CAN-CBM-COM1

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

CANopen Software Manual

Manual file:	I:\texte\Doku\MANUALS\CAN\CBM\CBM_COM1\Englisch\COM1_CANopen_12S.en9
Date of print:	18.11.2003

Software described:	CANopen
Revision:	V1.ODJ

Changes in the chapters

The changes in the manual below affect changes in the **firmware** as well as changes in the **description** of the facts only.

Chapter	Changes versus previous version
5.	CAN identifier assignment corrected.
-	-

N O T E

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

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

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

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

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

USA / Canada
esd
PMB 292
20423 State Road 7 #F6
Boca Raton, Florida 33498-6797
USA

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

Contents

Page

1. General

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

A complete CANopen description is too extensive for the purpose of this manual.
Further information can therefore be taken from the CAL / CANopen specification (DS-xxx).

1.1 Definition of Terms

COB ...	Communication Object	
Emergency-Id...	Emergency Data Object	
n/a ...	not available	
NMT...	Network Management (Master)	
Rx...	receive	
SDO...	Service Data Object	
Sync...	Sync(frame) Telegram	synchronisation identifier
tbd ...	to be defined	data has yet to be defined
Tx...	transmit	
n.a.	not applicable	

1.2 NMT Boot-up

1.2.1 Minimum Capability Device Boot-Up

The CAN-CBM-COM1 module can be initialized via the 'Minimum Capability Device'-boot-up, described in the CiA-Draft Standard 301 [2] in chapter 9.4.2

Usually, only a telegram to switch from *pre-operational* status into *operational* status after power-on is required. For this, you have to transmit for example the 2-bytes telegram '0100_h' (= Start Remote Node all Devices) to the CAN identifier '0000_h'.

1.3 CANopen Object Directory

The object directory is mainly a (sorted) group of objects which can be accessed via the network. Each object in this directory is addressed by a 16-bit index. This index is given in hexadecimal form in the object directories.

The index can be a 16-bit parameter according to the CANopen specification (CiA-Draft DS-301) or a manufacturer-specific code. By means of the MSBs of the index the object class of the parameter is determined.

Part of the object directory are among others:

Index [HEX]	Object	Example
0001 ... 009F	definition of data types	
1000 ... 1FFF	Communication Profile Area	1001_h = error register
2000 ... 5FFF	Manufacturer Specific Profile Area	2800_h, 14_d = Tx-Cycle Time
6000 ... 9FFF	Standardised Device Profile Area	not used for CAN-CBM-COM1
A000 ... FFFF	reserved	

PDOs (Process Data Objects)
PDOs are used to transmit the process data.
In the 'Receive'-PDO process data is received by the CAN-CBM-COM1 module.
In the 'Transmit'-PDO the CAN-CBM-COM1 module transmits data in the CANopen network.

The asynchronous mode of transmitting the PDOs is defined by 'PDO transmission type' in the communication parameter of the according process data object.
The synchronous mode of transmitting is not supported at the moment.

1.4 Communication Parameters of the PDOs

The communication parameters of the PDOs (according to DS-301, chap.10.1.2) are transmitted as SDO (Service Data Objects) on ID '**600_h + Node-ID**' (Request). The receiver acknowledges the parameters on ID '**580_h + Node-ID**' (Response).

The **Node-ID** (module No.) is configured via coding switches (SW110, SW111). Please refer to the hardware manual of the CAN-CBM-COM1 module for a detailed description of possible configurations.

1.4.1 Accessing the Communication Parameters via SDO Telegrams

The SDOs (Service Data Objects) are used to access the object directory of a device.
An SDO is therefore a 'channel' to access the parameters of the device. Access via this channel is possible in *operational* and *pre-operational* status in the CAN-CBM-COM1 module.

This chapter does not describe all possible, but only some important modes of access in the CAN-CBM-COM1 module.
Definitions for the modes of access can be taken from CiA DS-202-2 (CMS-Protocol Specification, chap. 6: CMS Multiplexed Domain Protocols).

An SDO is structured as follows:

Identifier	command code	index (low)	index (high)	sub-index	LSB	data field		MSB
Example:								
600_h + Node-ID	23_h (write)	00_h (Index = 1400_h) (Receive-PDO-Comm-Para)	14_h	01_h (COB-def.)	$7F_h$	04_h	00_h	00_h
					COB = $047F_h$			

Identifier
The parameters are transmitted on ID '**600_h + Node-ID**' (Request).
The receiver acknowledges the parameters on ID '**580_h + Node-ID**' (Response).

Command code
The command code transmitted consists of the command specifier and the length, among others. Frequently used combinations are, e.g.:

$40_h = 64_d$: Read Request, i.e. a parameter is to be read
$23_h = 35_d$: Write Request with 32 bits data, i.e. a parameter is to be written.

The CAN-CBM-COM1 module responds to each received telegram with a response telegram. The response telegram can contain the following command codes:

$43_h = 67_d$:	Read Response,	this telegram contains the desired parameter
$60_h = 96_d$:	Write Response,	i.e. a parameter has been successfully configured
$80_h = 128_d$:	Error Response,	i.e. the CAN-CBM-COM1 module reports a communication error.

Frequently used command codes

The following table gives an overview of frequently used command codes. The command frames always have to have 8 data bytes. Notes on the syntax and further command codes can be taken from CiA DS-202-2 (CMS-Protocol Specification, chap. 6: CMS Multiplexed Domain Protocols).

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

Error Codes of the SDO Domain Transfer

The following error codes might occur (according to CiA Work Draft WD-301, table 20):

Error code [HEX]	CAN-CBM-COM1 designation	Explanation
0x05040001	SDO_CS_UNKNOWN	wrong command specifier
0x06010002	SDO_READ_ONLY	no write access
0x06020000	SDO_WRONG_INDEX	wrong index
0x06070010	SDO_WRONG_LENGTH	wrong number of data bytes
0x06070012	SDO_PARA_TO_LONG	length of service parameter is too long
0x06070013	SDO_PARA_TO_SHORT	length of service parameter is too short
0x06090011	SDO_WRONG_SUBIND	wrong sub-index
0x08000000	SDO_OTHER_ERROR	undefined cause of error

Index, Sub-Index
Index and sub-index will be described in chapter 'Communication Profile Area'.

Data field
The maximum 4-bytes long data field is generally structured following the rule 'LSB first, MSB last'. The LSB is **always** in 'Data 1'.
In 16-bits values the MSB (bits 8...15) is always in 'Data 2', and in 32-bits values the MSB (bits 24...31) is in 'Data 4'.

1.4.2 Non-volatile Storage of Parameters to EEPROM

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

The storage mode is shown in the content of the object 1010_h:
Bit 1 of object 1010_h, sub-index 1 is not set, i.e the CAN-CBM-COM1 module does not save the configuration automatically. The stortage must be initiated by writing the character string 'save' (73_h 61_h 76_h 65_h, order from CAN telegram) to object 1010_h, sub-index 1_d.

