designed for module-specific applications. By this sub command it is possible to process up to 127 parameters with 16 bit width.

The user parameters are described in detail in the manual CAN-CBM-SIO/SIO4, **Manual of the User-specific Software**'.

The assignment of the user parameters can be taken from the module-specific software.

By the command 'set user parameters' the parameters on the module are set. The command 'request user parameters' makes the module return the parameter contents on the Tx identifier CTxId which had been interchanged in the request.

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Function	Command	Sub command	Not used	Module no.	Parameter 1	Parameter 2
Value	Request: $06	$00...$7F	Always allocate with $00	Selected module no. $01...$FF	CTxId $0000...$07FF	
	Set: $86				Para $0000...$FFFF	

Table 3.6.1: Bytes of the command set/request user parameters

Command... By the command $06 the parameter data are requested by the module. After this request the module transmits the information on the identifier 'CTxId' of byte 5 and 6.
By the command $86 the parameters are set.

Sub command... By the sub command the desired parameter is selected.

Parameter 1 and 2... By these parameters it is reported to the module at a <u>request</u> <u>command</u> onto which CAN Id (CTxId) it should transmit the requested reply.
The module transmits <u>only once</u> on this identifier. The identifier is not stored on the module. If another transmission is desired, another INIT Id with the corresponding parameters has to be transmitted to the module again.

At a <u>set command</u> these two bytes contain the 16 bits of the parameter which should be interchanged. The possible parameters have got numbers which correspond to the value of the corresponding sub command. A maximum of 127 ($7F) parameters is possible.

Reply of the modules to the command 'request user parameters':

The parameter returned by the module is selected by the corresponding sub command.

The following table shows the information which is transmitted to the CAN by the module.

Sub command	Parameter	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
$00	1	$06	$00	Para0		-	-	-	-
$01	2	$06	$01	Para1		-	-	-	-
...	...	...	...	...	-	-	-	-	-
$7F	127	$06	$7F	Para7F		-	-	-	-

(-) Byte is not transmitted

Table 3.6.2: Module transmits parameter contents

Explanation of the bytes transmitted by the module:

Byte 1... The first byte returns the contents of the received command byte (here always $06).

Byte 2... The second byte returns the contents of the received sub command byte.

Byte 3,4:
Para... In these cells the actual parameter contents are returned:

Output	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
Bits of the parameter	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Allocation of the parameter to the bytes 5 and 6	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
	Byte 5								Byte 6							

Table 3.6.3: Allocation of the parameter bits to the outputs

3.7 Service Request

By this command it is possible to inquire the error condition of the *esd*-CAN modules. It is possible to make various or individual modules return a transmission if they show the error status 'error on CAN' or the status bit 'new on bus' is active.
The message consists of a Tx frame which has got no data.

The parameters of the modules remain unchanged when this command is requested.

The command can be used to find out which modules operate faultless on the bus and which modules have got transmission problems.

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Function	Command	Sub command 1	Sub command 2	Not used	Parameter 1	Parameter 2
Value	$7F	Module no_LOW $00...$FF	Module no_HIGH $00...$FF	Always allocate with $00	CTxId $0000...$07FF	

Table 3.7.1: Bytes of the command service request

<u>Explanation of the bytes transmitted to the module:</u>

Command... This command has got the value $7F.

Sub command1,
Sub command2... By the sub commands the modules are selected which should be checked. The modules are addressed by their module numbers.
To select an area with various modules, in sub command 1 the lowest and in sub command 2 the highest module no. of the desired area has to be entered. The limit values are included. If only one module should transmit a status message, the same module no. has to be entered for both sub commands.

Byte 4... In contrast to most of the other commands, here byte 4 and not byte 3 should be allocated with '$00'.

Parameter 1,
Parameter 2... In byte 5 and byte 6 the Tx identifier is interchanged on which the module should transmit a reply.

Reply of the module to the command 'service request':

If a module receives this command, and if a short term CAN error existed (bit 'error on CAN' = 1) or/and if the bit 'new on bus' is active, the module transmits a message with the length 0 on the received Tx identifier. No bytes are transmitted.

TxId	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
CTxId	-	-	-	-	-	-	-	-

(-) Byte is not transmitted

Table 3.7.2: Reply by CTxId

3.8 Supervisor Commands

In contrast to the other commands, supervisor commands are able to address individual or all modules in the CAN net.
By the module no. $00 all modules are selected. Therefore it is not recommendable to adjust the module no. $00 on a module.
The supervisor commands are not able to request parameters from the module.

The command RESET has got the same consequences as a 'power-on RESET'. It resets all parameters which are not stored in the EEPROM. After the RESET the parameters filed in the EEPROM are active, or, if the EEPROM data are defective or the EEPROM is not available, the default parameters.

If individual or all modules should be reset in a way that they operate with the default parameters after the RESET, this can be achieved by the command default RESET.

During a local RESET the message outputs are activated.

The command Supervisor Watchdog can be called by the Supervisor, to activate the watchdog functionality (precondition is that the watchdog time and the life time factor are greater than ‘0’). After the first activation the command has to be repeated within the time 'WDtime x WDLifeTimeFactor' otherwise a local RESET is generated at the module (refer to chapter 'System Parameter').

The command suspend module causes one or all modules on the CAN to 'freeze' the actual condition of their outputs (if available) and not transmit any messages (Tx transfers) to the CAN (exception: configuration reply).

Parameters and commands are received and filed by the module, but not evaluated: It is possible therefore, e.g., to set output data on the module without changing the output conditions at once.

