Helmholz
COMPATIBLE WITH YOU

CAN 300 PRO – CANopen® Slave

CAN Communication Modules for S7-300 as CANopen® Slave

Manual

Edition 3 / 22.12.2011

Systeme Helmholz **GmbH** | Hannberger Weg 2 | D-91091 Großenseebach
Phone +49 9135 7380-0 | Fax +49 9135 7380-110 | info@helmholz.de | www.helmholz.com

Systeme Helmholz GmbH

Hannberger Weg 2, 91091 Grossenseebach, Germany

Note:

We have checked the content of this manual for conformity with the hardware and software described. Nevertheless, because deviations cannot be ruled out, we cannot accept any liability for complete conformity. The information in this manual is regularly updated. When using purchased products, please heed the latest version of the manual, which can be viewed in the Internet at www.helmholz.de, from where it can also be downloaded.

Our customers are important to us. We are always glad to receive suggestions for improvement and ideas.

Revision history of this document:

Edition	Date	Revision
1	16.06.2009	1st version
2	18.08.2010	Handling blocks modified; BUSY included
3	22.12.2011	Error codes completed and further small corrections

Contents

1 Overview

1.1 General

The CAN 300 PRO module is intended for use in Siemens S7-300 programmable controllers. The CAN 300 PRO module is used to connect the programmable controller to the CAN bus.

Besides applications as the CANopen® Master or for Layer 2 communication, the CAN 300 PRO can also be used to integrate the S7-300 PLC as a CANopen® Slave in a CANopen® network.

The CAN 300 PRO module can operate as a CANopen® Slave with the necessary data handling blocks and after installation of the CANopen® Slave firmware and a slave definition file.

This manual describes the module in its function as a CANopen® Slave. It is intended to be used in conjunction with the manual for the CAN 300 PRO module.

1.2 Connections

The CAN 300 PRO module features a 9-way SubD connector behind the hinged front cover for the CAN bus and a USB connector for configuration and diagnostics.

Pin assignment:

Pin	SUBD connector CAN
1	-
2	CAN Low
3	CAN GND
4	-
5	-
6	-
7	CAN High
8	-
9	-

A 24V power supply is not applied to the CAN bus connector.

1.3 LED displays

The LEDs on the front of the module inform you about its operating state.

LED "SF" (orange):

System error: Shows a slave definition file error.

LED "BF" (red):

This LED indicates a CAN error. A CAN error has occurred if the error counters are not zero and the CAN status is not "OK," or a CAN FIFO overflow has occurred. You can obtain further information in debug mode of the CANParam software (see also Section 4.3).

LED "RX" (green):

CAN bus reception active: Indicates correct reception of a CAN frame.

LED "TX" (orange):

CAN bus transmission active: Indicates correct transmission of a CAN frame.

LED "CPU" (orange):

Data transmission to the PLC active: Indicates transmission of a frame or command on the backplane bus (between the S7-CPU and the module).

LED "ON" (green):

A continuous light indicates that the module is operating as a CANopen® Slave in "Operational" mode. Slow flashing indicates that the module is in the "Preoperational" or "Stopped" state.

1.4 DIP switch

The 10-way DIP switch on the housing front is provided to set the CAN baudrate and define the node address.

Address	2^6	+ 64
	2^5	+ 32
	2^4	+ 16
	2^3	+ 8
	2^2	+ 4
	2^1	+ 2
	2^0	+ 1
Baud	2^2	+ 4
	2^1	+ 2
	2^0	+ 1

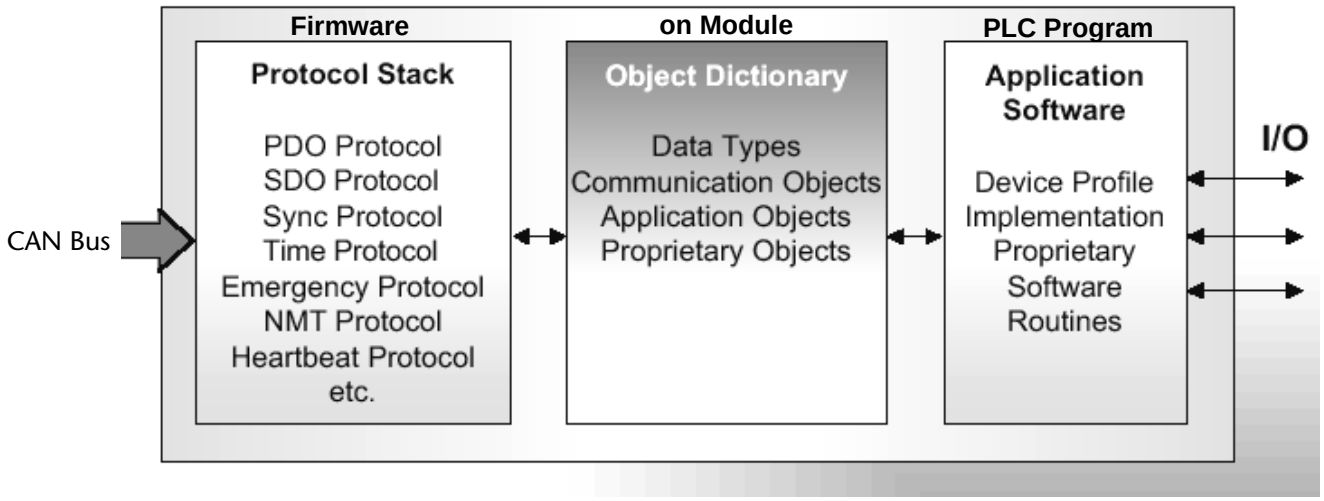
Baudrates:

0	1	2	3	4	5	6	7
10K	50K	100K	125K	250K	500K	800K	1M

2 CANopen® Slave Function

2.1 Objects

The interface for all information of a CANopen® Slave is the "Object Dictionary." The objects of the Object Dictionary contain all the information about the status of the slave and all process values to be exchanged with the CAN bus master or with other slaves (IO values, actual values, setpoint values, error states, etc.).



The objects (1000...1FFF) are implemented for the basic setting of the CANopen® Slave. These objects distinguish the CANopen® Slave from the CANopen® Master and contain all status information.

Objects 6000h…9FFFh are reserved for process information when standard CANopen® profiles are used.

Objects 2000h…3FFFh can be defined by the manufacturer.