Read request for the current memory mode:

identifier	command code	index (low)	index (high)	sub-index	data 1	data 2	data 3	data 4
600_h+ Node-ID	40_h (read request)	10_h	10_h	01_h	00_h	00_h	00_h	00_h
					Data are not evaluated			

Response of CAN-CBM-COM1 module (default setting):

identifier	command code	index (low)	index (high)	sub-index	data 1	data 2	data 3	data 4
600_h+ Node-ID	$4F_h$ (read response, 1 byte)	10_h	10_h	01_h	01_h	00_h	00_h	00_h
					Storing of the parameter only at request			

Command for the storage of the parameters:

identifier	command code	index (low)	index (high)	sub-index	data 1	data 2	data 3	data 4
600_h+ Node-ID	23_h (write, 4 bytes)	10_h	10_h	01_h	73_h 's'	61_h 'a'	76_h 'v'	65_h 'e'
					= Storing of the parameters			

1.4.3 Restoring of Default Status of the parameters

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

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

2. Communication Profile Area

2.1 Terms and Abbreviations Used

The following terms will be used in the tables to describe the communication parameters:

PDO-Mapping	PDO-Mapping is available for this sub-index of the PDO
Save to EEPROM	the value of the Parameter is stored to the EEPROM, if the command 'save' is called (page 8)
Access Mode	permissible modes of access to this parameter: ro... read_only This parameter can only be read. Write access triggers an error code. const.... constant This parameter cannot be changed by the user. It can be read. Write access triggers an error code. rw... read&write This parameter can be read or configured.
Value Range	value range of the parameter
Default-Value	default configuration of parameter when module is shipped
Name/Description	name and short description of parameter

This page is intentionally left blank.

3. CANopen Profile

3.1 Communication Profile Area

3.1.1 Overview of Communication Parameters

The format of communication parameters can be taken from CiA DS-301, chap. 9.1. The module only supports the communication parameters shown in the table below.

Index [HEX]	Name	Sub-index	Data Type	Access Mode	Default Value
1000	Device Type	-	Unsigned32	ro	00800000_h
1001	Error Register	-	Unsigned8	ro	Error code
1002	Status Register	-	Unsigned32	ro	status
1003	Predefined Error	0, (1...16)	Unsigned32	ro	-
1004	Number of PDOs	0, 1, 2	Unsigned32	ro	0, 2, 0
1008	Manufacturer's Device Name	-	Visible String 1*)	ro	'esd_CBM-COM1 1*)
1009	Manufacturer's Hardware Version	-	Visible String 1*)	ro	0 1*)
100A	Manufacturer's Software Version	-	Visible String 1*)	ro	e.g.:V1.ODJ/ CANopen 1*)
100B	Node-ID	-	Unsigned32	ro	f(Coding switch)
100E	Node Guarding ID	-	Unsigned32	ro	700_h + Node-ID
100F	Number of SDOs	-	Unsigned32	ro	1
1010	Store Parameter	0, 1	Unsigned32	rw	--
1011	Restore Parameter	0, 1	Unsigned32	rw	--
1016	Consumer Heartbeat Time	0, 1, ...16	Unsigned32	rw	--
1017	Producer Heartbeat Time	0	Unsigned16	rw	10.000_d
1018	Identity Object	0, 1, ...4	Unsigned32	ro	see page 24
1020	Verify Configuration	0, 1, 2	Unsigned32	ro	--
1029	Error Behaviour Object	0, 1	Unsigned8	rw	0
1400 -1403	Receive PDO Communication Parameter	0, 1, 2, 3	PDOCommPar	rw	--
1600 -1603	Receive PDO-Mapping Parameter	0, 1	PDOMapping	rw	--
1800 -1803	Transmit PDO Communication Parameter	0, 1,...5	PDOCommPar	rw	--
1A00 -1803	Transmit PDO-Mapping Parameter	0, 1	PDOMapping	rw	--

ro - Read Only, rw - Read/Write
1*) depending on the software or hardware revision

3.1.2 Error Register 1001_h

The CAN-CBM-COM1 module uses the error register to indicate error codes.

INDEX	**1001_h**
Name	*error register*
Data Type	unsigned 8
Default Value	No

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

Bit	Meaning
0	*generic*
1	-
2	-
3	-
4	-
5	-
6	-
7	*manufacturer-specific error*

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

The following messages are possible:

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

3.1.3 Manufacturer Status Register 1002_h

INDEX	**1002_h**
Name	*manufacturer status register*
Data Type	unsigned 32
Default Value	No

The bits of the status register are set as described below:

Register bit (assembler)	Description	Level Assignment	
D25...D8	Always returned as '0'	0	(always '0')
D7	New on bus	0 1	reserved, may be returned as '0' or '1'
D6	Default wake up	0 1	normal start module is started with default parameters
D5	I²C busy	0 1	no access to EEPROM local SW access to I²C-EEPROM
D4	manufacturer-specific error (corresponding with bit 7 of object 1001_h)	0 1	no error manufacturer specific error is detected
D3	I²C error	0 1	no I²C-error I²C-error
D2	Error on CAN	0 1	no CAN-bus error detected CAN-bus error
D1	Suspend-Bit	0 1	module is in state 'operational' module is in state 'preoperational' or 'stopped'
D0	Power-up reset	0 1	last reset is not generated by power-up last reset is generated by power-up

3.1.4 Pre-defined Error Field 1003_h

INDEX	**1003_h**
Name	*pre-defined error field*
Data Type	unsigned 32
Default Value	No

The *pre-defined error field* provides an error history of the errors that have been signalled via the Emergency Object.
Sub-index 0 contains the number of actual errors that are recorded in the array.
A new error is stored at sub-index 1. If a new error occurs, then the previous error gets stored to sub-index 2 and the new error is stored to sub-index 1 and so on. So the older errors move down the list.

The error field works like a ring buffer. If it is full and a new error occurs, the older errors move down the list and the oldest error is deleted.
The CAN-CBM-COM1 module supports a maximum of 16 error entries. At the appearance of the 17th error the oldest error is deleted.

Writing a '0' to sub-index 0 deletes the entire error list. Higher values are not allowed. '0' is the only permitted write access for this object.

With every new entry in the list the CAN-CBM-COM1 module sends an emergency frame.

Index [Hex]	Sub-index	Description	Value Range [Hex]	Default	Data Type	Access Mode
1003	0	*no_of_errors_in_list*	0, 1...10	-	unsigned 8	rw
	1	*error-code n*	0...FFFFFFFF	-	unsigned 32	ro
	2	*error-code (n-1)*	0...FFFFFFFF	-	unsigned 32	ro
	:	*:*	:	:	:	ro
	16	*error-code (n-15)*	0...FFFFFFFF	-	unsigned 32	ro

Parameter description:

no_of_errors_in_list This parameter contains the actual number of the error codes written down on the list.
n = number of the error occurred last
Set *no_of_errors_in_list* to '0' to delete the error list.
If *no_of_errors_in_list* ≠0, then error register (object 1001_h) is set

error-code x The 32 bit error code consists of the CANopen-Emergency-Error-Code (DS-301, Table 21) and the **esd** specific error codes (Manufacturer-Specific Error Field).

error code unsigned32		error description	meaning
CANopen error code [Hex]	manufacturer-specific error field [Hex]		
8130	Node-ID ($1...127_d$)	heartbeat_err	heartbeat error with detail of the Node-ID of the failed heartbeat generator

3.1.5 Manufacturer's Device Name 1008_h

INDEX	**1008_h**
Name	*manufacturer's device name*
Data Type	visible string
Default Value	string: 'esd_CBM-COM1'

Please refer to CiA DS-202-2 (CMS-Protocol Specification) for a detailed description of the Domain Upload.

