CANopen-DP/2

PROFIBUS-DP / CANopen-Gateway

Software Manual

to Product C.2909.02

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

N O T E

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

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

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

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

Manual file:	I:\Texte\Doku\MANUALS\CAN\CANopen-DP2\Englisch\CANopen-DP2_Software-Manual_en_10.wpd
Date of print:	2013-01-29

Described software version:	Rev. 1.00, cocbx_10

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
1.0	-	First English version of CANopen-DP/2 manual

Technical details are subject to change without notice.

Contents

Page

Contents

Page

1. Overview

1.1 About this Manual

This manual describes the local firmware of the CANopen-DP/2.

CANopen-Implementation

The firmware controls the communication of CAN-bus devices via CANopen® protocol [1] with the PROFIBUS-DP® (herein after referred to as PROFIBUS). The CANopen implementation supports CANopen-slave functions and basic CANopen-master functions (START, SYNC, Heartbeat).

The CANopen-DP/2 supports cyclic PROFIBUS transfers according to PROFIBUS DP-V0 [2] as well as acyclic PROFIBUS transfers according to DP-V1 [2]. The cyclic PROFIBUS transfers are for process data (PDOs) and the acyclic for the CANopen-parameter setting (SDOs).

CAN- Layer 2
The firmware supports the communication on the CAN-bus on the level of CAN-Layer-2.

11-Bit and 29-Bit Identifier
The module CANopen-DP/2 supports 11-bit and 29-bit CAN identifiers (CAN2.0A/B).
The CANopen protocol CiA® 301 [1] supports 11-bit-CAN identifiers only, so the support of the 29-bit identifiers is restricted to the CAN-Layer-2 functions.

1.2 Introduction into Functionality of the Firmware

The gateway simulates a slave device with a defined number of input and output bytes to the PROFIBUS. After the gateway has been configured CANopen nodes can be addressed as PROFIBUS slaves.

The PROFIBUS output bytes are transmitted to the CAN bus. One to eight output bytes are assigned to an TxPDO by means of PROFIBUS data exchange. Optionally the output bytes can be cyclically transmitted on the CAN bus.
RxPDOs are assigned to the input bytes on CAN side. Received CAN data is treated as input data by the PROFIBUS.

Note:
If the parameter *Slave PDO-Orientation* is set to ‘yes’ in the parameter telegram, CAN frames are transmitted with standard TxPDO-identifiers on the outputs and CAN frames are received with standard RxPDO-identifiers on the inputs.
If the parameter *Slave PDO-Orientation* is ‘no’ (default), the CAN frames are transmitted via RxPDO-identifiers and they are received via TxPDO-identifiers.

The PROFIBUS station address is set directly at the CANopen-DP/2 module by means of coding switches.

1.3 Configuration via PROFIBUS-DP

The CANopen-DP/2 module is configured via the PROFIBUS. The Siemens SIMATIC Manager for S7, for example, can be used as a configuration tool. Here, the gateway is assigned with logical modules from which the CANopen Node-ID and the PDO-identifier result.
In the course of the configuration further parameters such as the PLC address, data direction and data length are assigned to the modules.

2. Functionality of the Local Firmware

The following figure represents the functionality of the firmware.

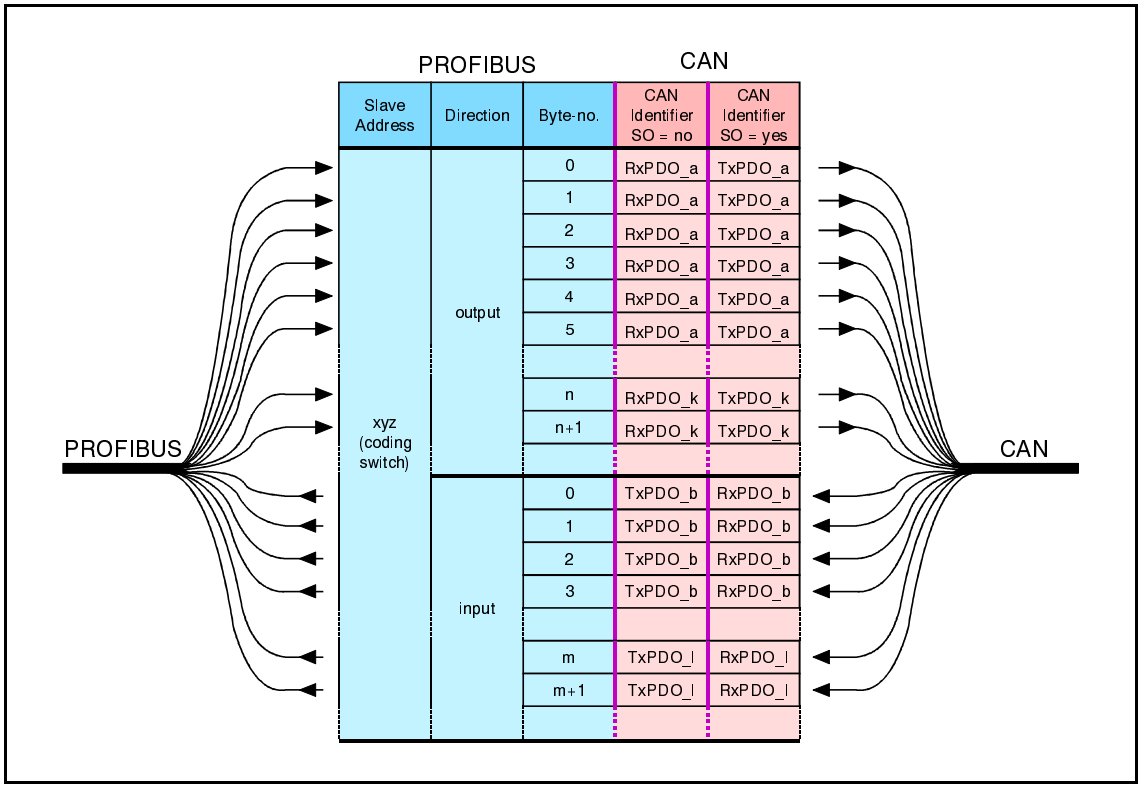

Fig. 1: Overview of functions of the CANopen-DP/2 firmware

Note:
If the parameter *Slave PDO-Orientation* is set to 'yes' in the parameter telegram, CAN frames are transmitted at the outputs with standard TxPDO-identifiers and CAN frames are received at the inputs with standard RxPDO-identifiers.
If the parameter *Slave PDO-Orientation* is 'no' (default), the CAN frames are transmitted via RxPDO-identifiers and they are received via TxPDO-identifiers.

2.1 PROFIBUS-Slave Address

The CANopen-DP/2 device simulates a slave module on the PROFIBUS side. The slave address is set by means of coding switches at the CANopen-DP/2. When switching on the module the hexadecimal PROFIBUS address is requested. The settings have to be changed before switching the module on, because changes are ineffective during operation.
The address range which can be set is *hexadecimal* 03_h to $7C_h$ or *decimal* 3 to 124. If an address is set which is smaller than 3_h, address 3 is valid. If an address is set which is larger than $7C_h$ or larger than 124 (decimal), address 124 is valid.

The left coding switch (HIGH) is used to set the MSBs, while the LSBs are set by means of the right

coding switch (LOW) (refer to the hardware manual, chapter:"Setting of the PROFIBUS-Address via Coding Switches"[3]).

Note:
The PROFIBUS-slave address can *only* be set via coding switches. It *cannot* be programmed by means of a class 2 master via the command 'Set_Slave_Address'.

2.2 User Data

The CANopen-DP/2-module is equipped with a VPC3-type DP-slave controller and simulates a total of up to 240 bytes per data direction on the PROFIBUS.

During a PROFIBUS data exchange one to eight bytes each are assigned to a TxPDO or RxPDO. Optionally the bytes, assigned to a Tx-Identifier, can be transmitted cyclically on the CAN bus.

2.3 Watchdog

The firmware can be run with activated or deactivated watchdog. It is recommendable, though, to run it with activated watchdog.

2.4 Diagnostics

The status of the LED displays and the DP-slave diagnostics can be used for diagnostics. The module supports five device-related diagnostic-bytes and if the heartbeat function is used 16 additional identifier-related diagnostic-bytes.
The diagnostics will be described in more detail on page 14.

2.5 Parameter Telegram (CAN Bit Rate)

In addition to the seven standard bytes of the configuration, the CANopen-DP/2 module supports 16 device-specific bytes. Here, the DP master can change e.g. the CAN bit rate. Setting the bit rate by means of the parameter telegram is described on page 26.

2.6 Global-Control Services (FREEZE, SYNC, UNSYNC)

The Global-Control services are not yet implemented.

2.7 PROFIBUS-DP Profiles

The PROFIBUS-DP profiles are not being supported yet.

3. Implementing and Diagnostics

3.1 Prerequisites for Implementation

This chapter describes the implementation of the CANopen-DP/2 module at a PROFIBUS which is controlled by a Siemens SIMATIC-S7-300 or S7-400.
In order to be able to implement the module as described here, you need the configuration program 'SIMATIC-Manager' with the tool 'HW-configurator'.

Note:
Configure the CANopen-DP/2 module absolutely first with the PLC via the SIMATIC-Manager as described in chapter: "5.1 Course of Configuration via the SIMATIC Manager". Only after the configuration is carried out, the CANopen-DP/2 module can be identified as CAN device!

3.2 Implementation

3.2.1 Procedure

Please make the following steps to implement the module:

1	Install and wire the CANopen-DP/2 module (PROFIBUS, power supply, CAN bus, see hardware manual).
2	Set the PROFIBUS address of the CANopen-DP/2 module via coding switches (HIGH = higher order bits, LOW = low-order bits)
3	Connect the PROFIBUS connector to the PROFIBUS interface of the CANopen-DP/2 module.
4	Configure the settings of the CANopen-DP/2 module in the PLC via the SIMATIC manager
5	Switch on the power supply for the CANopen-DP/2. Now the module has to run. The CANopen-DP/2 module is now automatically configured via the PLC.