The object table is located on the module and is exchanged with the PLC by the data handling blocks. The objects are defined by the slave definition file which is imported into the module

2.2 Node addresses

Every CANopen® Slave must have node address. The node address can be between 1 and 127 and can be set in the slave definition file or using the DIP switch.

2.3 Network management

The CANopen® Slave data handling supports BootUp messages, heartbeat, and node guarding.

The objects 100Ch (“GuardTime”) and 100Dh (“LifetimeFactor”) are used for monitoring in nodeguarding. If the set time expires, the CANopen® Slave automatically assumes the “Preoperational" state.

Heartbeat supports transmission of the slave heartbeat (time in object 1017h “Producer Heartbeat Time“), as well as maximum 2 consumer heartbeats (object 1016h subindex 1 und 2) for monitoring received heartbeats.

For a more detailed description of the CANopen® frames please see Section 6.

2.4 Emergencies

Transmission of emergency messages and management of the current error state (objects 1001h and 1003h) are supported.

2.5 Service data (SDO)

Reading and writing of SDOs (1-4 bytes) is supported. The SDOs can be provided with read or read/write rights.

Reading and writing SDOs of more than 4 bytes (segmented transfer) is also supported.

2.6 Process data (PDO)

CANopen® Slave data handling supports up to 4 send and transmit PDOs (RPDO1-4, TPDO1-4). The COB IDs of the PDOs are permanently written in the data handling blocks and cannot be changed.

TPDO1: 180h+node ID	RPDO1: 200h+node ID
TPDO2: 280h+node ID	RPDO2: 300h+node ID
TPDO3: 380h+node ID	RPDO3: 400h+node ID
TPDO3: 480h+node ID	RPDO3: 500h+node ID

The PDOs can be mapped via the usual objects 1600h ff. and 1A00h ff.

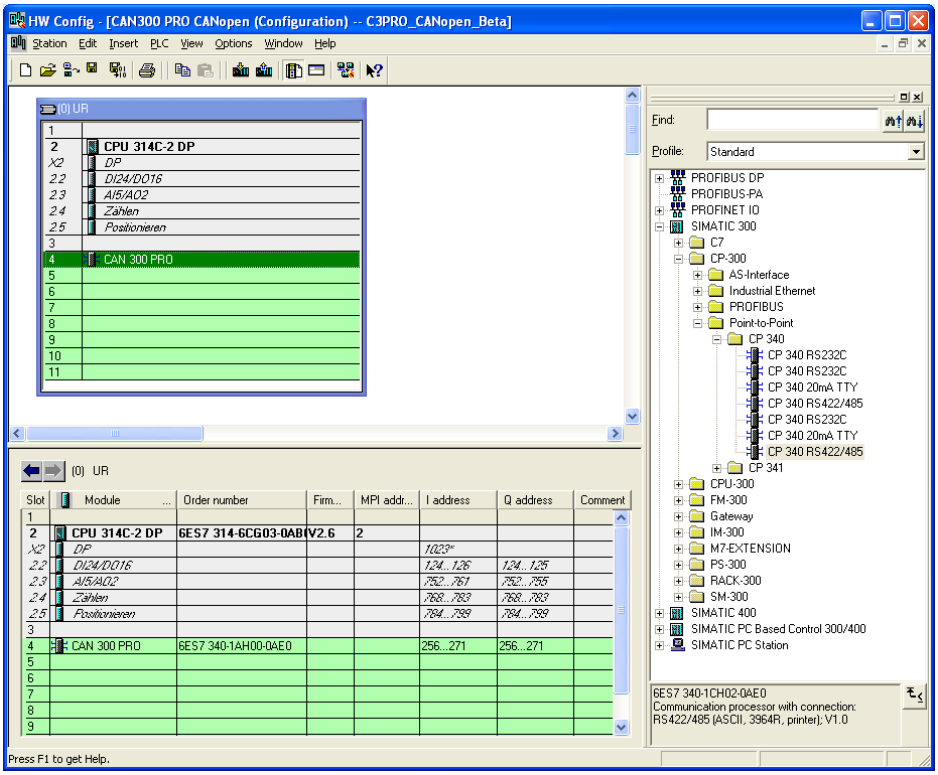
The following transmission types are available for TPDOs (objects 1800h ff.):

Transmit after x SYNC frames:	1-240
Transmit only after request (RTR):	252, 253
Transmit after change	254, 255

RPDO values are applied as soon as they are received.

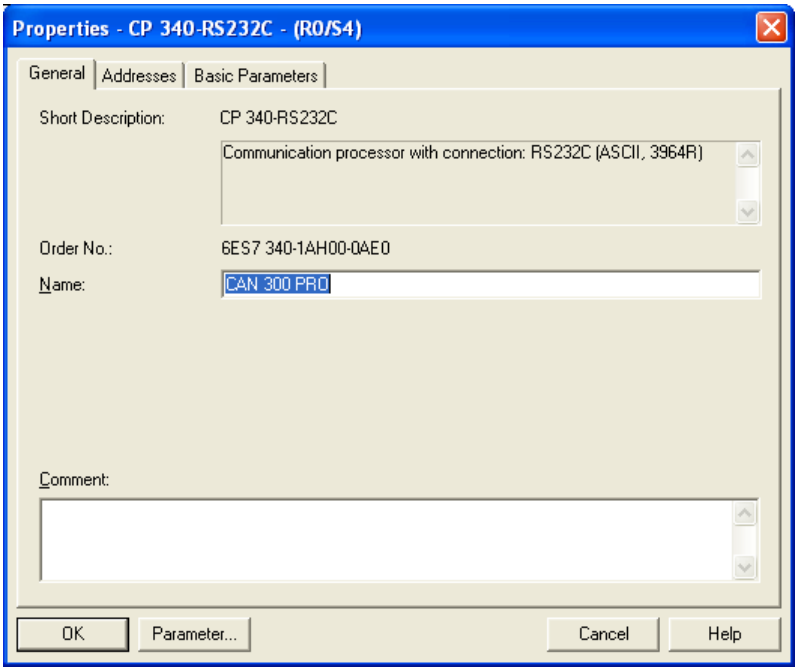
3 Configuration in the PLC

The CAN 300 PRO module is configured as the CP 340 communication module in the programming software of the PLC.