3.1.6 Manufacturer's Hardware Version 1009_h

INDEX	**1009_h**
Name	*manufacturer's hardware version*
Data Type	visible string
Default Value	ASCII-string: '0'

This object contains the manufacturer hardware version description. The hardware version is set on the module at production.

3.1.7 Manufacturer's Software Version $100A_h$

INDEX	**$100A_h$**
Name	*manufacturer's software version*
Data Type	visible string
Default Value	string: e.g.: 'V1.ODJ/CANopen' 1*)

1*) depending on the software revision

Similar to reading the Manufacturer Device Name, the software version is read via the Domain Upload Protocol. A detailed description of the upload can be taken from CiA DS-202-2 (CMS-Protocol Specification).

3.1.8 Node Guarding Identifier $100E_h$

The module only supports 11-bit identifiers. The parameter can only be read.

INDEX	**$100E_h$**
Name	*node guarding identifier*
Data Type	unsigned 32
Default Value	700_h + Node-ID

Structure of parameter *node guarding identifier* :

Bit No.	Value	Meaning
31 (MSB) 30	-	reserved
29...11	0	always 0, because 29-bit IDs are not supported
10...0 (LSB)	0 x	bits 0...10 of Node Guarding Identifier

The Node guarding identifier is used for the heartbeat-frame. The heartbeat function is described on page 21.

3.1.9 Store Parameters 1010_h

This object stores the parameters to the EEPROM. Only the command 'Save all Parameters' is supported.

Storing is only executed when the specific signature (see below) is written to the appropriate sub-index. On read access to the appropriate sub-index the device provides information about its storage functionality (for further information see CiA DS-301).

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

Index [Hex]	Sub-index	Description	Value Range [Hex]	Data Type	Access Mode
1010	0	*number_of_entries*	3_h	unsigned 8	ro
	1	*save_all_parameters*	no default, write: 65 76 61 73_h (= ASCII: 'e' 'v' 'a' 's')	unsigned 32	rw

3.1.10 Restore Default Parameters 1011_h

With this object the default values of parameters are restored. The individual choice of parameters, stored to the EEPROM, will get lost. Only the command 'Restore all Parameters' is supported.

In order to avoid the restoring of default parameters by mistake, restoring is only executed when the specific signature (see below) is written to the appropriate sub-index.
On read access to the appropriate sub-index the device provides information about its default parameter restoring capability (for further information see CiA DS-301).

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

Index [Hex]	Sub-index	Description	Value Range [Hex]	Data Type	Access Mode
1011	0	*number_of_entries*	3	unsigned 8	ro
	1	*load_all_default_parameters*	no default, write: 64 61 65 $6C_h$ (= ASCII: 'd' 'a' 'o' 'l')	unsigned 32	rw

3.1.11 Consumer Heartbeat Time 1016_h

INDEX	**1011_h**
Name	*consumer heartbeat time*
Data Type	unsigned 32
Default Value	No

The heartbeat function can be used for mutual monitoring of the CANopen modules (particularly for recognizing connection failures). Unlike Node Guarding/Life Guarding the heartbeat protocol defines an error control service without need for remote frames.

Heartbeat Function
A heartbeat *producer* transmits a heartbeat message on the CAN bus via the Node-guarding identifier (see Object $100E_h$) cyclically. One or more heartbeat *consumer* receive the indication. The relationship between *producer* and *consumer* is configurable via the object dictionary.
The heartbeat *consumer* guards the reception of the heartbeat within the heartbeat consumer time.
The guarding of the heartbeat *consumer* starts after the reception of the first heartbeat-CAN frame with data(0) ≠ '0'.
If the heartbeat is not received within the heartbeat consumer time a heartbeat event will be generated.
At the CAN-CBM-COM1 module the heartbeat event generates a heartbeat error.

The CAN-CBM-COM1 module supports a *heartbeat consumer* the monitoring of one heartbeat producer.

Index [hex]	Sub-index [Dec]	Description	Value Range [Hex]	Default [Dec.]	Data Type	Access Mode
1016_h	0	*number_of_entries*	1	16	unsigned 8	ro
	1	*consumer-heartbeat_time_1*	0... 00 7F FF FF_h	0	unsigned 32	rw

Description of the parameter *consumer-heartbeat_time_x*:

	consumer-heartbeat_time_x		
Bit	31 24	23 16	15 0
Description	*heartbeat_mask* (unsigned 8)	*Node-ID* (unsigned 8)	*heartbeat_time* (unsigned 16)

Node-ID Node-Id of the heartbeat producer module that has to be controlled.

heartbeat_mask

Evaluation of the CANopen status of the heartbeat producer.
If a heartbeat producer generates a local reset during operation and goes on transmitting heartbeat messages cyclically, the heartbeat consumer might not be able to notice that. This could happen, if the consumer-heartbeat time is higher than the time, the producer needs to transmit heartbeat messages after the RESET.

The parameter *heartbeat_mask* prevents this. If the parameter is set correspondingly, the CAN-CBM-COM1 module as heartbeat consumer controls the CANopen status of the producer, the producer transmits in the heartbeat message

If the heartbeat producer is not in the desired CANopen status, the received heartbeat will be ignored and a heartbeat error will be generated.

heartbeat_mask register bit	Description	level assignment	
D7	Others_Mask	0 1	no masking ignore heartbeat, if data $\neq 00_h$ and $\neq 04_h$ and $\neq 05_h$ and $\neq 7F_h$
D6 D5 D4	reserved, set always to '0'	0	(always)
D3	Operational_Mask	0 1	no masking ignore heartbeat, if data = 05_h (operational)
D2	Stopped_Mask	0 1	no masking ignore heartbeat, if data = 04_h (stopped)
D1	Preoperational_ Mask	0 1	no masking ignore heartbeat, if data = $7F_h$ (preoperational)
D0	Bootup_Mask	0 1	no masking ignore heartbeat, if data = 00_h (bootup)

In default setting is *heartbeat-mask* = 00_h, i.e. the heartbeat protocol works as described in DS-301.
The setting *heartbeat-mask* = 87_h allows the transmission of heartbeats only, if the heartbeat producer is in status operational.

heartbeat_time

If the heartbeat producer does not send a message at the Node Guarding-ID within the time *heartbeat_time*, a heartbeat event will be generated

The consumer *heartbeat_time* has to be higher than the corresponding producer *heartbeat_time* configured on the device producing this heartbeat.

3.1.12 Producer Heartbeat Time 1017_h

INDEX	**1017_h**
Name	*producer heartbeat time*
Data Type	unsigned 16
Default Value	10.000 ms

The *producer* heartbeat time defines the cycle time of the heartbeat. The time has to be a multiple of 1 ms. The heartbeat function is described on page 21.