Note:
Take into account that in particular the CAN bit rate and the module ID (CANopen) must be set via the PROFIBUS.

3.2.2 Start-Up

After switching on the power supply, the CANopen-DP/2 module starts automatically. It does not have its own mains switch.
During start-up the LEDs “E” (PROFIBUS-DP status) and “D” (Profibus-DP data exchange) are flashing. The PROFIBUS address set via the coding switches is read in.
The CANopen-DP/2 receives projection data from the DP master and evaluates the specifications in them. If the projection complies with the structure, the CANopen-DP/2 device starts the data transfer.

3.2.3 Data Transfer

If the CANopen-DP/2 device is configured, after start-up the CANopen-DP/2 unit transmits on the CAN bus a message with the data content '0' on every PDO defined as output. Thus the PDOs are initialised with the value '0'.
After that the data transfer starts automatically: If the PLC master changes transmission data of an output, the data is transmitted from the CANopen-DP/2 module to the CAN bus. If the CANopen-DP/2 module receives data on the CAN bus, then it provides the data for the PLC master.
The configuration is described in chapter: “ 5.1 Course of Configuration via the SIMATIC-Manager” from page 24.

3.3 Diagnostics via LED Indication

The function of LEDs has been defined by the firmware. In normal operation at least one of the four LEDs is flashing or constantly on.

The flash sequences which are listed in the following table are repeated about every six seconds.

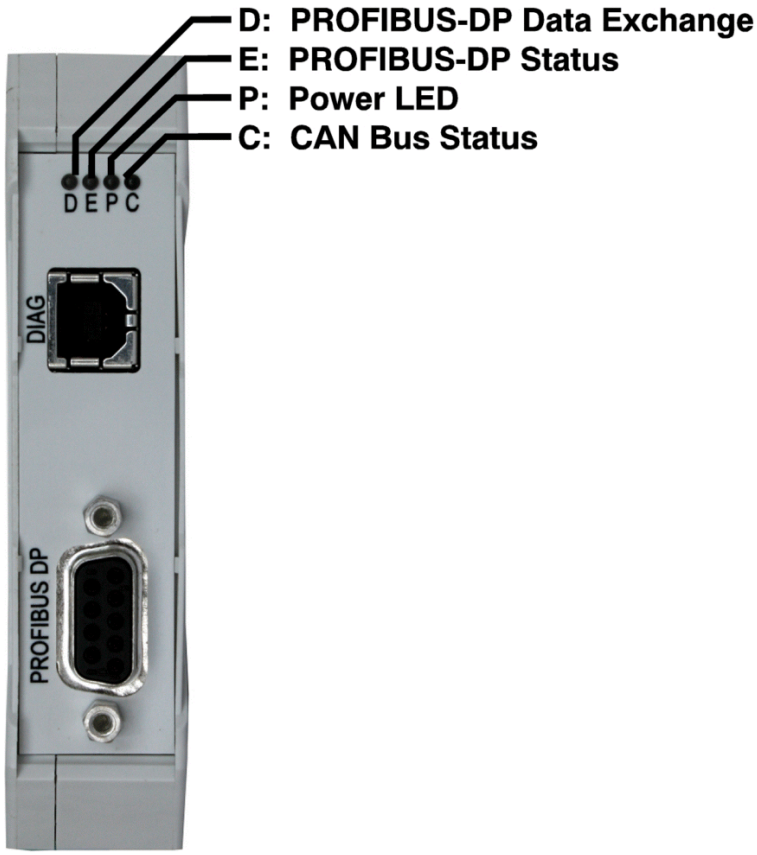

Fig. 2: Position of the LEDs

LED	Function	Status	Meaning	Error handling
D (green)	PROFIBUS-data exchange	off	no data exchange	-
		on	data exchange via PROFIBUS	-
E (red)	PROFIBUS-DP status	off	PROFIBUS OK	-
		1x short flash	looking for bit rate	the connection to the DP master has failed, check the PROFIBUS connection (fault in wiring in PROFIBUS cable, short circuit, terminating impedance in wrong position ?)
		2x short flashes	bit rate is monitored	check the PROFIBUS address specified
		3x short flashes	waiting for parameter telegram	parameter telegram is faulty. Diagnostics via SIMATIC-Manager or system function SFC13 (DPNRM_DG) (see chap. 3.4)
		4x short flashes	waiting for configuration telegram	configuration telegram is faulty. Diagnostics via SIMATIC-Manager or system function SFC13 (DPNRM_DG) (see chap. 3.4)
		on	malfunction	internal error
P (green)	Power	off	initialization **not** completed	-
		on	initialization completed successfully	-
C (green)	CAN bus status	off	no power supply	check the 24 V power supply
		1x short flash	CAN error (morse signal ‘E’)	malfunction on CAN bus check the wiring and the bit rate, see also hardware manual
		3x long flash	CAN off (morse signal ‘O’)	
		short-long-long	CAN warning (‘W’)	
		on	CAN bus OK	-

Table 1: LED status

3.4 Slave Diagnostics

In addition to the six diagnostic bytes predefined in standard DIN EN 19245, part 3, the CANopen-DP/2 supports five further device-related diagnostic bytes and when using the heartbeat function 16 further identifier related diagnostic bytes.

The slave diagnostic can be requested by the following function components:

Automation device family	Number	Name
SIMATIC with IM 308-C SIMATIC S7/M7	FB 192 SFC 13	FB IM308C SFC DPNRM_DG

Table 2: Function component for requesting the slave diagnostic

3.4.1 Diagnostic Bytes 1...6

The assignment of these diagnostic bytes has been predefined in standard DIN EN 19425, part 3. Below, the status messages will be described in consideration of the CANopen-DP/2 module.

The following designations will be used for this:

Byte number	Status-byte designation
1	station status 1
2	station status 2
3	station status 3
4	master-PROFIBUS address
5	manufacturer-identification high byte
6	manufacturer-identification low byte

Table 3: Diagnostic bytes 1...6

3.4.1.1 Station Status 1

Station status 1 contains error messages of the DP slave. If a bit is ‘0’, no error applies. A bit set to ‘1’ signalizes an error.

Bit	Error message if bit = ‘1’	Error handling
0	DP slave cannot be addressed by the master	- correct PROFIBUS address set at the CANopen-DP/2? - bus connector correctly wired? - power supply available at CANopen-DP/2? - power off/power on executed at CANopen-DP/2 in order to read in DP address?
1	DP slave is not yet ready for data exchange	- wait until the CANopen-DP/2 has completed start up
2	The configuration data transmitted from DP master to DP slave do not correspond to the DP slave structure.	- check whether the station type and the CANopen-DP/2 structure have been correctly entered via the configuration tool
3	The slave has got external diagnostic data.	- request and evaluate external diagnostic data
4	The requested function is not being supported by the DP slave.	- check projecting
5	DP master cannot interpret the response of the DP slave.	- check bus structure
6	Wrong setting.	- evaluate diagnostic bytes 10 and 11
7	DP slave has already been set by another master.	- this bit is always ‘1’, if you, e.g., just access the CANopen-DP/2 by means of a PG or another DP- master. The PROFIBUS address of the setting master is in the diagnostic byte ‘Master-PROFIBUS address’.

Table 4: Bits of station status 1

3.4.1.2 Station Status 2

Station status 2 contains status messages to the DP slave. If a bit is ‘1’, the corresponding message is active. A bit set to ‘0’ signalizes an inactive message.

Bit	Error message if bit = ‘1’
0	DP slave has to be set again.
1	A diagnostic message occurred. The DP slave cannot operate until the error has been removed (static diagnostics).
2	This bit is always ‘1’.
3	The watchdog for the CANopen-DP/2 is activated.
4	DP-slave has received freeze command.
5	DP-slave has received SYNC command.
6	This bit is always ‘0’.
7	DP-slave is deactivated.

Table 5: Bits of station status 2

3.4.1.3 Station Status 3

Station status 3 is reserved and without significance for the CANopen-DP/2.

3.4.1.4 Diagnostic Byte 4: Master-PROFIBUS Address

The PROFIBUS address of the master, which was the last to set the DP slave and has got reading and writing access to the DP slave, is stored in this byte.

3.4.1.5 Diagnostic Bytes 5 and 6: Manufacturer Identification

The manufacturer identification has been coded into two bytes. For the CANopen-DP/2 module the designation $\mathbf{098E_h}$ is returned.

3.4.2 External (Device Related) Diagnostic Bytes

The CANopen-DP/2 module supports diagnostic bytes 7 to 11 for module-specific diagnostic messages.

Diagnostic bytes	Meaning
1...6	defined in the PROFIBUS specification (see previous chapter)
7	length specification for module-specific diagnostic information (here always 5)
8	header byte: bits 0...5 contain the block length including header (here always 4)
9	DP service (SAP) which led to error
10	depending on status of byte 9: byte 9 = $3D_h$ setting (SAP61) faulty, byte 10 contains the number of the faulty setting byte byte 9 = $3E_h$ configuration (SAP62) faulty, byte 10 contains the number of the faulty PROFIBUS module (= address of the simulated PLC module)
11	depending on status of byte 9: byte 9 = $3D_h$ setting (SAP61) faulty, byte 11 shows the PROFIBUS master with the correct values byte 9 = $3E_h$ configuration (SAP62) faulty 3 wrong length correct: length = word 1 ... 16 byte 1 ... 32 4 only one byte has been specified for identifier 5 format specification is missing 6 wrong identifier 7 Node-ID conflict: the Node-ID of another CANopen module, which is determined by the selection of the slot and the Node-ID of the CANopen-DP/2-Gateway are identical 8 wrong PDO length selected

Table 6: Module-specific diagnostic bytes

3.4.3 Identifier Related Diagnostic Bytes

The CANopen-DP/2 unit uses the diagnostic bytes 7 to 23 for heartbeat-diagnostic messages.