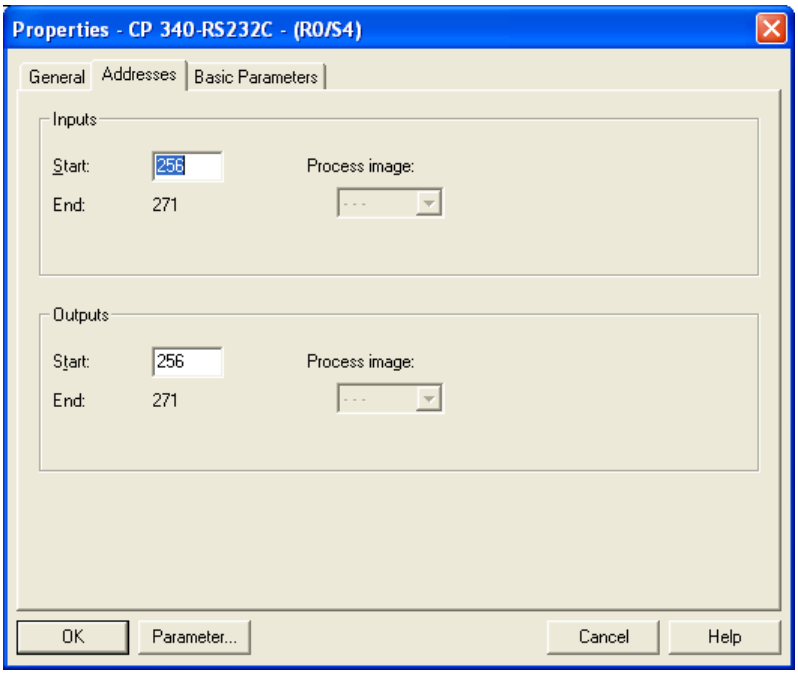
The module can be used wherever a CP module is allowed, i.e. also in the expansion unit after an interface module.

When the CAN 300 PRO module is used in an ET200M system, noticeably poorer performance must be expected.



*The I/O addresses should **not** be in the cyclic process image!*

In parameterization of the module, only the range of I/O addresses is relevant. All other settings have no effect on the module.



Only the input image is used in the data handling blocks; the output image has no relevance to the function.

Accesses to the input image can only be performed with the I/O direct access commands: L PEB, L PEW, L PED.

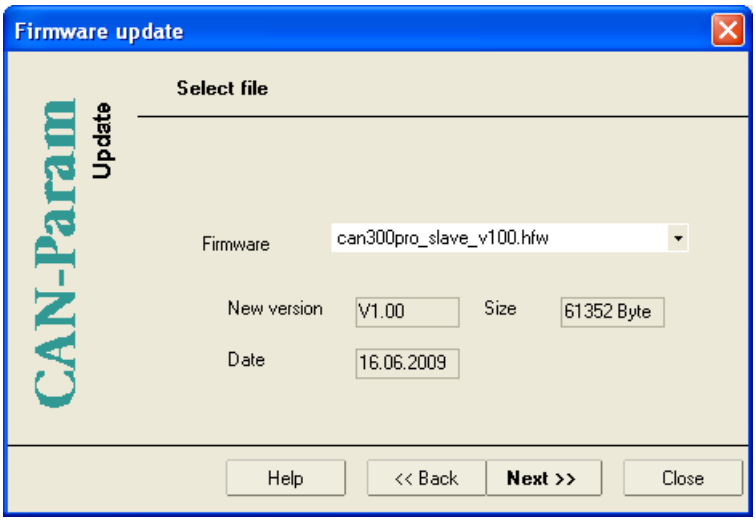
In the case of the CPU 318, the I/O addresses must be outside the cyclic process image.

4 Configuring the CAN 300 PRO Module

4.1 Transferring the CANopen® Slave operating system

The CAN modules are configured on the PC with the “CANParam V4.1x” and newer software. The CAN 300 PRO module is supplied with the operating system for a CANopen® Master. The latest CANopen® Slave firmware must be installed on the module before initial start-up

The function “Firmware Update” in menu “Online” provided in the CANParam software can be used for this. When updating starts, the latest slave firmware must be selected.



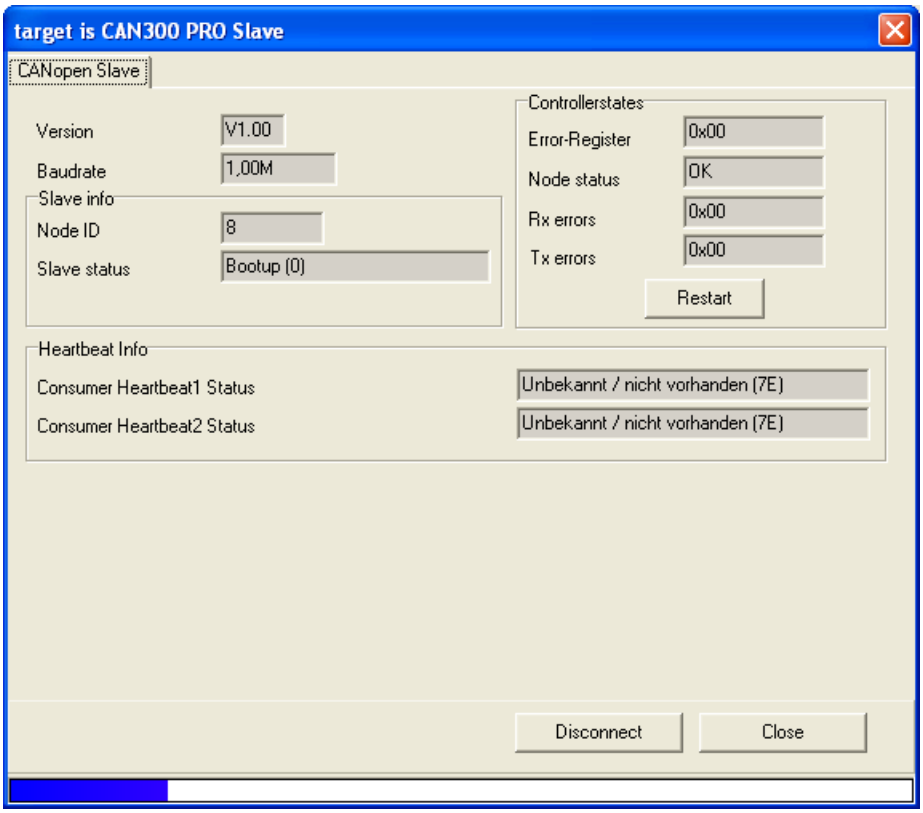
4.2 Transferring the slave definition file

In menu “Online,” the slave definition file for the module can be sent with the “Send Project from File” function.