The *producer* heartbeat time has to be set to '0' if it is not used.

Index [Hex]	Sub-index	Description	Value Range [Hex]	Default Value	Data Type	Access Mode
1017	0	*producer_heartbeat_time*	0...FFFF	10.000 ms	unsigned 16	rw

Parameter description:

producer_heartbeat_time Cycle time of the heartbeat producer to send the heartbeat at the Node guarding-ID (see object $100E_h$).
The consumer heartbeat time has to be higher than the corresponding *producer_heartbeat_time* configured on the device producing this heartbeat.

3.1.13 Identity Object 1018_h

INDEX	**1018_h**
Name	*identity object*
Data Type	unsigned 32
Default Value	No

The identify object returns general information about the CAN module.

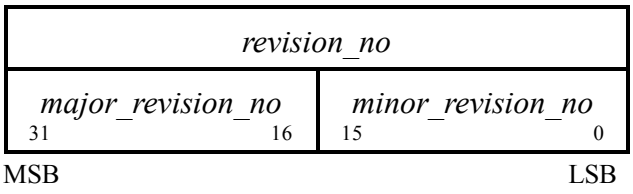
Index [Hex]	Sub-index	Description	Value Range [Hex]	Default Value [Hex]	Data Type	Access Mode
1018	0	*no_of_entries*	1	1	unsigned 8	ro
	1	*vendor_id*	0...FFFFFFFF	0000 0017	unsigned 32	ro
	2	*product_code*	0...FFFFFFFF	2284 1030	unsigned32	ro
	3	*revision_number*	0...FFFFFFFF	'Software revision'	unsigned32	ro
	4	*serial_number*	0...FFFFFFFF	'Hardware revision'	unsigned32	ro

Parameter description:

vendor_id This parameter returns the **esd**-Vendor-ID. The value is always 00000017_h.

product_code This parameter returns the **esd**-order number of the module.
Example:
In the value '2**2 841.03**0_h' the order number 'C.2841.03' is coded.

revision_number This parameter returns the software version. The upper two bytes return the revision numbers of the major changes according to DS-301 and the lower two bytes return the revision number of minor changes.



The software version contains of the following sections:
- Kernel Software (described by the parameters '*lev*' (level) and '*rev*' (revision))
- Application Software (described by the parameter '*ext*' (extension))
- Protocol Software (described by the parameter '*plev*' (protocol level))

The minor revision number is a fixed Int-16 variable between 0000_h and $FFFF_h$. The returned software version is coded for 'major' and for 'minor'-revision number as follows:

revision_no =xxyy
and is determined by the numbers of the ASCII-code of the characters
with xx = (*lev* - '0') · 26 + (*rev* - 'A')
yy = (*plev* - 'H') · 16 + (*ext* - 'A')

Example:
If the actual software has the revision:
Kernel: *lev* = '1' *rev* = 'M'
Application: *ext* = 'A'
Protocol: *plev* = 'H'

the returned value for *revision_no* will be 26 00 00 00_h.

serial_number

This parameter returns the serial number code of the PCB-hardware.
The higher two bytes of *serial_no* contain the letters that code the production lot. They return the ASCII-code of the letters with the most significant bit set to '1', to distinguish letters from numbers:
(ASCII-code) + 80_h = returned bytes

The following characters code the number of each module as BCD-value.

Example:
If the value 'C1 C1 1234_h ' is returned, this will mean the hardware serial number code 'AA 1234'. This value has to match with the value labelled at the module.

3.1.14 Verify Configuration 1020_h

INDEX	**1020_h**
Name	*verify configuration*
Data Type	unsigned 32
Default Value	No

In this parameter the date and the time of the last configuration can be stored. This can be used to check if this is the correct configuration version.

Index [Hex]	Sub-index	Description	Value Range [Hex]	Default Value	Data Type	Access Mode
1020	0	*no_of_entries*	2	2	unsigned 8	ro
	1	*configuration_date*	0...FFFFFFFF	0	unsigned 32	rw
	2	*configuration_time*	0...FFFFFFFF	0	unsigned 32	rw

Parameter description:

configuration_date Date of the last configuration.
The value returns the number of days since the 01.01.1984.

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

3.1.15 Error Behaviour Object 1029_h

INDEX	**1029_h**
Name	*error behaviour object*
Data Type	Unsigned 8
Default Value	No

If an error event occurs (e.g. heartbeat error), the module will switch to the state that is defined in the parameter *communication_error.*

Index [Hex]	Sub-index	Description	Value Range [Hex]	Default Value	Data Type	Access Mode
1029	0	*no_of_error_classes*	1	1	unsigned8	ro
	1	*communication_error*	0, 1, 2	0	unsigned8	rw

Parameter description:

no_of_error_classes Number of error classes (here always '1')

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

3.1.16 Receive PDO Communication Parameter 1400-1403$_h$

Object 'Receive PDO Communication Parameter 1400$_h$-1403$_h$' defines the features of a receive PDO (Rx-PDO). The CAN-CBM-COM1 module uses maximum two Rx-PDOs.

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

INDEX	**1400$_h$-1403$_h$**
Name	*receive PDO parameter*
Data Type	PDOCommPar

Index [Hex]	Sub-Index [Dec.]	PDO-mappable	Save to EEPROM	Access Mode	Data Type	Default	Description
1400	0	no	no	const.	Unsigned8	3$_h$	number of entries
	1	no	yes	rw	Unsigned32	200$_h$ + Node-ID	COB-ID used by PDO
	2	no	no	const.	Unsigned8	255$_d$	transmission type
	3	no	no	const.	Unsigned16	0	inhibit time
1401	0	no	no	const.	Unsigned8	3$_h$	number of entries
	1	no	yes	rw	Unsigned32	80000000$_h$ + 300$_h$ + Node-ID	COB-ID used by PDO (only for CANlink protocol)
	2	no	no	const.	Unsigned8	255$_d$	transmission type
	3	no	no	const.	Unsigned16	0	inhibit time
1402	0	no	no	const.	Unsigned8	0	no entries
1403	0	no	no	const.	Unsigned8	0	no entries

Value range refer DS-301, table 10, 11.

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

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

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

3.1.17 Receive PDO-Mapping Parameter 1600_h-1603_h

Object 'Receive PDO-Mapping Parameter 1600_h-1603_h' changes the assignment of receive data to Rx-PDOs.

INDEX	**1600_h-1603_h**
Name	*receive PDO mapping*
Data Type	PDO-Mapping

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

Index [Hex]	Sub-index [Dec]	PDO-mappable	Save to EEPROM	Access Mode	Data Type	Default Value	Description
1600	0	no	no	const.	Unsigned8	1_h	number of entries
	1	no	no	const.	CANopen Map Struct	2880 00 40_h	Data CAN -> serial, byte 0...8
1601	0	no	no	const.	Unsigned8	2_h	number of entries
	1	no	no	const.	CANopen Map Struct	2882 00 08_h	Data CAN -> serial at CANlink protocol, byte 1
	2	no	no	const.	CANopen Map Struct	2882 01 08_h	Data CAN -> serial at CANlink protocol, byte 2
1602	0	no	no	const.	Unsigned8	0	no entries
1603	0	no	no	const.	Unsigned8	0	no entries

Value range refer DS-301, table 10, 11.