Diagnostic Byte	Meaning
1...6	defined in the PROFIBUS specification (see preceding chapters)
7	= 51_h (identifier related diagnostics, length 17 bytes)
8	status of the heartbeat for CANopen node with Node-ID = 1... Node-ID = 8: bit 0: heartbeat-status Node-ID = 1 bit 1: heartbeat-status Node-ID = 2 : bit 7: heartbeat-status Node-ID = 8
9	status of the heartbeat for CANopen-node with Node-ID = 9... Node-ID = 16: bit 0: heartbeat-status Node-ID = 9 bit 1: heartbeat-status Node-ID = 10 : bit 7: heartbeat-status Node-ID = 16
:	:
23	status of the heartbeat for CANopen-node with Node-ID = 121... Node-ID = 128: bit 0: heartbeat-status Node-ID = 121 bit 1: heartbeat-status Node-ID = 122 : bit 6: heartbeat-status Node-ID = 127 bit 7: reserved

Table 7: Identifier related diagnostic bytes

Value of the heartbeat status bit	Meaning
0	Participant is not available or participant is available and takes part in heartbeat.
1	Participant is available but heartbeat failed.

3.5 Identification

The hard- and software status of the CANopen-DP/2 can be requested by means of the so called I&M0 services (Identification & Maintenance). This can be made by a sequence of DPV1-write- and read services (see chapter 6.6) or by requesting the status of the module with the hardware configurator. Therefor select the connected CANopen-DP/2 in the hardware-configurator and in menu "PLC", "Module Information". The following window is shown:

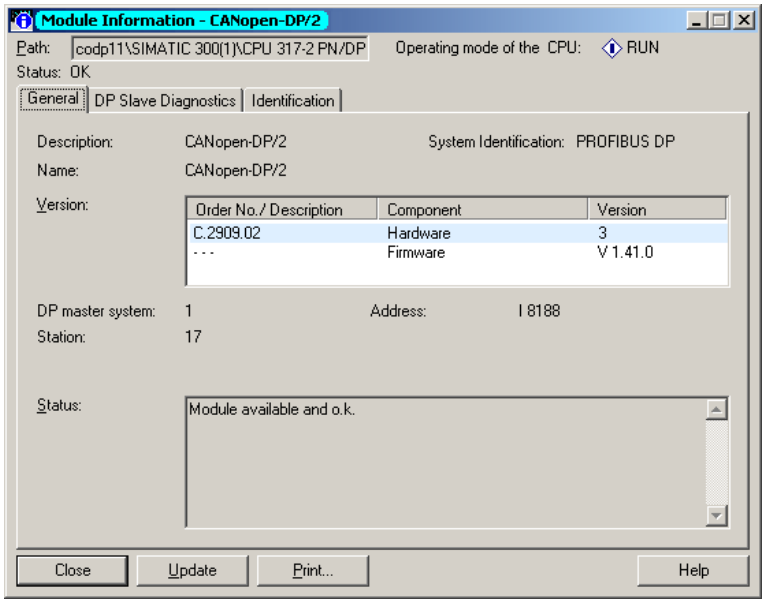

Fig. 3: *Module Information/General*

With *Module Information/General* the hardware- and firmware versions can be read.
To read the serial number choose the tab *Identification*.