The slave definition file is a text file with the extension “PAR”. This file describes the structure and basic state of the slave with all its SDOs and PDOs.

A slave definition file is supplied for the DS 401 CANopen® profile. Any number of definition files can be created for other CANopen® applications; please consult Systeme Helmholz Support for help.

4.3 Diagnostics



The CANopen® Debug dialog box provides the following information:

Version	Version number of the operating system
Baudrate	Active CAN baud rate
Controller status	Content of the CAN status register:
Error register	Content of the CAN error register EFLG (Sec. 5.1.2)
Node status	Content of the CAN status register (see above):"OK," "Warning," "Passive," "Bus Off"
Rx error counter	Error counter CAN reception
Tx error counter	Error counter CAN transmission

Slave information:

Node ID	Currently valid node ID of slave
Slave status	00 = Bootup 04 = Stop 05 = Operational 7F = Preoperational

Heartbeat info:

The status of the monitored slave is displayed here.

! *Node status should always be "OK" to ensure fault-free CAN data transmission.*

5 Programming in the PLC

The CANopen® Slave is programmed in the PLC using data handling blocks and information from the process image of the module.

5.1 Process image in the PLC

The CAN 300 PRO module occupies 16 bytes in the input and output process image. The content of the output process image is not used.

The content of the input process image can be used for information purposes by the user in the application.

Byte	Meaning
0	Module status generally, CAN group error display
1	CAN controller status (register of the CAN controller)
2	FIFO status bits (send & receive)
3	CAN controller: TX error counter
4	CAN controller: RX error counter
5	CANopen® slave status (0, 4, 5, 0x7f)
6	Status of Heartbeat Consumer 1
7	Status of Heartbeat Consumer 2
8	Active node ID of slave
9…15	*used internally*

Accesses to the input image can only be performed with the I/O direct access commands: L PEB, L PEW, L PED.

5.1.1 Byte 0: Module status

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
CAN controller group error	0	Module is CAN 300 PRO Slave	0	0	0	0	Module parameterized and running

Bit 0: The CAN 300 PRO module has processed the configuration and is ready for operation.

Bit 5: This bit is always 1 in order to detect the CAN 300 PRO Slave.

Bit 7: Group error bit for errors on the CAN controller, more precise information about the cause of error can be found in byte 1.

5.1.2 Byte 1: Error status (EFLG) of the CAN controller

RX1OVR	RX0OVR	TXBO	TXEP	RXEP	TXWAR	RXWAR	EWARN
bit 7							bit 0

bit 7 **RX1OVR**: Receive Buffer 1 Overflow Flag
- Set when a valid message is received for RXB1 and CANINTF.RX1IF = 1
- Must be reset by MCU

bit 6 **RX0OVR**: Receive Buffer 0 Overflow Flag
- Set when a valid message is received for RXB0 and CANINTF.RX0IF = 1
- Must be reset by MCU

bit 5 **TXBO**: Bus-Off Error Flag
- Bit set when TEC reaches 255
- Reset after a successful bus recovery sequence

bit 4 **TXEP**: Transmit Error-Passive Flag
- Set when TEC is equal to or greater than 128
- Reset when TEC is less than 128

bit 3 **RXEP**: Receive Error-Passive Flag
- Set when REC is equal to or greater than 128
- Reset when REC is less than 128

bit 2 **TXWAR**: Transmit Error Warning Flag
- Set when TEC is equal to or greater than 96
- Reset when TEC is less than 96

bit 1 **RXWAR**: Receive Error Warning Flag
- Set when REC is equal to or greater than 96
- Reset when REC is less than 96

bit 0 **EWARN**: Error Warning Flag
- Set when TEC or REC is equal to or greater than 96 (TXWAR or RXWAR = 1)
- Reset when both REC and TEC are less than 96

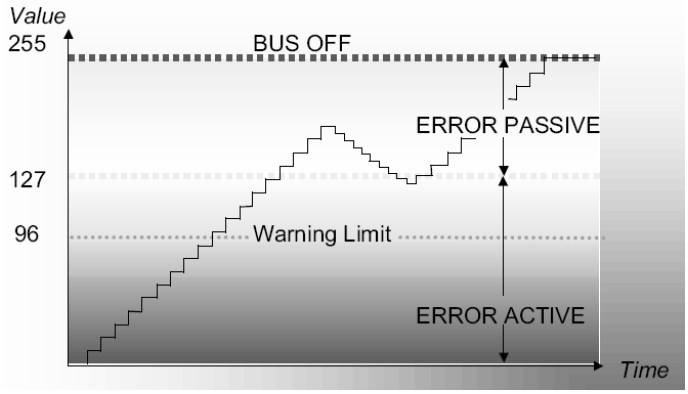
5.1.3 Byte 2: FIFO status bits

Bit 7	Bit 6	Bit 5	Bit 4
Send-FIFO (high) half full	Send-FIFO (high or low) overflow	Send-FIFO (low) half full	Send-FIFOs (high & low) completely empty

Bit 3	Bit 2	Bit 1	Bit 0
Receive-FIFO (high) half full	Receive-FIFO (high or low) overflow	Receive-FIFO (low) half full	Receive-FIFOs (high & low) completely empty

5.1.4 Byte 3/4: CAN controller Tx/Rx error counter

The error counter is incremented on every CAN frame transmitted or received with an error. If a CAN frame has been correctly transmitted, the error counter is decremented again. If the counter is greater than 96, the CAN controller goes into “warning” mode (see 5.1.2). If the error counter exceeds 127, the CAN controller goes into “Error Passive.”



5.1.5 Byte 5: CANopen® Slave status

This byte indicates the current state of the CANopen® Slave state machine.

0 = Bootup
4 = Stop
5 = Operational
7F = Preoperational

5.1.6 Byte 6+7: Status of Heartbeat Consumer 1+2

This byte indicates the current status of devices monitored by the Consumer Heartbeat:

0 = Bootup
4 = Stop
5 = Operational
7E = unknown / not available
7F = Preoperational

5.1.7 Byte 8: Active node ID

This byte indicates the active node ID of the slave.