Note:	The $288x_h$-objects registered in the PDO-Mapping are not accessible via SDOs. They are used as dummies to correspond to the structure of the objects 1600_h, 1601_h, $1A00_h$ and $1A01_h$.

3.1.18 Object Transmit PDO Communication Parameter 1800_h-1803_h

This object defines the features of a transmit PDO.

INDEX	**1800_h-1803_h**
Name	*transmit PDO parameter*
Data Type	PDOCommPar

Index [Hex]	Sub-index [Dec.]	PDO-mappable	Save to EEPROM	Access Mode	Data Type	Default Value	Description
1800	0	no	no	const.	Unsigned8	5_h	number of entries
	1	no	yes	rw	Unsigned32	180_h + Node-ID	COB-ID used by PDO
	2	no	yes	rw	Unsigned8	255_d	transmission type
	3	no	no	const.	Unsigned16	0	inhibit time
	4	no	no	const.	Unsigned8	6	(CMS-Prio-Group)
	5	no	yes	rw	Unsigned16	100 *	Cyclic_Tx_Time [ms]
1801	0	no	no	const.	Unsigned8	5_h	number of entries
	1	no	yes	rw	Unsigned32	80000000_h + 280_h + Node-ID	COB-ID used by PDO (only at CANlink)
	2	no	yes	rw	Unsigned8	255_d	transmission type
	3	no	no	const.	Unsigned16	0	inhibit time
	4	no	no	const.	Unsigned8	6	(CMS-Prio-Group)
	5	no	yes	rw	Unsigned16	0	Cyclic_Tx_Time [ms]
1802	0	no	no	const.	Unsigned8	0	no entries
1803	0	no	no	const.	Unsigned8	0	no entries

Value range refer DS-301, table 10, 11.

Example: Definition of a new Transmit PDO with Index '1800_h' at CAN identifier '$047F_h$'

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

Byte	Name	Value	Content
0	Command	23	Write Request
1	Index Low	0	New PDO 1800_h in Intel-Notation
2	Index High	18	
3	Subindex	1	defines COB
4	Data 1	7F	COB $047F_h$ in Intel-Notation
5	Data 2	4	
6	Data 3	0	
7	Data 4	0	

3.1.19 Transmit PDO-Mapping Parameter $1A00_h$-$1A03_h$

Object 'Transmit PDO-Mapping Parameter $1A00_h$-$1A03_h$' can change the assignment of transmit data to Tx-PDOs.

INDEX	**$1A00_h$-$1A03_h$**
Name	*transmit PDO mapping*
Data Type	PDO-Mapping

Transmit PDO-Mapping Parameter

Index [Hex]	Sub-index [Dec]	PDO-mappable	Save to EEPROM	Access Mode	Data Type	Default Value	Description
1A00	0	no	yes	rw	unsigned8	1_h	number of entries
	1	no	no	const.	CANopen Map Struct	2881 00 40_h	Data serial -> CAN, byte 0...8
1A01	0	no	yes	rw	unsigned8	2_h	number of entries
	1	no	no	const.	CANopen Map Struct	2882 00 08_h	Data CAN -> serial at CANlink protocol, byte 1
	2	no	no	const.	CANopen Map Struct	2883 01 08_h	Data serial -> CAN at CANlink protocol, byte 2
1A02	0	no	yes	rw	unsigned8	0	number of entries
1A03	0	no	yes	rw	unsigned8	0	number of entries

Value range refer DS-301, table 10, 11.

Note:	The $288x_h$-objects registered in the PDO-Mapping are not accessible via SDOs. They are used as dummies to correspond to the structure of the objects 1600_h, 1601_h, $1A00_h$ and $1A01_h$.

3.2 Transmission Modes

3.2.1 Supported Transmission Modes According to DS-301, Table 55

Transmission Type	PDO transmission					Supported by CAN-CBM-COM1
	cyclic	acyclic	synchronous	asynchro-nous	RTR	
0	-	X	X	-	-	NO
1...240	X	-	X	-	-	NO
241...251	reserved					NO
252	-	-	X	-	X	NO
253	-	-	-	X	X	NO
254	-	-	-	X	X	NO
255	-	-	-	X	X	**YES**

The CAN-CBM-COM1 module only supports transmission type 255. It transmits Tx frames depending on the parameters *MinChar* and *MaxChar*, *TxByRxTimeout* and after receiving RTRs.

4. Data Transfer and PDO Assignment

4.1 Function Description of local Firmware

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

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

4.1.1 Data Transfer CAN -> Serial Interface

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

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

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

The data is transmitted from CAN via identifier RxPDO1 to the module. Up to 8 bytes ca be selected to be transmitted.

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

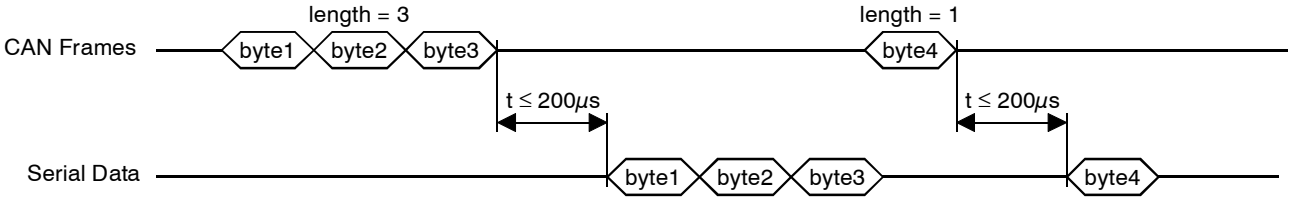


Fig. 4.1.1: Data transfer CAN -> serial interface (parameter *TxMinCharStart* = 0)

In addition, the CAN-CBM-COM1 module offers an RTR handshake at which the module informs the CAN partner via RTR, that the local ring buffer still can receive data. No data are sent any more if the ring buffer is full (see page 49).

The Parameter *MinChar* determines the minimum number of data bytes which are to be transmitted on the CAN within a CAN frame (see page 40).

The eight data bytes of the Tx-PDOs are shown below:

RX-PDO	Data Bytes							
	1	2	3	4	5	6	7	8
Rx-PDO1	0 ... 8 byte data CAN -> serial							

Table 4.1.1: Receiving data to be transmitted via Rx-identifier

4.1.2 Data Transfer Serial Interface-> CAN

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

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

The module offers possibilities to influence the last two factors via parameters *Inhibit-Time*, *MaxChar* and *MinChar*.

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

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

The data is transmitted by the module via TxPDO1 on the CAN.