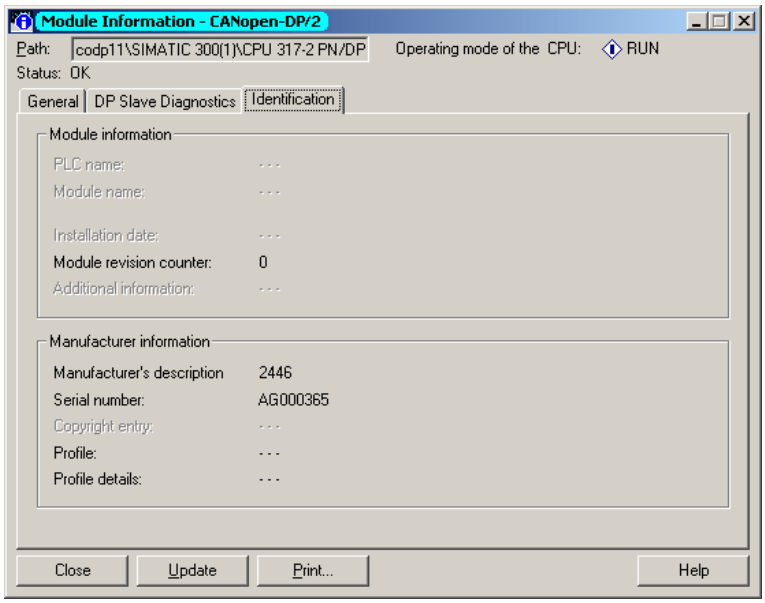

Fig. 4: *Module Information/Identification*

4. GSD File

Below, the GSD file (Device Master Data File) of the CANopen-DP/2 module has been printed. The specification printed here are for orientation. Decisive is the data contained in the GSD file **`codp098e.gsd`**., included in the product package.

```
;==========================================================================================
; (c) esd electronic system design GmbH Hannover
;
; PROFIBUS-DP Geraetestammdatei
; Version: 1.8
;
; Autor: Ulrich Hartmann
; Erstellungsdatum: V1.0   01.03.2005  uh
;                   V1.1   11.05.2005  uh   Heartbeat configuration added
;                   V1.2   03.06.2005  uh   C1_Response_Timeout and SDO Timeout changed to 10s
;                   V1.3   29.01.2007  uh   new flag 'slave pdo'
;                   V1.4   05.06.2008  uh   TxPDO cycle
;                   V1.5   08.05.2009  uh   Identification & Maintenance added;
;                   V1.6   30.12.2010  uh   Data counter added
;                   V1.7   22.11.2012  uh   Modul_offset corrected, Firmware version updated
;                   V1.8   04.01.2013  uh   Changed for new CANopen-DP/2
==========================================================================================
; Art des Parameters
; (M) Mandatory (zwingend notwendig)
; (O) Optional (zusõtzlich m÷glich)
; (D) Optional mit Default=0 falls nicht vorhanden
; (G) mindestens einer aus der Gruppe passend zur entsprechenden Baudrate
#PROFIBUS_DP
;--- Kapitel 2.3.2     Allgemeine DP-Schluesselwoerter ---
GSD_Revision            = 5                 ; (M ab GSD_Revision 1) (Unsigned8)
Vendor_Name             = "esd"             ; (M) Herstellername (Visible-String 32)
Model_Name              = "CANopen-DP/2"    ; (M) Herstellerbezeichnung des DP-Geraetes (Visible-String 32)
Revision                = "V1.0"            ; (M) Ausgabestand des DP-Geraetes (Visible-String 32)
Revision_Number         = 8                 ; (M ab GSD_Revision 1) (Unsigned8 (1 bis 63)) (1234)
Ident_Number            = 2446              ; (M) Gerõtetyp des DP-Gerõtes (Unsigned16)
Protocol_Ident          = 0                 ; (M) Protokollkennung des DP-Geraetes  0: Profibus-DP (Unsigned8)
Station_Type            = 0                 ; (M) DP-Geraetetyp  0: DP-Slave (Unsigned8)
FMS_supp                = 0                 ; (D) kein FMS/DP-Mischgeraet (Boolean)
Hardware_Release        = "V1.0"            ; (M) Hardware Ausgabestand des DP-Geraetes (Visible-String 32)
Software_Release        = "V1.0 "           ; (M) Software Ausgabestand des DP-Geraetes (Visible-String 32)
9.6_supp                = 1                 ; (G)   9,6  kBaud wird unterstuetzt
19.2_supp               = 1                 ; (G)  19,2  kBaud wird unterstuetzt
;31.25_supp             = 1                 ; fuer Gateway CAN-CBM-DP nicht moeglich (1234)
45.45_supp              = 1                 ; (G ab GSD_Revision 2)  45,45 kBaud wird unterstuetzt
93.75_supp              = 1                 ; (G)  93,75 kBaud wird unterstuetzt
187.5_supp              = 1                 ; (G) 187,5  kBaud wird unterstuetzt
500_supp                = 1                 ; (G) 500    kBaud wird unterstuetzt
1.5M_supp               = 1                 ; (G)   1,5  MBaud wird unterstuetzt
3M_supp                 = 1                 ; (G ab GSD_Revision 1)   3    MBaud wird unterstuetzt
6M_supp                 = 1                 ; (G ab GSD_Revision 1)   6    MBaud wird unterstuetzt
12M_supp                = 1                 ; (G ab GSD_Revision 1)  12    MBaud wird unterstuetzt
MaxTsdr_9.6             = 60                ; (G)
MaxTsdr_19.2            = 60                ; (G)
;MaxTsdr_31.25          = 15                ; fuer Gateway CAN-CBM-DP nicht moeglich (1234)
MaxTsdr_45.45           = 60                ; (G ab GSD_Revision 2)
MaxTsdr_93.75           = 60                ; (G)
MaxTsdr_187.5           = 60                ; (G)
MaxTsdr_500             = 100               ; (G)
MaxTsdr_1.5M            = 150               ; (G)
MaxTsdr_3M              = 250               ; (G ab GSD_Revision 1)
MaxTsdr_6M              = 450               ; (G ab GSD_Revision 1)
MaxTsdr_12M             = 800               ; (G ab GSD_Revision 1)
Redundancy              = 0                 ; (D) keine redundante Uebertragungstechnik
Repeater_Ctrl_Sig       = 0                 ; (D) RTS-Signalpegel (CNTR-P) Pin 4 des 9pol. SUB-D
                                            ; 0: nicht vorhanden     1: RS 485     2: TTL
24V_Pins                = 0                 ; (D) Bedeutung der 24V Pins des 9pol. SUB-D (Pin 7  24V;  Pin 2 GND)
                                            ; 0: nicht angeschlossen     1: Input     2: Output
                                            ; Implementation_Type = "Visible-String" ; (1234)
Bitmap_Device           = "CDPS00_N" ; (O ab GSD_Revision 1)
Bitmap_Diag             = "CDPS00_D" ; (O ab GSD_Revision 1)
Bitmap_SF               = "CDPS00_S" ; (O ab GSD_Revision 1)
```

```
;--- Kapitel 2.3.4     DP-Slave-bezogene Schluesselwoerter ---
Freeze_Mode_supp                  = 0      ; (D) Der Freeze-Mode wird nicht unterstuetzt
Sync_Mode_supp                    = 0      ; (D) Der Sync-Mode wird nicht unterstuetzt
Auto_Baud_supp                    = 1      ; (D) Die Automatische Baudratenerkennung wird unterstuetzt
Set_Slave_Add_supp                = 0      ; (D) Die Slave-Adresse kann vom Master nicht gesetzt werden
;User_Prm_Data_Len                = 9      ; (D) Hoechstlaenge von User-Parameter-Daten
;User_Prm_Data=0x00,0x06,0x00,0x00,0x00,0x00,0x00,0xff,0xff,0xff  ; (O) User-Parameter-Daten ( byte 7 - 15 )
Min_Slave_Intervall               = 6      ; (M) Minimaler Abstand zwischen 2 DDLM_Data_Exchange-Aufrufen (xx * 100us)
Modular_Station                   = 1      ; (D) 0: Kompaktstation  1: Modulare Station
Max_Module                        = 244    ; (M falls modulare Station) Hoechstanzahl der Module einer Modularen Station
Max_Input_Len                     = 240    ; (M falls modulare Station) Hoechstlaenge der Eingangsdaten einer Modularen Station
Max_Output_Len                    = 240    ; (M falls modulare Station) Hoechstlaenge der Ausgangsdaten einer Modularen Station
Max_Data_Len                      = 465    ; (O nur falls modulare Station) Groesste Summe der Ein- und Ausgangsdaten einer Modularen Station in Bytes
DPV1_Slave                        = 1      ; supports DPV1
Ident_Maintenance_Supp            = 1      ; I&M Data supported
C1_Max_Data_Len                   = 240    ;
C1_Response_Timeout               = 1100   ; Timeout = 1100 * 10ms = 11s
C1_Read_Write_supp                = 1      ;
C1_Read_Write_required            = 0
Diagnostic_Alarm_supp             = 0      ;
Process_Alarm_supp                = 0      ;
Pull_Plug_Alarm_supp              = 0      ;
Status_Alarm_supp                 = 0      ;
Update_Alarm_supp                 = 0      ;
Manufacturer_Specific_Alarm_supp  = 0      ;
Extra_Alarm_SAP_supp              = 0      ;

Module= "Empty" 0x00
1
EndModule
Max_Diag_Data_Len                 = 24     ; max. 24 Byte Diagnosedaten
Modul_Offset                      = 1      ; (D ab GSD_Revision 1) erste Steckplatznummer
Max_User_Prm_Data_Len             = 20
PrmText=1
Text(0)="1000   kbit/s"
Text(1)=" 666.6 kbit/s"
Text(2)=" 500   kbit/s"
Text(3)=" 333.3 kbit/s"
Text(4)=" 250   kbit/s"
Text(5)=" 166   kbit/s"
Text(6)=" 125   kbit/s"
Text(7)=" 100   kbit/s"
Text(8)="  66.6 kbit/s"
Text(9)="  50   kbit/s"
Text(10)="  33.3 kbit/s"
Text(11)="  20   kbit/s"
Text(12)="  12.5 kbit/s"
Text(13)="  10   kbit/s"
EndPrmText
PrmText=2
Text(0)="No"
Text(1)="Yes"
EndPrmText
PrmText=3
Text(0)="Yes"
Text(1)="No"
EndPrmText
ExtUserPrmData=1 "CAN-Bitrate"
Unsigned8 6 0-13
Prm_Text_Ref=1
EndExtUserPrmData
ExtUserPrmData=2 "Communication Window"
Bit(7) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
ExtUserPrmData=3 "Slave PDO orientation"
Bit(6) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
ExtUserPrmData=4 "Heartbeat Configuration"
Bit(5) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
ExtUserPrmData=5 "RTR-Frames"
Bit(4) 0 0-1
Prm_Text_Ref=3
EndExtUserPrmData
ExtUserPrmData=6 "CANopen-Slave"
Bit(3) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
```

```
ExtUserPrmData=7 "CANopen-Master"
Bit(2) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
ExtUserPrmData=8 "Start-Frame"
Bit(1) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
ExtUserPrmData=9 "Page-Mode"
Bit(0) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
ExtUserPrmData=10 "ModuleID"
Unsigned8 1 1-127
EndExtUserPrmData
ExtUserPrmData=11 "WakeUp Time (0=Off, 255=Default)"
Unsigned8 255 0-255
EndExtUserPrmData
ExtUserPrmData=12 "Sync Time (0=Off, 65535=Default)"
Unsigned16 65535 0-65535
EndExtUserPrmData
ExtUserPrmData=13 "Heartbeat Consumer Time (0=Off)"
Unsigned16 0 0-65535
EndExtUserPrmData
ExtUserPrmData=14 "Heartbeat Producer Time (0=Off)"
Unsigned16 0 0-65535
EndExtUserPrmData
ExtUserPrmData=15 "SDO Timeout"
Unsigned16 1000 1-10000
EndExtUserPrmData
ExtUserPrmData=16 "MPDO Identifier"
Unsigned16 385 385-1791
EndExtUserPrmData
ExtUserPrmData=17 "TxPDO cycle Time (0=Off)"
Unsigned16 0 0-65535
EndExtUserPrmData
ExtUserPrmData=18 "Rx-Counter"
Bit(0) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
ExtUserPrmData=19 "Tx-Counter"
Bit(1) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
Ext_User_Prm_Data_Const(0)=0x80,0x00,0x00,0x06,0x00,0x00,0x00,0x00,0xff,0xff,0xff,0x00,0x00,0x00,0x00,0x03,0xE8,0x01,0x81,0x00
Ext_User_Prm_Data_Ref(3)=1
Ext_User_Prm_Data_Ref(4)=2
Ext_User_Prm_Data_Ref(4)=3
Ext_User_Prm_Data_Ref(4)=4
Ext_User_Prm_Data_Ref(4)=5
Ext_User_Prm_Data_Ref(4)=6
Ext_User_Prm_Data_Ref(4)=7
Ext_User_Prm_Data_Ref(4)=8
Ext_User_Prm_Data_Ref(4)=9
Ext_User_Prm_Data_Ref(5)=10
Ext_User_Prm_Data_Ref(6)=17
Ext_User_Prm_Data_Ref(19)=18
Ext_User_Prm_Data_Ref(19)=19
Ext_User_Prm_Data_Ref(8)=11
Ext_User_Prm_Data_Ref(9)=12
Ext_User_Prm_Data_Ref(11)=13
Ext_User_Prm_Data_Ref(13)=14
Ext_User_Prm_Data_Ref(15)=15
Ext_User_Prm_Data_Ref(17)=16
Slave_Family = 9@CANopen@V01
;OrderNumber                                    = "C.2909.02"
```

5. Cyclic PROFIBUS Data Transfer to CANopen

The CANopen-DP/2 device is configured via the PROFIBUS. This chapter describes the configuration of the cyclic PROFIBUS data transfer of process data between PROFIBUS and CAN. Acyclic PROFIBUS data transfers can be performed as well, e.g. for the parameter passing. They are described from page 44 on.

5.1 Course of Configuration via SIMATIC Manager

Please follow the steps below to configure the CANopen-DP/2 device:

1. Select CANopen-DP/2
Select menu *Hardware Catalogue* and there *Additional Field Devices* and *Gateway*.
Select CANopen-DP/2 there.

2. Set PROFIBUS address
Set the PROFIBUS address as described in chapter 5.1.1 on page 25.

3. Parameter Telegram (set CAN bit rate, general configuration and CANopen module ID)
Configure the settings by means of the parameter telegram as described in chapter 5.1.2 on page 26.

4. Assignment of the Slots and setting the CANopen node-ID
Assign the slots as described in chapter 5.1.3 on page 35.

5. Configuration of the Slots (PLC address)
Configure the slots as described in chapter 5.1.4 on page 37.

6. Save settings on hard disk
Save the settings as described in chapter 5.1.5 on page 37.

5.1.1 Set PROFIBUS Address

A window opens in which you have to specify the PROFIBUS station address.

Attention!
The ***hexadecimal*** address set at the coding switches has to be ***converted*** into a ***decimal*** value and entered here!