If the module is activated again by the command 'continue', possibly available outputs are set immediately according to the last received data and the module operates with the last received parameters.

	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
Function	Command	Sub command	Not used	selected module no.	Parameter 1	Parameter 2
Value	$FF	$00 Trigger RESET	Always allocate with $00	$00 All modules selected $01..$FF One module selected	$AAAA - RESET	
		$01 Reserved			-	
		$02 Supervisor Watchdog			-	
		$03 Default RESET			$AAAA - Default RESET	
		$04 Suspend/continue module			$5A5A - Suspend $A5A5 - Continue	
		$05 Reset CAN-error bit			-	

Table 3.8.1: Bytes of the supervisor commands

Explanation of the bytes transmitted to the module:

Command... The supervisor commands all have got the value $FF.

Sub command... The sub command selects the corresponding supervisor command:

Sub command	Function
$00	Reset module
$01	Reserved
$02	Supervisor Watchdog
$03	Reset module and activate the default parameters
$04	Suspend module
$05	Reset CAN error bit

Table 3.8.2: Selection of the supervisor commands by the sub command

Module no. ... Here the desired actual module no. is entered. The setting of individual modules occurs by the module no. $01 to $FF.

At this command all *esd* modules connected to this CAN net are accessed at the same time by the module no. $00!

Parameter 1,.. Depending on the sub command here are parameters interchanged, which are specific for the function:

The commands RESET, default RESET, suspend/continue modules and resetting the CAN error bits are only carried out, if exactly the specified parameters are interchanged.

4. Examples for Parameterization

In this chapter the operation and the initialisation of a module should be clarified with some examples.

4.1 Setting the Tx Identifier TxId1

In this example it will be shown how the Tx identifier TxId1 is programmed on a module. The Tx identifier should get the value $1AA. The Tx identifier had not been programmed at this module so far.

Before the new identifier is set, the module should transmit a reply by the actual Tx identifier on the Tx identifier $567. The reply is requested by command $02.
As actual module no. the value $12 will be presumed here.

CAN Id	Byte 1 (command)	Byte 2 (sub command)	Byte 3 (always $00)	Byte 4 (module no.)	Byte 5 (CTxId-H)	Byte 6 (CTxId-L)
$700 =INIT Id	$02	$00	$00	$12	$05	$67

Table 4.1.1: Request of the Tx identifier

Byte 7 and 8 are not needed for this command.

After this INIT Id had been transmitted to the module, the module responds on the Tx identifier $567:

CAN Id	Byte 1 (returns command)	Byte 2 (returns sub command)	Byte 3 (returns TxId1-H)	Byte 4 (returns TxId1-L)
$567	$02	$00	$00	$92

Table 4.1.2: Reply of the Tx identifier

Byte 5 to 8 are not transmitted.

As can be seen from byte 3 and 4 the module has actually got the Tx identifier TxId1 $092.

Examples

Now the Tx identifier TxId1 (=$1AA) is allocated to the module:

CAN Id	Byte 1 (command)	Byte 2 (sub command)	Byte 3 (always $00)	Byte 4 (module no.)	Byte 5 (TxId-H)	Byte 6 (TxId-L)
$700 =INIT Id	$82	$00	$00	$12	$01	$AA

Table 4.1.3: Setting the Tx identifier TxId1

Byte 7 and 8 are not needed for this command.

The Tx identifier TxId1 of the module is now assigned with the value $1AA. The new identifier is active immediately after it has been received by the module.

4.2 Restoring the Default Parameters

This example should show how a module whose default parameters have been changed during an initialisation is put back into the original condition.

There are various ways to reactivate the default parameters.

4.2.1 ...if the actual module no. is unknown:

If only on this *esd*-CAN module the default parameters should be restored, you have to proceed as follows:

The coding switches of the module have to be set to $00 and a RESET has to be triggered.

Because the module no. is unknown, the RESET is triggered by interrupting the supply voltage of the module or the voltage at the EMERGENCY-STOP outputs or by triggering a global module RESET on the CAN (RESET all modules).

This global RESET resets all modules. But the default parameters are only restored on those modules whose coding switches are adjusted to $00.

CAN Id	Byte 1 (command)	Byte 2 (sub command)	Byte 3 (always $00)	Byte 4 (RESET type)	Byte 5	Byte 6
$700 =INIT Id	$FF	$00	$00	$00 (global)	$AA	$AA

Table 4.2.1: RESET of all *esd* modules of the CAN

The bytes 7 and 8 are not needed for this command.

If on all *esd*-CAN modules on the CAN the default parameters should be reactivated, the global command 'default RESET' is given to the bus:

CAN Id	Byte 1 (command)	Byte 2 (sub command)	Byte 3 (always $00)	Byte 4 (RESET type)	Byte 5	Byte 6
$700 =INIT Id	$FF	$03	$00	$00 (global)	$AA	$AA

Table 4.2.2: Default RESET of all *esd* modules of the CAN

The bytes 7 and 8 are not needed for this command.
After this RESET the module starts again with the default parameters. By coding switches the desired identifier can be determined again.

4.2.2 ...if the module no. is known:

If the module no. is known it is possible to restore the default parameters by the command 'default RESET' at the desired module. The module no. of the module is presumed with $12 here, again:

CAN Id	Byte 1 (command)	Byte 2 (sub command)	Byte 3 (always $00)	Byte 4 (RESET type)	Byte 5	Byte 6
$700 =INIT Id	$FF	$03	$00	$12 (selective)	$AA	$AA

Table 4.2.3: Default RESET of an *esd* module on the CAN

The bytes 7 and 8 are not needed for this command.