The eight data bytes of the Tx-PDOs are:

Tx-PDO	Data Bytes							
	1	2	3	4	5	6	7	8
Tx-PDO1	0 ... 8 byte data serial -> CAN							

Table 4.1.2: Transmission of data received by serial interface via Tx-identifier

4.2 Device Profile Area

4.2.1 Parameter Overview 6000_h - 6200_h

The module does not support any objects in the area 6000_h - 6200_h

4.3 Manufacturer Specific Profile Area

4.3.1 Parameter Overview 2800_h - 2801_h

Index [Hex]	Object	Sub-index 0 [Dec]	Description
2800	Serial communication parameters	14	-
2810	Cyclic_String	0 ... 15	cyclic transmittable string

Table 4.3.1: Manufacturer-specific parameters

Note:	The $288x_h$-objects registered in the PDO-Mapping are not accessible via SDOs. They are used as dummies to correspond to the structure of the objects 1600_h, 1601_h, $1A00_h$ and $1A01_h$.

4.3.2 Serial Communication Parameter 2800_h

4.3.2.1 Overview

Index [Hex]	Sub-index [Dec]	Name/Description	Value Range	Default [Hex]	Data Type	Access	Save to EEPROM
2800	0	*number_of_entries*	14_d	0E	Unsigned8	ro	no
	1	*Inhibit-Time*	00 ...FF_h	14	Unsigned8	rw	yes
	2	*MaxChar_MinChar*	00...FF_h	11	Unsigned8	rw	yes
	3	*Serial_Baudrate*	00...FF_h	22 (9600 Baud)	Unsigned8	rw	yes
	4	*Serial_Mode*	00...FF_h	73	Unsigned8	rw	yes
	5	*TxMinCharStart*	00...FF_h	00	Unsigned8	rw	yes
	6	*Protocol*	00...83_h	80	Unsigned8	rw	yes
	7	*Handshake_Off*	00...FF_h	F6	Unsigned8	rw	yes
	8	*Handshake_On*	10...FF_h	0A	Unsigned8	rw	yes
	9	*TxByRxTimeout*	00...FF_h	00	Unsigned8	rw	yes
	10	*End_Char_1*	00...FF_h	0D	Unsigned8	rw	yes
	11	*End_Char_2*	00...FF_h	0A	Unsigned8	rw	yes
	12	*Special_Baudrates*	0000...$07FF_h$	0034	Unsigned16	rw	yes
	13	*RTR_T_Cycle_Time*	0000...$FFFF_h$	1388	Unsigned16	rw	yes
	14	*Tx_Cycle_Time*	0000...$FFFF_h$	0000	Unsigned16	rw	yes

Table 4.3.2: Parameters of object 2800_h

4.3.2.2 *Inhibit-Time*

Parameter name	***Inhibit-Time***
Object-index, Sub-index	2800_h, 01_d
Access	R/W
Data Type	unsigned8
Value Range	$00...FF_h$
Measurement Unit	ms
Default Value	14_h = 20 ms

Description

This Parameter determines the delay between two transmissions of serial data on the CAN bus.

The *Inhibit-Time* determines the time the CAN controller waits after the last successful transmission of CAN data, before it initiates another transmission. The delay is specified in [ms].
The default setting is $14 (= 20 ms).

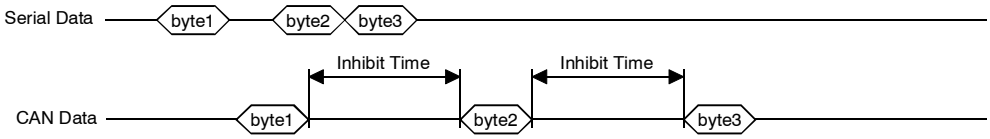


Fig. 4.3.1: Function of parameter *Inhibit Time* (Parameter *MaxChar* = 1)

4.3.2.3 MaxChar_MinChar

Parameter name	***MaxChar_MinChar***
Object-index, Sub-index	2800_h, 02_d
Access	R/W
Data Type	Unsigned8
Value Range	$00...FF_h$
Measurement Unit	n.a.
Default Value	11_h = every byte is sent

Description: The nibbles of this parameter determine the number of bytes from which a transmission to the CAN is to be started, and the maximum number of data bytes to be transmitted in a CAN frame. The higher nibble (*MaxChar*) determines the maximum number and the lower nibble (*MinChar*) determines the minimum number. Default setting is 11_h, i.e. each received byte is transmitted individually in a frame.

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

Table 4.3.3: Structure of the parameter *MaxChar_MinChar*

Transmission command:
MinChar is only evaluated as described above, if values between $1...8_h$ are specified. If values between 9_h and F_h are specified, the functionality of the parameter changes: In this case the local software evaluates the entry of characters which have been specified via parameter *End_Char1* (2800_h, 10_d) or *End_Char2* (2800_h, 11_d) as transmission command.

In default setting of parameter *End_Char1, 2* these are the flags 'Carriage Return' ($0D_h$) or 'Line Feed' ($0A_h$). As soon as one or both flags are detected, the data which has been received by the serial interface is transmitted to the CAN.
If 8 bytes have been received before the flags have been received, the transmission starts automatically.

You can also specify for the transmission, whether the flags are to be transmitted on the CAN together with the data or not. The following table shows the various options:

MinChar [HEX]	Transmission command	Flag transmission to CAN	Comments
1...8	-	-	At least 1 to 8 bytes are always transmitted. Parameter *MaxChar* is also evaluated.
9	<*Char_1*>	with <*Char_1*>	Data is transmitted after <*Char_1*> has been received. <*Char_1*> is also transmitted.
A	<*Char_2*>	with <*Char_2*>	As under '9', but with <*Char_2*>.
B	<*Char_1*>	without <*Char_1*>	As under '9', but with <*Char_1*> is not transmitted as well.
C	<*Char_2*>	without <*Char_2*>	As under 'B', but with <*Char_2*>.
D	<*Char_1*>	without <*Char_1*> and without <*Char_2*>	The end of the message can have <*Char_1*> and <*Char_2*>. The data is transmitted after <*Char_1*> has been received. Neither <*Char_1*> nor <*Char_2*> are transmitted as well.
E	<*Char_2*>	without <*Char_1*> and without <*Char_2*>	As under 'D', but with <*Char_2*>.
F	-	-	reserved

Table 4.3.4: Selection of transmission command via parameter *MinChar*

Examples: The examples below are representative for the CAN-CBM-COM1 being operated by parameters in default setting, i.e. *Char_1* = <Cr>, *Char_2* = <Lf>.