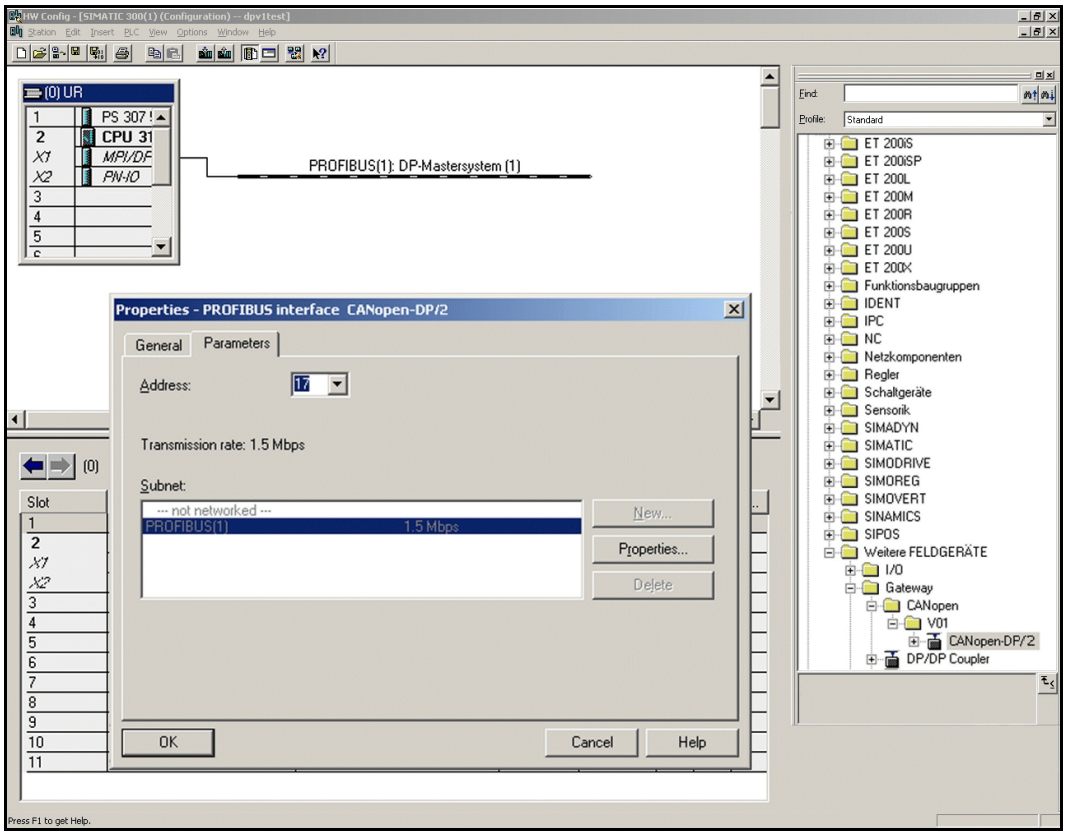

Fig. 5: Setting the PROFIBUS address of the CANopen-DP/2

5.1.2 Parameter Telegram

The module ‘DP slave’ is now automatically inserted in the configuration window.
Settings can be changed by means of the parameter telegram.

The parameter setting of the DP-Slave can be done in the Properties window. To open the Properties window double click the header of the DP-slave window (here ‘`(17) CANopen-DP/2`’).

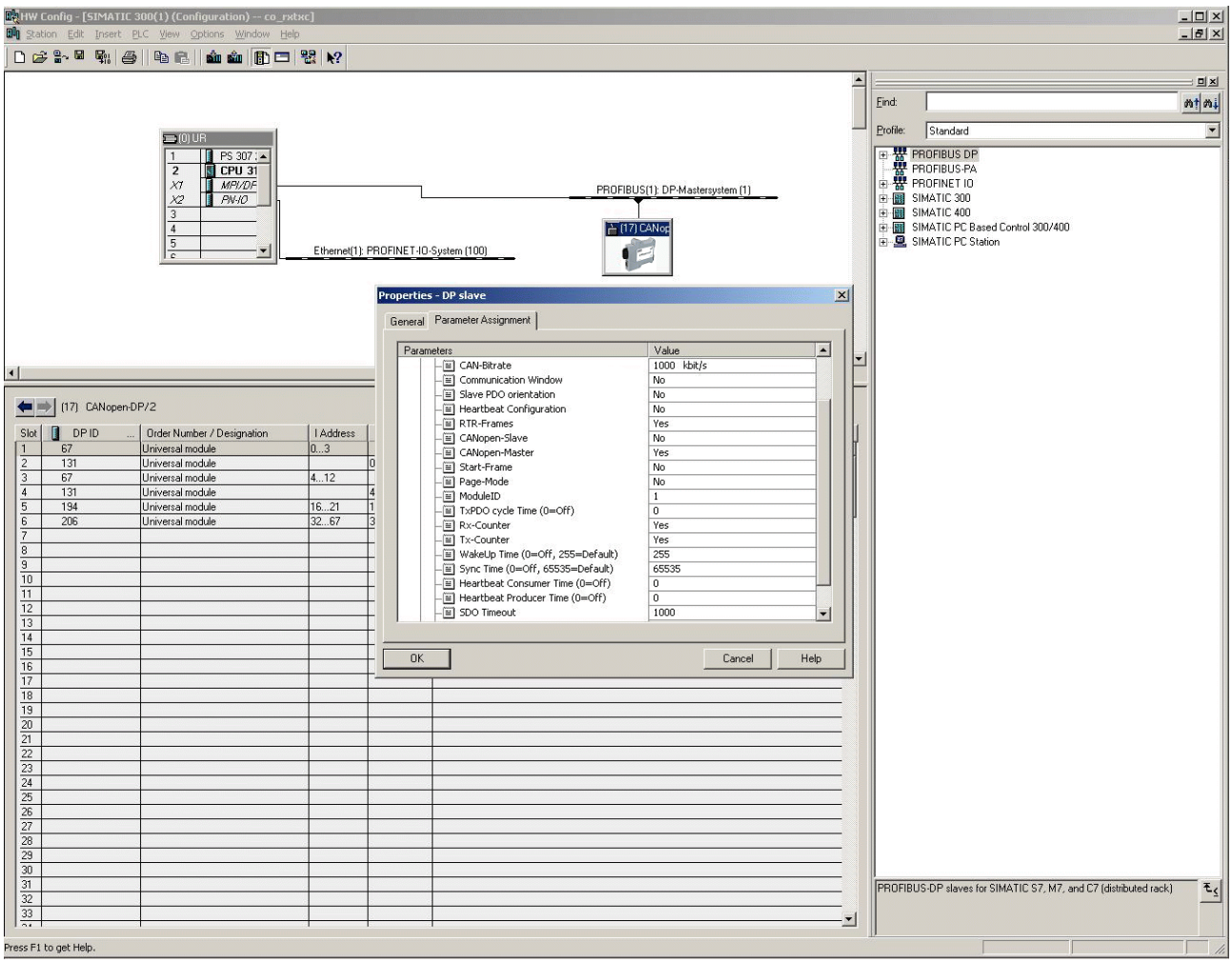

Fig. 6: Setting the parameters in the DP-slave properties window

Description of Parameters:

CAN-Bitrate **For the bit rate the following selections can be made:**

Bit rate [Kbit/s]	*Bit rate* [Kbit/s]
1000	100
666,6	66,6
500	50
333,3	33,3
250	20
166	12,5
125	10

Table 8: Setting the bit rate in 14 levels

Communication Window: (CW) This parameter activates the Communication Window. It is described in detail on page 76.

Slave PDO Orientation: (SO) If this parameter is 'yes', CAN frames with standard-TxPDO-identifiers are transmitted via the outputs and CAN-frames with standard RxPDO-identifiers are received via the inputs (see table 16 page 36).
If the parameter is 'no' (default), CAN-frames with RxPDO-identifiers are transmitted and CAN-frames with TxPDO-identifiers are received

Heartbeat Configuration: (HC) This parameter activates or deactivates the configuration of the Heartbeat monitoring and generation. It is described in detail on page 30.

RTR-Frames: (NR) Transmit RTR-frames for the Rx-identifiers configured via PROFIBUS.

CANopen-Slave: (CS) Configure gateway as CANopen slave.

CANopen-Master: (CM) Configure gateway as CANopen master.

Start-Frame: (AS) After wake-up time (in seconds) has expired, a start frame is transmitted, if the gateway is a master (autostart).

Page-Mode: (PM) Activate Page-Mode.
Page mode is described in detail in the CAN-DP/2 software manual [4]

Example for permissible combinations of the parameters:

CW	*SO*	*HC*	*NR*	*CS*	*CM*	*AS*	*PM*	Description
x	yes	no	yes	yes	no	x	no	- after wake-up time the module automatically transmits *128 dec + Module-No.* and is in 'Pre-Operational' status - after a start frame has been received: transmission of Tx-PDO, transmit RTR-frames on PDO-RxId - If parameter *SO* is 'yes', CAN-frames with standard-TxPDO-Identifiers are transmitted via the outputs and CAN-frames with standard-RxPDO-identifiers are received via the inputs.
x	yes	no	no	yes	no	x	no	- after wake-up time the module automatically transmits *128 dec + Module-No.* and is in 'Pre-Operational' status - after a start frame has been received: output of TxId - If parameter *SO* is 'yes', CAN-frames with standard-TxPDO-Identifiers are transmitted via the outputs and CAN-frames with standard-RxPDO-identifiers are received via the inputs.
x	no	no	yes	no	yes	no	no	- output of PDO-RxId after wake-up time - transmit RTR-frames on PDO-TxId
x	no	no	no	no	yes	no	no	- after wake-up time, output of PDO-RxID
x	no	yes/ no	yes	no	yes	yes	no	- after wake-up time start frame, output of PDO-RxID, transmit RTR-frames on PDO-TxId
x	no	yes/ no	no	no	yes	yes	no	- after wake-up time start frame, output of PDO-RxID

x... status of the parameter without meaning here

Table 9: Example for permissible settings

Module-ID: Node-ID of the Gateway as CANopen-node.
The Node-ID under which the gateway is addressed is set via this byte.
Value range: 1 ... 127 (decimal)

TXPDO cycle Time: The CANopen-DP/2 unit can optionally transmit all configured Tx-PDOs (see chapter 5.2) cyclically. The ***TxPDO cycle Time*** has to be entered in milliseconds as a decimal value.