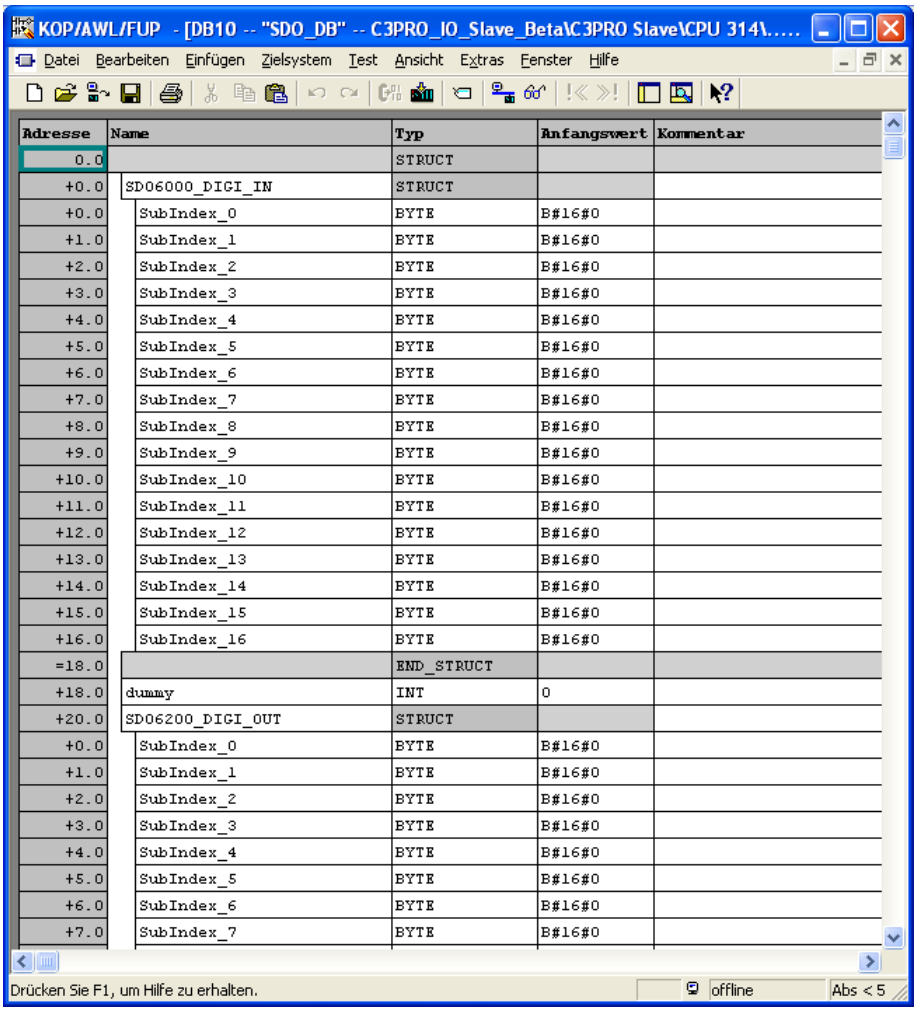
5.2 Data handling blocks

The CANopen® Slave is programmed in the PLC with the following data handling blocks:

FB 90	**Get SDO Block**	Fetch SDO data into the PLC
FB 91	**Send SDO Block**	Send SDO data to the module
FB 92	**SDO Write**	Write an SDO
FB 93	**SDO Read**	Read out an SDO
FB 94	**Send Emergency**	Send an emergency message

5.2.1 SDO data block

The current data of all defined SDOs are kept in the CAN 300 PRO module. To be able to work with these data in the PLC, they are copied from the module into data block SDO DB at the beginning of the cycle with function block FB 90.



Adresse	Name	Typ	Anfangswert	Kommentar
0.0		STRUCT		
+0.0	SDO6000_DIGI_IN	STRUCT		
+0.0	SubIndex_0	BYTE	B#16#0	
+1.0	SubIndex_1	BYTE	B#16#0	
+2.0	SubIndex_2	BYTE	B#16#0	
+3.0	SubIndex_3	BYTE	B#16#0	
+4.0	SubIndex_4	BYTE	B#16#0	
+5.0	SubIndex_5	BYTE	B#16#0	
+6.0	SubIndex_6	BYTE	B#16#0	
+7.0	SubIndex_7	BYTE	B#16#0	
+8.0	SubIndex_8	BYTE	B#16#0	
+9.0	SubIndex_9	BYTE	B#16#0	
+10.0	SubIndex_10	BYTE	B#16#0	
+11.0	SubIndex_11	BYTE	B#16#0	
+12.0	SubIndex_12	BYTE	B#16#0	
+13.0	SubIndex_13	BYTE	B#16#0	
+14.0	SubIndex_14	BYTE	B#16#0	
+15.0	SubIndex_15	BYTE	B#16#0	
+16.0	SubIndex_16	BYTE	B#16#0	
=18.0		END_STRUCT		
+18.0	dummy	INT	0	
+20.0	SDO6200_DIGI_OUT	STRUCT		
+0.0	SubIndex_0	BYTE	B#16#0	
+1.0	SubIndex_1	BYTE	B#16#0	
+2.0	SubIndex_2	BYTE	B#16#0	
+3.0	SubIndex_3	BYTE	B#16#0	
+4.0	SubIndex_4	BYTE	B#16#0	
+5.0	SubIndex_5	BYTE	B#16#0	
+6.0	SubIndex_6	BYTE	B#16#0	
+7.0	SubIndex_7	BYTE	B#16#0	

A fixed assignment between the SDO and the memory location in the SDO data block is defined in the slave definition file.

The data of the SDOs can now be processed and modified in the PLC cycle. At the end of the PLC cycle the entire contents of the SDO DB are transferred back to the module with FB 91.

Write access to SDOs via the CAN bus have priority!

If in the meantime the CAN bus has written to the SDO, this has priority over the changes in the PLC.

To keep data block to a manageable size and thus limit the transmission time between the PLC and the module, not all defined SDOs have to be copied.

Only those SDOs that are actually needed for cyclic operation are “mapped” to the SDO data block. All other SDOs can be read and written to via function blocks FB 92 and FB 93 if this is done just once or rarely.

5.2.2 FB 90 Get SDO Block

The function block Get SDO Block (FB90) transfers the latest status of the SDO table to the PLC and stores those data in any data block.

The FB should be called at the beginning of the PLC cycle.