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

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

4.3.2.4 Serial_Baudrate

Parameter name	***Serial_Baudrate***
Object-index, Sub-index	2800_h, 03_d
Access	R/W
Data Type	Unsigned8
Value Range	$00...FF_h$
Measurement Unit	n.a.
Default Value	22_h = 9600 bits/s

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

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

Table 4.3.5: Structure of parameter *Serial_Baudrate*

With the baud rate data are transmitted (Tx) or received (Rx) on the serial interfaces. It is specified in 4 bits. The default setting is 9600 kbit/s for Rx- and Tx-baud rates. The physically attainable baud rate is limited by the hardware (please refer to hardware manual)

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

Via parameter *Special_Baudrates* (2800_h, 12_d) further baud rates can be set! If you want to do so, the value E_h has to be specified for parameters *Rx-Baudrate* and *Tx-Baudrate.*

Parameter *Rx-(Tx) Baudrate* [HEX]	**Baud rate** (set value) [bit/s]	Baud rate (realized values) [bit/s]
0	3.8400	3.8462
1	19200	19231
2	**9600**	9615
3	4800	4808
4	2400	2404
5	1200	1199
6	600	600
7, 8	300	300
9	7200	7246
A	14400	14286
B	28800	29412
C	(57600)	55556
D	(115200)	(125000)
E	variable baud rate	variable baud rate
F	76800	71429

Table 4.3.6: Setting of baud rate of the serial interface in 14 steps

4.3.2.5 Serial_Mode

Parameter name	***Serial_Mode***
Object-index, Sub-index	2800_h, 04_d
Access	R/W
Data Type	Unsigned8
Value Range	$00...FF_h$
Measurement Unit	n.a.
Default Value	73_h

Description: The bits of parameter *Serial_Mode* have got the following functions:

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

Table 4.3.7: Assignment of parameter *Serial_Mode*

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

Bit						Function
7, 6, 5	4	3	2	1	0	
RTS, CTS, Stop: see following tables	*Parity-Mode*		*Parity-Type*	*Bits per Character*		
	0	0	O/E*	1	1	Tx-Parity acc. to Bit ‘Parity Type’, no Rx-Parity, 1 Stop-Bit
	0	1	O/E*	1	1	Tx Parity and Rx-Parity acc. to Bit ‘Parity Type’, 1 Stop-Bit
	1	0	0	1	1	No Parity, evaluate 1 or 2 Stop-Bits
	1	0	1	1	1	Transmitter in multidrop mode, if *RTS-Mode* = ‘1’ (see page 45)
	1	1	a	1	1	Force Tx-Parity to value of ‘a’, 1 Stop-Bit

Table 4.3.8: Permissible combinations of bits 4...0

Explanation of bits of parameter *Serial_Mode*:

RTS-Mode... Via this bit the RTS-modem mode for RS-485 interfaces can be selected.

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

Table 4.3.9: Evaluation of *RTS-Mode* bit

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

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

Table 4.3.10: Evaluation of *CTS enable* bits

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

Stop Bit	Number of Stop Bits
0	1 Stop Bit
1	2 Stop Bits (default setting)

Table 4.3.11: Number of Stop Bits

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

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

Table 4.3.12: Parity evaluation

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

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

Table 4.3.13: Setting the polarity

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

Explanation to the Multidrop Mode:

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

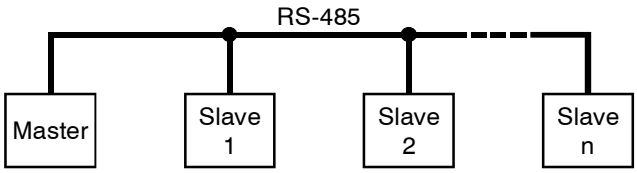


Fig. 4.3.2: Structure of a Multipoint bus

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

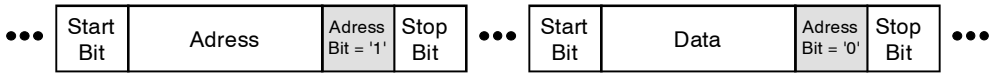


Fig. 4.3.3: Structure of the Tx-data in multidrop mode

4.3.2.6 *TxMinCharStart*

Parameter name	***TxMinCharStart***
Object-index, Sub-index	2800_h, 05_d
Access	R/W
Data Type	Unsigned8
Value Range	$00...FF_h$
Measurement Unit	n.a.
Default Value	0

Description: In this parameter the minimum number of data bytes is specified that has to be stored in the serial Tx-buffer before the transmission of data on the serial interface is started (data CAN -> serial).
This function is required, if strings of more than 8 bytes are to be transmitted together on an RS-485 interface (such as with Master/Slave protocols).

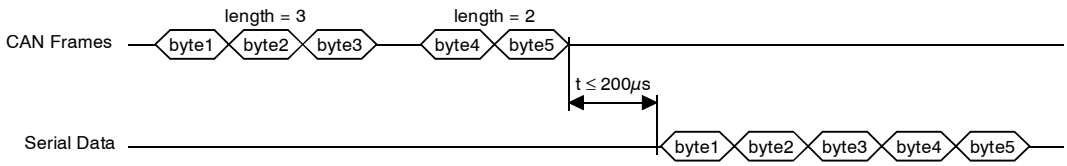


Fig. 4.3.4: Function of parameter *TxMinCharStart*

4.3.2.7 *Protocol*

Parameter name	***Protocol***
Object-index, Sub-index	2800_h, 06_d
Access	R/W
Data Type	Unsigned8
Value Range	$00...83_h$
Measurement Unit	n.a.
Default Value	80_h

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

Bit	Name	Meaning
0	*RTR_HS*	0 - no RTR-handshake (default setting) 1 - remote mode (RTR-handshake)
1	*XON/XOFF_EN*	0 - no XON/XOFF 1 - XON/XOFF-handshake enabled
2 : 6	-	reserved (always set to '0')
7	*CANlink*	0 - module transmits and receives CANlink frames 1 - no CANlink frames

Table 4.3.13: Bits of parameter *Protocol*

Bit 0: *RTR_HS* (RTR-Handshake)
In this handshake mode the CAN-CBM-COM1 module responds with an RTR-frame to the Rx-data received by a CAN master (data CAN -> serial), if capacity for at least one more CAN frame is available in the buffer. The RTR-frame is transmitted on the identifier on which the data has been received (RxPDO1).

The module can send RTR-frames cyclically if the data of the CAN master fail to appear. For this option the parameter *RTR_T_Cycle* (2800_h, 13_d, see page 56) has to be set correspondingly and the buffer of the CAN-CBM-COM1 module must be empty.
If the CAN-CBM-COM1 receives an RTR on TxPDO1, the data is transmitted from the Rx-buffer, even if the number of data in the buffer is smaller than the number defined in *MinChar*.

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

Bit 7: ***CANlink***
This bit enables or disables the CANlink protocol. The default value of the bit is determined by the hardware (jumper X100).
CANlink is a handshake protocol for the CAN. It transmits up to 6 bytes effective information and two bytes handshake information in a CAN frame.
If CANlink is activated, the CAN-CBM-COM1 module utilizes another Tx-identifier (TxPDO2) and another Rx-identifier (RxPDO2) on which acknowledge-messages are transmitted or received.
CANlink will not be explained further in this manual. Please do not hesitate to contact our support to get more information.

4.3.2.8 Handshake_On, Handshake_Off

Parameter name	***Handshake_Off***	***Handshake_On***
Object-index, Sub-index	2800_h, 07_d	2800_h, 08_d
Access	R/W	R/W
Data Type	Unsigned8	Unsigned8
Value Range	$00...FF_h$	$10...FF_h$
Measurement Unit	n.a.	n.a.
Default Value	$F6_h$	$0A_h$

Description: By means of these parameters the handshake function of the serial interface can be set for the data direction serial -> CAN (*this function is not supported in modem mode!*).