Parameter	Value range [dec] in [ms]	Description
TXPDO cycle Time	0	No cyclic transmissions
	1...65535	cycle time in milliseconds (1...65535 ms)

Table 10: Function of parameter *TxPDO cycle Time*

RX-Counter=yes: In the last byte of the input module an 8-bit counter is counted up for each received CAN message on this identifier. In addition the length for each configured CAN message must be increased by one.
If for example a message with 8-byte data length shall be received, in the input window *Properties DP- Slave* the entry must be *length* = 9.
Hereby the receipt of each message, even if the data did not change, can be monitored.

Tx-Counter=yes: In the last byte of the output module a byte is reserved, in which e.g. a counter can be counted up. With each change of this byte the CAN message is sent, even if the data were not changed. Also here the length of the CAN message configured must be increased by one.
If for example a message with 8-byte data length shall be sent, in the input window *Properties DP- Slave* the entry must be *length* = 9.
The counter byte will not be sent.

Wakeup Time: Via the parameter *Wakeup Time* a delay in seconds is specified. It determines the time a module has to wait after a RESET or power-on, before it starts to transmit data on the CAN bus.

The *Wakeup Time* specified here, overwrites the value of the *Wakeup Time* stored previously in the CANopen-DP/2 gateway, if a value not equal ‘255’ was specified. If ‘255’ is specified, the value stored in the gateway will be used.

If parameter *Wakeup Time* is set to ‘0’, the module does not wait, but starts the transmission of data as soon as they are available.

The ***Wakeup Time*** is specified as a decimal value, here.

Parameter	Value range [dec] in [s]	Description
Wakeup Time	0	Wakeup Time function off
	1...254	Wakeup Time in seconds
	255	Use current value from gateway (default)

Table 11: Function of the parameter *Wakeup Time*

SYNC Time: The CANopen-DP/2 device transmits as CANopen master cyclically the command SYNC for simple CANopen applications.

The cycle is specified in milliseconds.

SYNC Time is specified as a decimal value, here.

Parameter	Value range [dec] in [ms]	Description
SYNC Time	0	No SYNC transmissions possible
	1...65534	SYNC time in milliseconds (1...65534 ms)
	65535	Use current value from gateway (default)

Table 12: Function of parameter *SYNC Time*

Heartbeat Consumer Time: The Heartbeat function can be used for mutual monitoring of the CANopen modules (particularly for detecting connection failures).

General Operation of the Heartbeat:
A module, the so-called heartbeat producer transmits cyclically a heartbeat message on the CAN bus with the node guarding identifier (700_h + Node-ID, see [1]). One or more heartbeat consumer receive the message.
If the message is not received within the heartbeat consumer time, a heartbeat event will be generated on the heartbeat consumer module. The heartbeat consumer time always has to be set to a higher value than the corresponding heartbeat producer time. A module can act as heartbeat consumer and producer at the same time.

Heartbeat Implementation on the CANopen-DP/2-Gateway:
The setting of the heartbeat parameters of the configured CANopen nodes (a configured node is e.g. slot 3 = Node-ID 3 on page 35) by the gateway can be activated or deactivated with the parameter *Heartbeat Configuration*. This parameter can be set in the same window as the *Heartbeat Consumer Time*.

If the parameter *Heartbeat Configuration* is set to YES, for <u>every</u> configured CANopen node the heartbeat producer time in CANopen object 1017_h of every node is set to the value of the parameter *Heartbeat Producer Time*.
In addition for <u>every</u> configured CANopen node the heartbeat consumer time in CANopen object 1016_h is set to the value of the parameter *Heartbeat Consumer Time* (the entry is always entered in sub-index 1 of object 1016_h [1]).

The figure 7 on page 31 shows the transmission of the heartbeat times via SDO-accesses in the objects of the configured nodes after the request of *Heartbeat Configuration*. The heartbeat time is also set on the CANopen-DP/2.
Before the heartbeat monitoring can be started by setting the parameter *Heartbeat Configuration* to YES, the *Parameter Heartbeat Producer Time* or/and *Heartbeat Consumer Time* have to be set to values unequal '0', otherwise the corresponding functions will not be activated.

Figure 8 on page 32 shows the heartbeat monitoring of the CANopen-DP/2 by the other CANopen modules: The CANopen-DP/2 is the heartbeat producer here. As heartbeat consumer the configured CANopen nodes monitor the transmission of the heartbeat of the CANopen-DP/2.
Figure 9 on page 32 shows the monitoring of the CANopen modules, which are configured as heartbeat producers by the CANopen-DP/2, which works as heartbeat consumer here.

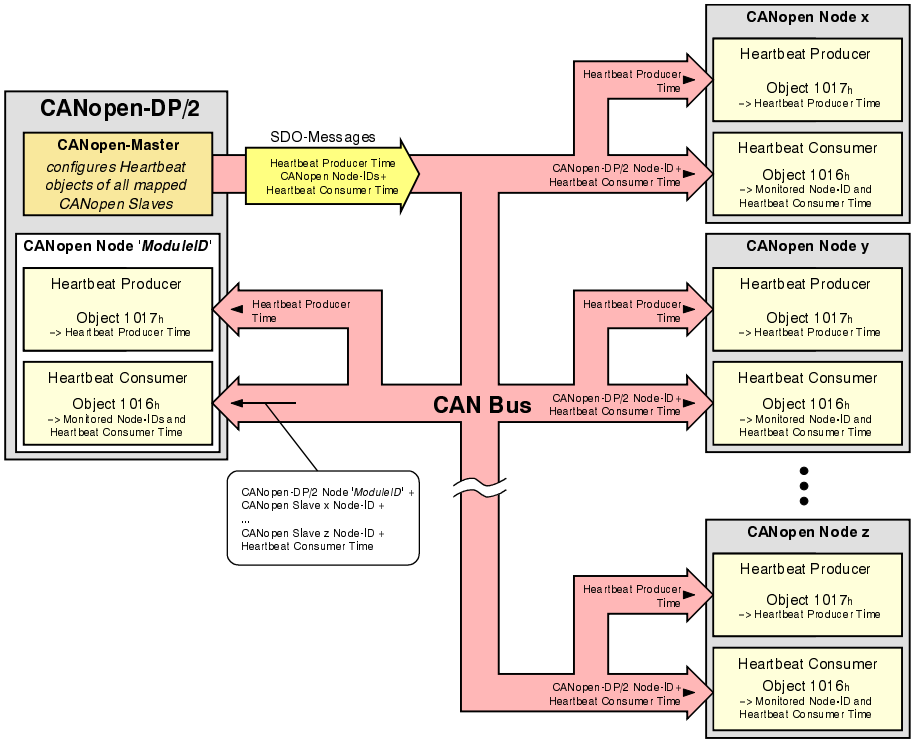

Fig. 7: Setting of the heartbeat parameters on the CANopen slaves after setting ***Heartbeat Configuration*** to **YES**

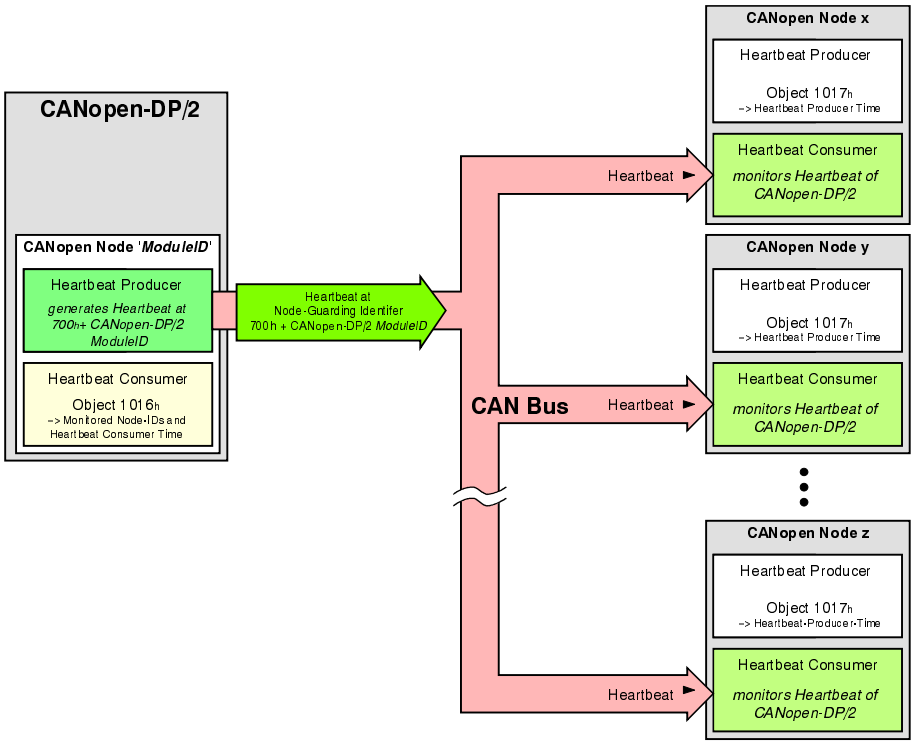

Fig. 8: Acting as heartbeat producer the CANopen-DP/2 is monitored by the configured CANopen nodes