Parameter	Direction	Type	Example
Base	IN	INT	256
SDO_Block	IN	ANY	P#DB10.DBX0.0 BYTE 100
STAT	OUT	WORD	MW 10
Err	OUT	BOOL	M 94.7
RetVal	OUT	INT	MW12
Busy	IN_OUT	BOOL	
Lock	IN_OUT	BOOL	

Base	Address of the CAN 300 PRO module
SDO_Block	ANY pointer to the SDO data block
STAT	Status of the module, see Section 5.2.7
Err	Error bit is 0 on successful implementation
RetVal	Error number, see Section 5.4
Busy/Lock	reserved for ET200M applications

5.2.3 FB 91 Send SDO Block

The function block Send SDO Block (FB91) sends the latest status of the SDO table from the SDO data block to the module.

The FB should be called at the end of the PLC cycle.

Parameter	Direction	Type	Example
Base	IN	INT	256
SDO_Block	IN	ANY	P#DB10.DBX0.0 BYTE 100
STAT	OUT	WORD	MW 10
Err	OUT	BOOL	M 95.7
RetVal	OUT	INT	MW14
Busy	IN_OUT	BOOL	
Lock	IN_OUT	BOOL	

Base	Address of the CAN 300 PRO module
SDO_Block	ANY pointer to the SDO data block
STAT	Status of the module, see Section 5.2.7
Err	Error bit is 0 on successful implementation
RetVal	Error number, see Section 5.4
Busy/Lock	reserved for ET200M applications

5.2.4 FB 92 SDO Write

The function block SDO Write (FB 92) writes one SDO value to the module.

Parameter	Direction	Type	Example
Base	IN	INT	256
Index	IN	WORD	W#16#2001
Subindex	IN	BYTE	B#16#0
SDO_Data	IN	DWORD	MD 30
SDO_Len	IN	BYTE	MB 34
RetVal	OUT	INT	MW 35
Activate	IN_OUT	Bool	M 39.0
Busy	IN_OUT	Bool	M 39.1
Err	IN_OUT	Bool	M 39.2
Done	IN_OUT	Bool	M 39.3

Base	Address of the CAN 300 PRO module
Index	SDO index
Subindex	SDO subindex
SDO_Data	Data for the SDO (right justified)
SDO_Len	Size of the SDO (1, 2, 4 bytes)
RetVal	Error number, see Section 5.4
Activate	activate new SDO write job
Busy	SDO job is running
Err	SDO job finished with error
Done	SDO job finished without error

The value of the SDO must be right-justified in the double word irrespective of the data length.

Application example:

```
CALL  FB    92 , DB92
 Base    :=256
 Index   :=W#16#2001
 Subindex:=B#16#0
 SDO_Data:=MD30
 SDO_Len :=B#16#2
 RetVal  :=MW35
 Activate:=M39.0
 Busy    :=M39.1
 Err     :=M39.2
 Done    :=M39.3
…
```

5.2.5 FB 93 SDO Read

The function block SDO Read (FB 93) fetches one SDO value from the module.

Parameter	Direction	Type	Example
Base	IN	INT	256
Index	IN	WORD	W#16#2000
Subindex	IN	BYTE	B#16#0
RetVal	OUT	INT	MW 25
SDO_Data	IN	DWORD	MD 20
SDO_Len	IN	BYTE	MB 24

Base	Address of the CAN 300 PRO module
Index	SDO index
Subindex	SDO subindex
RetVal	Error number, see Section 5.4
SDO_Data	Data for the SDO (right justified)
SDO_Len	Size of the SDO (1, 2, 4 bytes)
Activate	Activate new SDO read job
Busy	SDO job is running
Err	SDO job finished with error
Done	SDO job finished without error

The value is always “right-justified” in the double word and can be further processed immediately; for 1-byte or 2-byte values the double word is filled up with leading zeros.

Application example:

```
      CALL  FB    93 , DB93
       Base    :=256
       Index   :=W#16#2000
       Subindex:=B#16#0
       RetVal  :=MW25
       SDO_Data:=MD20
       SDO_Len :=MB24
       Activate:=M29.0
       Busy    :=M29.1
       Err     :=M29.2
       Done    :=M29.3

      AN    M     29.3
      JC    next

      // use read value
      R     M     29.3
      L     MD    20
      …

next: …
```

5.2.6 FB 94 Send Emergency

The function block Emergency (FB 94) sends an Emergency frame and enters the message in the relevant SDOs.

Parameter	Direction	Type	Example
Base	IN	INT	256
Emcy_Code	IN	WORD	W#16#1000
Emcy_State	IN	BYTE	B#16#03
Emcy_ManuErr	INT	DWORD	DW#16#00000000
STAT	OUT	WORD	MW 68
Err	OUT	BOOL	M 63.7
RetVal	OUT	INT	MW 70

Base	Address of the CAN 300 PRO module
Emcy_Code	Emergency code
Emcy_State	Emergency status for object 1001h
Emcy_ManuErr	Emergency manufacturer error
STAT	Status of the module, see Section 5.2.7
Err	Error bit is 0 on successful implementation
RetVal	Error number, see Section 5.4

5.2.7 Parameter STAT

The STAT parameter has the same meaning in all data handling blocks and indicates the status of the module:

Bit 15	Bit 14	Bit 13	Bit 12
CAN controller group error	0	Module is CAN 300 PRO Slave (always 1)	0
Bit 11	**Bit 10**	**Bit 9**	**Bit 8**
0	0	0	Module running, read-in of the parameters completed

Bit 7	Bit 6	Bit 5	Bit 4
Send-FIFO (high) half full	Send-FIFO (high or low) overflow	Send-FIFO (low) half full	Receive-FIFOs (high & low) completely empty
Bit 3	**Bit 2**	**Bit 1**	**Bit 0**
Receive-FIFO (high) half full	Receive-FIFO (high or low) overflow	Receive-FIFO (low) half full	Receive-FIFOs (high & low) completely empty

The STAT parameter corresponds to the I/O input bytes 0 and 2.