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

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

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

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

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

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

4.3.2.9 *TxByRxTimeout*

Parameter name	***TxByRxTimeout***
Object-index, Sub-index	2800_h, 9_d
Access	R/W
Data Type	Unsigned8
Value Range	$00...FF_h$
Measurement Unit	ms
Default Value	0

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

For large data rates it is useful to set parameter *Minchar* = '8' in order to get as much data into a CAN frame as possible. It is possible, however, that less bytes than specified in *MinChar* are stored in the buffer of the CAN-CBM-COM1 with the last transmission. This data would then only be transmitted when transmissions will be resumed again. In order to prevent these data from remaining in the buffer for an unspecified period, parameter *TxByRxTimeout* (*Transfer_at_Rx-Timeout)* has been introduced.

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

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

4.3.2.10 *End_Char_1, End_Char_2*

Parameter name	***End_Char_1***	***End_Char_2***
Object-index, Sub-index	2800_h, 10_d	2800_h, 11_d
Access	R/W	R/W
Data Type	Unsigned8	Unsigned8
Value Range	$00...FF_h$	$00...FF_h$
Measurement Unit	n.a.	n.a.
Default Value	$0D_h$	$0A_h$

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

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

4.3.2.11 *Special_Baudrates*

Parameter name	***Special_Baudrates***
Object-index, Sub-index	2800_h, 12_d
Access	R/W
Data Type	Unsined16
Value Range	$0000...07FF_h$
Measurement Unit	n.a.
Default Value	0034_h

Description: The baud rate of the serial interface can be set in 14 steps by means of parameter *Serial_Baudrates*. Further baud rates can be set with the parameter *Special_Baudrates.*

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

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

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

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

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

4.3.2.12 *RTR_T_Cycle_Time*

Parameter name	***RTR_T_Cycle_Time***
Object-index, Sub-index	2800_h, 13_d
Access	R/W
Data Type	Unsined16
Value Range	0000...$FFFF_h$
Measurement Unit	ms
Default Value	1388_h = 5000 ms

Description: The *RTR_T_Cycle_Time* is only important, if the CAN-CBM-COM1 module runs in RTR-handshake mode. The bit RTR_HS in parameter *Protocol* (see page 49) enables the RTR-handshake mode.

The CAN-CBM-COM1 module responds in the RTR-handshake mode with a RTR-frame to the Rx-data (data CAN -> serial) received by a CAN master, if there is still enough space left in the buffer for at least one further CAN frame. The RTR-frame is transmitted via the same identifier on which the data are received (RxPDO1).

The module can send RTR-frames cyclically, if the data of the CAN master fail to appear. For this option the value of the parameter *RTR_T_Cycle* must be higher than '0' and the buffer of the CAN-CBM-COM1 module must be empty.

4.3.2.13 Tx_Cycle_Time

Parameter name	***TX_Cycle_Time***
Object-index, Sub-index	2800_h, 14_d
Access	R/W
Data Type	Unsined16
Value Range	$0000...FFFF_h$
Measurement Unit	ms
Default Value	0

Description: For recurring transmission orders with the same content, as e.g. used for the initialization of gauges on the serial bus, the CAN-CBM-COM1 module offers a function for cyclic transmission of a programmable character string. The function relieves the CAN bus because the character string must be sent only once to the CAN-CBM-COM1 module.

The time for cyclicyl transmissions of the character string on the serial bus is set via the parameter *Tx_Cycle_Time*. For Tx_Cycle_Time = '0' no transmission will be carried out.
The content of the character string is transmitted via the parameter *Tx_Cycle_String* (Object 2810_h).

4.3.3 *Tx_Cycle_String*

Index [Hex]	Sub-index	Description	Value Range [Hex]	Default Value	Data Type	Access Mode
2810	0	*length_of_string*	0...15	00	unsigned 8	ro
	1	*string*	00...FF	00	unsigned 8	rw
	:	:	:	:	:	:
	15	*string*	00...FF	00	unsigned 8	rw

In this parameter character strings can be programmed which can be transmitted cyclically on the serial bus (see page 56, 57) with the cycle time *Tx_Cycle_Time* (Object 2800_h, 14_d).

Every character string can contain up to 15 bytes.

5. Quick Start

5.1 Configuration for Use in CANopen Network

For a quick start with the most basic configuration the following steps have to be observed:

1. Wire CAN (do not forget terminations!)

2. Set baud rate: Only if a baud rate other than the default configuration is required. The default baud rate is 125 kbaud.
The baud rate can be configured via jumper X100, as described in the hardware manual (chapter: Configuration).

3. Set Node-ID.: Set Node-ID as described in the hardware manual (chapter: Configuration Via Rotary Switches). Permissible values for the Node-ID are between 1 and 127 (01-7F$_h$).

4. Switch on module: Supply power.

5. Start module with:

CAN-Identifier	Len	Data	
0	2 bytes	01_h	00_h

6. Send data CAN -> serial:

CAN-Identifier	Len	Data
200_h + Node-ID	0...8	0...8 byte process data

7. Receive data serial -> CAN:

CAN-Identifier	Len	Data
180_h + Node-ID	0...8	0...8 byte process data

5.2 Table of Most Important Identifiers and Messages for CANopen

CAN identifier [HEX]	Designation	Length	Data [HEX]	Explanations
0	NMT	2	02 xx	CAN module gets into *stopped* status
0	NMT	2	01 xx	Start (CAN module gets into *operational* status)
0	NMT	2	80 xx	CAN module gets into *pre-operational* status
0	NMT	2	81 xx	Reset CAN module
0	NMT	2	82 xx	Reset Communication (here the same function as '81xx')
700_h + Node-ID	NMT error control identifier	1	CANopen status	NMT error control identifier (see DS-301)
580_h + Node-ID	Tx_SDO	8	parameter	Acknowledging communication parameters by CAN-CBM-COM1 module (Tx)
600_h + Node-ID	Rx_SDO	8	parameter	Transmitting communication parameters to CAN-CBM-COM1 module (Rx)
200_h + Node-ID	Rx_PDO_1	0...8 bytes	user data	to CAN-CBM-COM1 (Rx/receive PDO)
180_h + Node-ID	Tx_PDO_1	0...8 bytes	user data	from CAN-CBM-COM1 (Tx/transmit PDO)
300_h + Node-ID	Rx_PDO_2	0...8 bytes	user data	to CAN-CBM-COM1 (Rx/receive PDO, only CANlink protocol)
280_h + Node-ID	Tx_PDO_2	0...2 bytes	user data	from CAN-CBM-COM1 (Tx/transmit PDO, only CANlink protocol)

xx = 00 (all modules) or
xx = Node-ID (module number)
Node-ID = 1...$7F_h$

6. References

[1]	CiA DS-202-2	CAN Application Layer for Industrial Applications	(02.1996)
[2]	CiA DS-301	CANopen Application Layer and Communication Profile	V4.01 (06.2000)
[3]	CiA DS-302	CANopen Framework for Programmable CANopen Devices	V3.0 (06.2000)