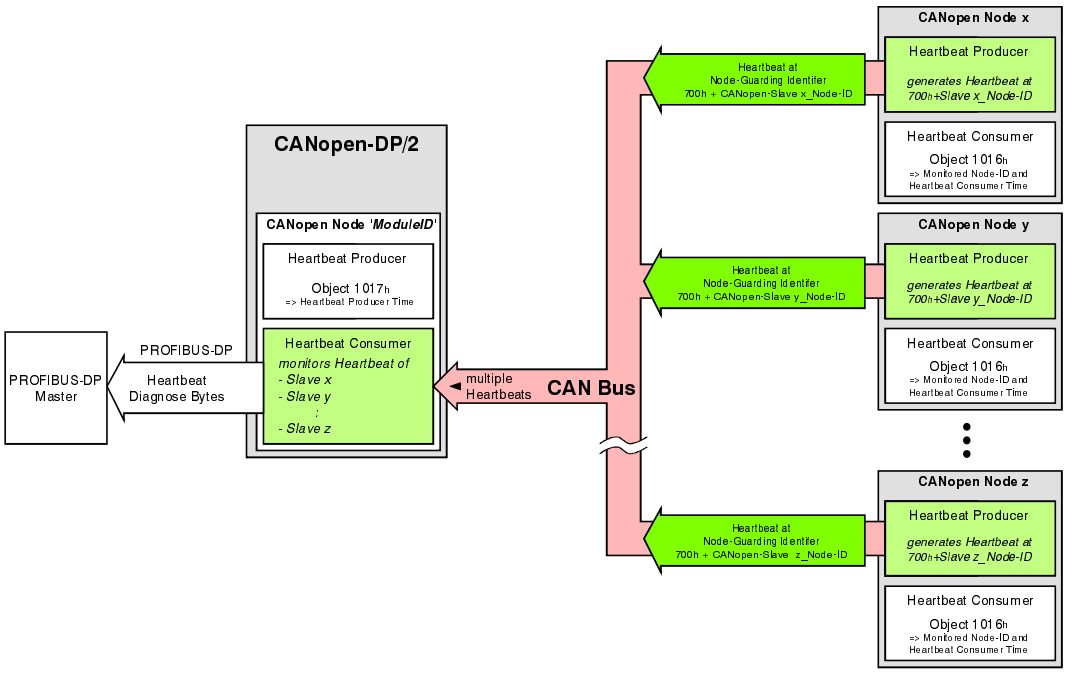

Fig. 9: Acting as heartbeat consumer the CANopen-DP/2 is monitoring the configured CANopen nodes, which are producing the heartbeats

Parameter	Value range [dec] in [ms]	Description
Heartbeat Consumer Time	0	no heartbeat consumer monitoring (= factory default setting)
	1...65535	Heartbeat Consumer Time (1...65535 ms)

Table 13: Value range of the *Heartbeat Consumer Time*

The CANopen-DP/2-Gateway can act as a heartbeat producer and heartbeat consumer at the same time. It monitors up to 126 heartbeat producers.

Caused by an interruption of the PROFIBUS-DP the heartbeat producer in the CANopen-DP/2 is stopped and after the resumption of the PROFIBUS-DP-communication it is started again.

After the first reception of a configured heartbeat telegram (blank node numbers are not monitored) and after failure of the heartbeat (no telegram within the *Heartbeat Consumer Time*) a PROFIBUS-diagnostic telegram is transmitted with the following data content:

Octet 1...6: complying with the standard
Octet 7: = 51_h (Identifier related diagnostics, length 17 byte)
Octet 8...23: Status of the heartbeat for node number 0...127
(1. byte: bit 0: *Heartbeat-Status node No. 0*,
bit 7: *Heartbeat-Status node No. 7*,
...
16. byte, bit 7: *Heartbeat-Status node No. 127*)

Heartbeat-Status node No. n =
0: Heartbeat evaluation is OK or
node is not configured for heartbeat

1: heartbeat is configured but *Heartbeat Consumer Time is* exceeded

Heartbeat Producer Time: Here the cycle time is defined, with which the CANopen-DP/2-Gateway, or the configured CANopen modules transmit a heartbeat frame on the node guarding identifier.
Setting the heartbeat producer time to ‘0’ stops the transmission of the heartbeat.
The heartbeat consumer time of the modules monitoring must always be higher than the heartbeat producer time of the heartbeat-transmitting module.

Caused by an interruption of the PROFIBUS-DP the Heartbeat-producer in the CANopen-DP/2 is stopped and after the resumption of the PROFIBUS-DP-communication it is started again.

Parameter	Value range [dec] in [ms]	Description
Heartbeat Producer Time	0	no heartbeat transmission (= factory default setting)
	1...65535	Heartbeat Producer Time (1...65535 ms)

Table 14: Value range of the *Heartbeat Producer Time*

SDO Timeout:

By means of the acyclic data transfer of the PROFIBUS (DP-V1) SDO-accesses on the CAN bus can be made.

Via the parameter *SDO Timeout* the monitoring of the SDO-transfer timeout can be activated, which causes the transmission of a PROFIBUS-diagnostic telegram when a timeout occurs.

If a SDO-read or SDO-write request is not answered within the given timeout-time with a corresponding response-message, a Profibus-DP diagnostic telegram with the *Error-Code 12* in Octet 3 will be transmitted (see page 52).

Parameter	Value range [dec] in [ms]	Description
SDO Timeout	1...10000	SDO-timeout-time (1...10000 ms) Default: 1000 ms

Table 15: Value range of the *SDO Timeout* time

MPDO Identifier:

Here the value of the multiplex PDO-identifier [1] is entered, on which the CANopen-DP/2 device transmits CAN frames to CANopen slaves.

The CANopen-DP/2 supports only the “Destination Address Mode” (DAM) with Node-ID = ‘0’, i.e. a broadcasting follows, at which <u>all</u> other modules are addressed.

Value range:
MPDO Identifier = 385 ...1791 (default value = 385)
= 181_h ... $6FF_h$ (default-value = 181_h)

5.1.3 Assignment of the Slots and Setting the CANopen Node-ID

Every simulated PLC-slot specifies a CANopen node. The *Node-ID* of a CANopen node results from the number of the slot selected.

> **Note:**
> Because the slots can only be assigned continuously, all slots up to the highest required *Node-ID* have to be selected.
> A maximum of 48 CANopen-nodes can be selected with one PDO respectively. Using several PDOs per node reduces the number of possible nodes.

The slots are selected, by double clicking the device ‘Universal Module’ in the hardware catalog with activated DP-slave window for every slot.

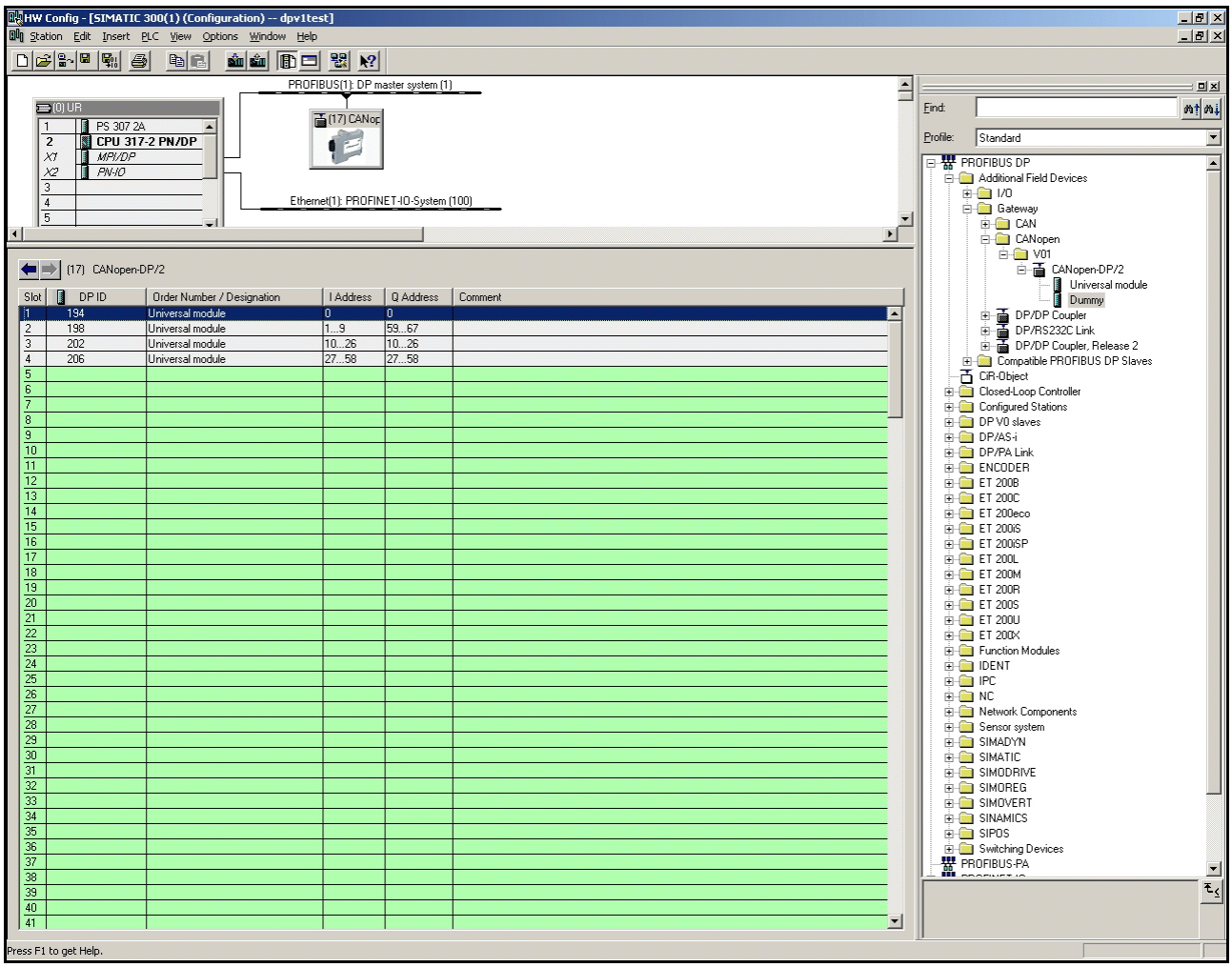

Fig. 10: Assignment of the slots and selection of the CANopen Node-ID

In the DP-slave window the assigned slots are represented by a ‘0’ in the row “DP ID”. In order to configure the selected slot in a properties window the entry has to be double clicked. Depending on the configuration the PLC enters a specific value for the DP-ID.