5.3 Abort codes

Code	Meaning
0503 0000h	"Toggle bit" has not been alternated
0504 0000h	SDO protocol "time out "
0504 0001h	Client/server command designation not valid or unknown
0504 0005h	Outside the memory
0601 0000h	Access to this object is not supported
0601 0001h	Attempted read access to an object that can only be written
0601 0002h	Attempted write access to an object that can only be read
0602 0000h	Object does not exist in the object directory
0604 0041h	Object cannot be "mapped" to a PDO
0604 0042h	Size and number of "mapped" objects exceeds the possible PDO length
0604 0043h	General parameters –incompatibility
0604 0047h	General incompatibility in the device
0606 0000h	Access violation due to a hardware error
0607 0010h	Data type does not match, length of the service parameter does not fit
0607 0012h	Data type does not match, length of the service parameter too large
0607 0013h	Data type does not match, length of the service parameter too small
0609 0011h	Subindex does not exist
0609 0030h	Out of value range of the parameter (only for write accesses)
0609 0031h	Value of the parameter too large
0609 0032h	Value of the parameter too small
0609 0036h	Maximum value is smaller than the minimum value

5.4 Error codes of the FBs

The return parameter RetVal of the function blocks can contain both function-specific errors or error numbers of the Siemens system function blocks SFC 52, SFC 53, SFC 14 and SFC 20.

Error codes of the CAN handling:

80E1h: SDO-FBs: Data len to Null is not allowed

80E2h: SDO-FBs: Data len greater than 4 is not allowed

80F1h: Module not ready

80F2h: Data set assigned

80F7h: CANopen® Slave still in Bootup

8xF8h: SDO data block pointer: Not enough memory

80FAh: Abort code for SDO job received.

6 CANopen® Protocol

6.1 General

CIA® = CAN in Automation e.V., www.can-cia.org

The CANopen® protocol is a layer 7 protocol (application layer) based on the CAN bus (ISO 11898). Layer 1 and 2 (physical layer and data link layer) of the CAN bus are not affected.

The CANopen® communication profiles for the various applications are managed by the CIA.

The services elements provided by the application layer permit implementation of an application distributed over the network. These service elements are described in “CAN Application Layer (CAL) for Industrial Applications.”

The 11 bit identifier and the 8 data bytes of a CAN layer 2 frame have a fixed meaning.

Every device in a CANopen® network has a fixed node ID (module number, 1-127).

6.2 Objects

Data exchange with a CANopen® Slave is performed either using permanently defined service data objects (SDO) or using freely configurable process data objects (PDO).

Each CANopen® Slave has a fixed list of SDOs that are addressed by an object number (16 bits) and an index (8 bits).

Example: Object 1000h/ Index 0 = Device Type, 32Bit Unsigned

SDOs with a width of 8/16/32 bits can be read and written with a CANopen® frame. SDOs that are longer are transmitted in more than one frame. For very large volumes of data, SDO block transmission is possible.

SDOs can be processed as soon as a CANopen® Slave is ready for operation. For the SDOs, only the COB ID functions “SDO request” or “SDO response” are available. The object number, access mode, and type are stored in the first 4 bytes of the CAN frame.

The last 4 bytes of the CAN frame then contain the value for the SDO.

PDOs contain the “working values” of a CANopen® Slave for cyclic process operation. Each CANopen® Slave can manage several PDOs (normally up to 4 for transmitting and up to 4 for receiving).

Each of the existing PDOs has its own COB-ID. It is possible to map any information of the CANopen® Slave to the 8 data bytes of the frame for reading and writing. These can be both existing SDOs and updated values of the slaves (e.g. analog value or an input).

The PDOs are automatically mapped from most CANopen® Slaves on startup. The assignment can be changed using certain SDOs.

6.3 Functions

The CANopen® functions are subdivided into the three basic groups:

- Reading and writing SDO
- Reading and writing PDO
- Network management

The function code is stored in the upper 4 bits of the identifier. Together with the node ID this makes up the COB identifier.

COB identifier (COB-ID):

10	9	8	7	6	5	4	3	2	1	0
Function				Node ID						

It is possible to change some COB-IDs to other values using special service data objects (SDOs).
This is NOT supported by the CANopen® slave!

Broadcast functions:

Function	Function code (binary)	Resulting COB-ID
NMT	0000	0h
SYNC	0001	80h
TIME STAMP	0010	100h

Node functions:

Function	Function code (binary)	Resulting COB-ID
EMERGENCY	0001	81h –FFh
PDO1 (tx)	0011	181h – 1FFh
PDO1 (rx)	0100	201h – 27Fh
PDO2 (tx)	0101	281h – 2FFh
PDO2 (rx)	0110	301h – 37Fh
PDO3 (tx)	0111	381h – 3FFh
PDO3 (rx)	1000	401h – 47Fh
PDO4 (tx)	1001	481h – 4FFh
PDO4 (rx)	1010	501h – 57Fh
SDO (tx)	1011	581h – 5FFh
SDO (rx)	1100	601h – 67Fh
NMT Error Control	1110	701h – 77Fh

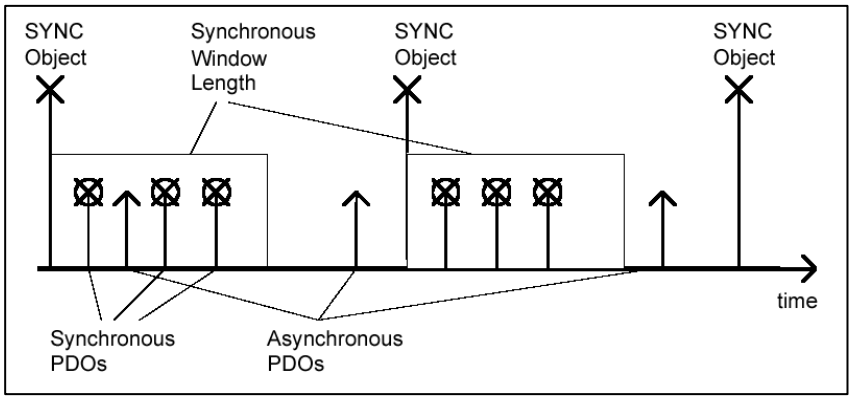
"Tx" = is transmitted by the slave
"Rx" = is received from the slave

6.4 Network management

SYNC:

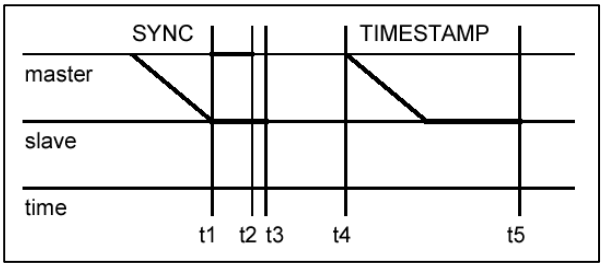
The SYNC frame is a cyclic “broadcast” frame and sets the basic bus clock. To ensure isosynchronism, the SYNC frame has a high priority.

COB-ID: 80h



Time Stamp:

The time stamp frame is a cyclic “broadcast” frame and provides the system time. The time stamp frame is usually transmitted directly after a SYNC frame and then provides the system time of the SYNC frame.



To ensure a precise transmission, the time stamp frame has a high priority.

COB-ID: 100h

Nodeguarding:

With the nodeguarding function, the master monitors the CANopen® Slave modules by transmitting frames cyclically to each slave. Each CANopen® Slave must respond to the nodeguarding message frame with a status frame.

The control can detect failure of a CANopen® Slave using nodeguarding.

COB-ID: 700h + node ID +RTR

Response: COB-ID: 700h + node ID + 1 byte data: Status

Lifeguarding:

In lifeguarding, each CANopen® Slave continuously monitors whether the master is performing nodeguarding once it has been started within certain time limits.

If the nodeguarding frame of the master fails, the distributed I/O module can detect that using lifeguarding and, for example, put all outputs into the safe state.

Heartbeat:

Heartbeat monitoring is equivalent to nodeguarding although no request frames are generated by CANopen® master. The heartbeat frame is transmitted automatically by the node and can be evaluated in the master.

Emergency message:

If a fault occurs on a CANopen® Slave, it sends an emergency message to the bus.

COB-ID: 80h + node ID]

All stations can perform an emergency stop on receiving an emergency frame, for example.

BootUp message:

CANopen® Slaves generate a BootUp message after switch-on that the master can recognize to initialize this new station.

COB-ID: 700h + node ID + 1 byte data: 00h

7 Appendix

7.1 Object directory

An overview of the implemented objects of the CANopen® Slave are given below.

The system objects are always available. All other objects are created by the slave definition file when it is imported into the module.

7.1.1 System objects (1000h - 1FFFh)

Object	Description	Value range	Info
1000	Device type	UNSIGNED32	Profile 401
1001	Error register	UNSIGNED8	Current error status
1003	Predefined error field	ARRAY	
/0		UNSIGNED8	Number of entries
/1		UNSIGNED32	Current error
1004	PDOs supported	UNSIGNED32	4 send and 4 receive PDOs
1005	COB-ID SYNC message	UNSIGNED32	ID for SYNC messages
1008	Manufacturer device name	STRING	
1009	Hardware version	STRING	
100A	Software version	STRING	
100B	Active node address	STRING	
100C	Guard time	UNSIGNED16	Nodeguarding monitoring time
100D	Lifetime factor	UNSIGNED8	Lifetime factor for nodeguarding
1012	COB ID timestamp	UNSIGNED32	COB ID for timestamp frames
1014	COB ID Emergency	UNSIGNED32	COB ID for emergency
1016	Consumer heartbeat	UNSIGNED8	
/0	Consumer heartbeat 1	UNSIGNED32	
/1	Consumer heartbeat 2	UNSIGNED32	
1017	Heartbeat time	UNSIGNED16	Time of send heartbeat (500ms)
1018	Identity	RECORD	
/0		UNSIGNED8	Number of entries
/1		UNSIGNED32	Vendor ID
/2		UNSIGNED32	Product ID
/3		UNSIGNED32	Release
/4		UNSIGNED32	Serial number
1029	Error behavior	ARRAY	
/0		UNSIGNED8	Number of entries
/1		UNSIGNED8	
/2		UNSIGNED8	
1400	RPDO1 Comm Param	ARRAY	
/0		UNSIGNED8	Number of entries
/1		UNSIGNED32	COB ID
/2		UNSIGNED8	Transmission type (FFh=event-triggered)
1401	RPDO2 Comm Param	ARRAY	
…	…		
1402	RPDO3 Comm Param	ARRAY	
…	…		
1403	RPDO4 Comm Param	ARRAY	
…	…		
1600	RPDO1 Mapping	ARRAY	
/0		UNSIGNED8	Number of entries
/1		UNSIGNED32	1. Mapping
/2		UNSIGNED32	2. Mapping
/3		UNSIGNED32	3. Mapping
/4		UNSIGNED32	4. Mapping
/5		UNSIGNED32	5. Mapping

/6		UNSIGNED32	6. Mapping
/7		UNSIGNED32	7. Mapping
/8		UNSIGNED32	8. Mapping
1601	RPDO2 Mapping	ARRAY	
	…		
1602	RPDO3 Mapping	ARRAY	
	…		
1603	RPDO4 Mapping	ARRAY	
	…		
1800	TPDO1 Comm Param	ARRAY	
/0		UNSIGNED8	Number of entries
/1		UNSIGNED32	COB ID
/2		UNSIGNED8	Transmission type (FFh=event-triggered)
1801	TPDO2 Comm Param	ARRAY	
…	…		
1802	TPDO3 Comm Param	ARRAY	
…	…		
1803	TPDO4 Comm Param	ARRAY	
…	…		
1A00	TPDO1 Mapping	ARRAY	
/0		UNSIGNED8	Number of entries
/1		UNSIGNED32	1. Mapping
/2		UNSIGNED32	2. Mapping
/3		UNSIGNED32	3. Mapping
/4		UNSIGNED32	4. Mapping
/5		UNSIGNED32	5. Mapping
/6		UNSIGNED32	6. Mapping
/7		UNSIGNED32	7. Mapping
/8		UNSIGNED32	8. Mapping
1A01	TPDO2 Mapping	ARRAY	
	…		
1A02	TPDO3 Mapping	ARRAY	
	…		
1A03	TPDO4 Mapping	ARRAY	
	…		

7.1.2 Application objects (6000h – 6FFFh)

The application objects are adapted to customer requirements.

Standard objects for an IO slave acc. to profile DS401:

SDO 6000h, subindex 1-16, unsigned8: 16-byte digital inputs

SDO 6200h, subindex 1-16, unsigned8: 16-byte digital outputs

SDO 6401h, subindex 1-8, unsigned16: 8-word analog inputs

SDO 6411h, subindex 1-8, unsigned16: 8-word analog outputs

7.1.3 Manufacturer-specific objects (2000h – 3FFFh)

These objects are created on request customer specifically.

7.2 Further Documentation

Internet: www.can-cia.org

CAN Specification 2.0, Part A & Part B

Notes