The standard PDOs result from the *Node-ID* according to CANopen protocol CiA 301 [1]:

PDO	Value
TxPDO1	180_h + *Node-ID*
TxPDO2	280_h + *Node-ID*
TxPDO3	380_h + *Node-ID*
TxPDO4	480_h + *Node-ID*
RxPDO1	200_h + *Node-ID*
RxPDO2	300_h + *Node-ID*
RxPDO3	400_h + *Node-ID*
RxPDO4	500_h + *Node-ID*

Table 16: Coding of the CANopen-PDOs

Note:
If the parameter *Slave PDO-Orientation* is set to 'yes' in the parameter telegram, CAN frames are transmitted at the outputs with standard TxPDO-identifiers and CAN frames are received at the inputs with standard RxPDO-identifiers.
If the parameter *Slave PDO-Orientation* is 'no' (default), the CAN frames are transmitted via RxPDO-identifiers and they are received via TxPDO-identifiers.

5.1.4 Configuration of a CANopen Node on a PLC-Slot

In order to configure the slots the slot entry has to be double clicked. The *Properties* window opens in which the simulated PLC-slot can be configured. To generate a CANopen node, in the properties window the I/O-Type “Out- input” has to be selected:

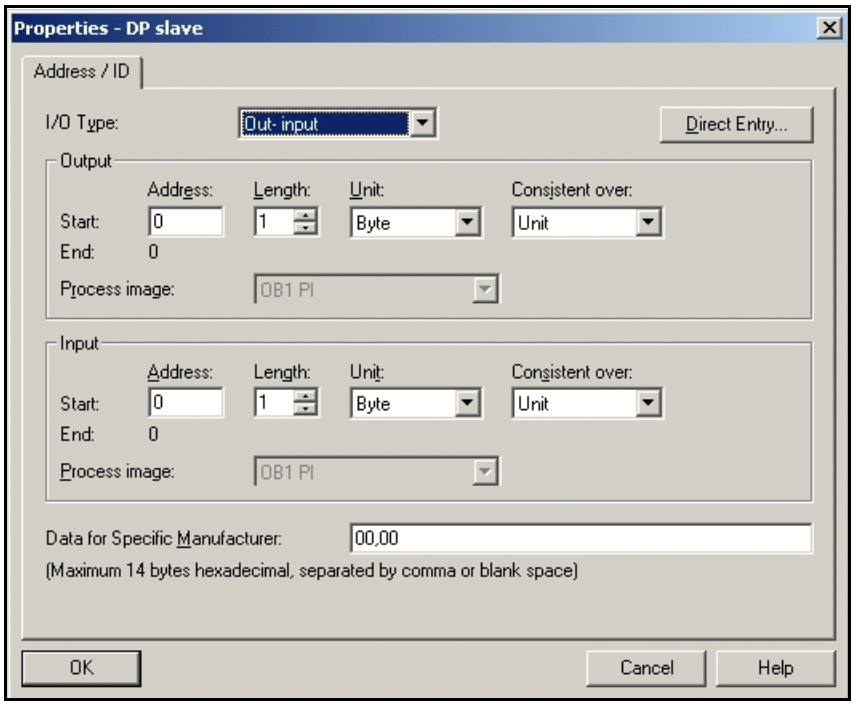

Abb. 11: Example: Configuration of a CANopen node

The individual parameters of the properties window are described in detail in the following chapter “Description of the properties window ‘*Properties - DP-Slave*’ ”.

5.1.5 Save settings to Hard Disk

Now you have to save the settings via the menu points *Station/Save* to hard disc. Afterwards the settings are transmitted to the PLC by means of the menu points *Target System/Load in Unit.*

5.2 Description of Input Window '*Properties - DP Slave*'

I/O-Type . . . . . To define a CANopen node for the selected PLC-slot, always the data direction 'Out- Input' has to be selected as ***I/O-Type***. (The firmware distinguishes between CANopen- and CAN-Layer-2 level data exchange by evaluation of this parameter.)

Address . . . . . In the ***Address*** field the PLC-I/O address is entered as a **decimal value**. The entries for inputs and outputs can be different.

Length, ***Unit*** . . . . . By means of the fields ***Length*** and ***Unit*** the number of data bytes for the DP-Slave is defined.
The entered length must comply with the sum of the data bytes, which are transmitted via the parameter *Length RxPDOx/TxPDOx* in the field *Data for specific Manufacturer* (see page 41).

Value range:

Unit = byte	=>	*Length* = 1...32 for *Rx*- or *Tx-Counter=yes* up to 36
Unit = word	=>	*Length* = 1...16 for *Rx*- or *Tx-Counter=yes* up to 18

Consistent over . . . . The entry in the field ***Consistent over*** specifies whether the data is to be transmitted as individual units (bytes, words, etc.) or as complete package (1...32 bytes) during a PLC-cycle. This function is only to be set to 'whole length' if actually required, because the transmission as 'Unit' is faster.

> **Note:**
> If the data is to be transmitted consistently for the entire length, you have to specify this here *and* you have to use SFC14 and SFC15 (refer to Step7-PLC Manual).

Manufacturer Specific Data . . . . . In the field ***Data for Specific Manufacturer*** can be specified, which of the maximum four Rx- and TxPDOs per CANopen-node are used (see table 16/page 36) and the length- and format-parameters of the PDOs are specified, each divided by commas.
The parameters have to be entered as **hexadecimal values** !

Please note that the position of the bytes in the field ***Manufacturer Specific Data*** depends on the setting of the parameter *SlavePDO-Orientation*.

Note:
If the parameter *Slave PDO-Orientation* is set to 'yes' in the parameter telegram, CAN frames are transmitted at the outputs with standard TxPDO-identifiers and CAN frames are received at the inputs with standard RxPDO-identifiers.
If the parameter *Slave PDO-Orientation* is 'no' (default), the CAN frames are transmitted via RxPDO-identifiers and they are received via TxPDO-identifiers.

Note:
Optional all Tx-PDOs can be cyclically transmitted. The cyclic transmission can be set via the parameter *TxPDO cycle Time*.

***SlavePDO-Orientation* = 'no'**
The position of the bytes in the field ***Manufacturer Specific Data*** is defined as follows:

1. Byte	2. Byte	...
Format-Byte TxPDO1	*Format-Byte RxPDO1*	...

...	3. Byte	4. Byte	5. Byte	6. Byte	...
...	*Length TxPDO1*	*Length RxPDO1*	*Format-Byte TxPDO2*	*Format-Byte RxPDO2*	...

...	7. Byte	8. Byte	9. Byte	10. Byte	...
...	*Length TxPDO2*	*Length RxPDO2*	*Format-Byte TxPDO3*	*Format-Byte RxPDO3*	...

...	11. Byte	12. Byte	13. Byte	14. Byte
...	*Length TxPDO3*	*Length RxPDO3*	*Format-Byte TxPDO4*	*Format-Byte RxPDO4*

***SlavePDO-Orientation* = 'yes'**
The position of the bytes in the field ***Data for Specific Manufacturer*** is defined as follows:

1. Byte	2. Byte	...
Format-Byte RxPDO1	*Format-Byte TxPDO1*	...

...	3. Byte	4. Byte	5. Byte	6. Byte	...
...	*Length RxPDO1*	*Length TxPDO1*	*Format-Byte RxPDO2*	*Format-Byte TxPDO2*	...

...	7. Byte	8. Byte	9. Byte	10. Byte	...
...	*Length RxPDO2*	*Length TxPDO2*	*Format-Byte RxPDO3*	*Format-Byte TxPDO3*	...

...	11. Byte	12. Byte	13. Byte	14. Byte
...	*Length RxPDO3*	*Length TxPDO3*	*Format-Byte RxPDO4*	*Format-Byte TxPDO4*

General Rules for entries in the field *Data of Specific Manufacturer* (for examples, see page 42, 43):

1. For every PDO that should be used, the data length of a value higher than '0' has to be entered.

2. If only the leading PDOs are required, the entries for the following PDOs are not necessary (e.g. enter only RxPDO1, TxPDO1, RxPDO2 and Tx-PDO2).

3. If only the first PDOs, RxPDO1 and TxPDO1, are specified, the entry of the length is not applicable, i.e. only the format bytes in the first and second byte in the field *Data of Specific Manufacturer* are registered.

4. If there are PDOs missing within the parameter list, the bytes of this PDOs have nevertheless to be entered (with 00_h), to allow for the correct byte assignment for the following PDOs.

5. For the last RxPDO and the last TxPDO of a node <u>no</u> length entry has to be entered.
   (The firmware calculates the remaining bytes in adding up the registered byte-lengths and subtracts this from the byte number calculated from the entries in the fields *Length* and *Unit*. The result is assigned to the last RxPDO and TxPDO respectively.)

Format Byte RxPDOx/TxPDOx:
The *Format Byte RxPDOx/TxPDOx* is used to convert the user data from Motorola format (high byte first = big endian) into Intel format (low byte first = little endian).

Attention!
The *Format byte RxPDOx/TxPDOx* always has to be entered in hexadecimal form!

Background:
Messages which are longer than 1 byte are normally transmitted via a CANopen network in Intel format, while the Siemens PLC operates in Motorola format.

Starting with bit 7 of the format byte you can decide whether the following byte is to be converted as well, i.e. swapped, or not. If a '1' is specified for a byte, the following bytes are converted until the next '0' transmitted. The functionality can be explained best by means of an example.

Example:
A CAN telegram has got a date in Intel format in the first word, followed by 2 bytes which are not to be swapped and a long word in the last 4 bytes which is in Intel format again.
Binary the following representation results for the format byte:

Bit No.	7	6	5	4	3	2	1	0
Bit of *format-byte*	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**
hex	8				E			
action	begin swap	end swap	unchanged	unchanged	begin swap	swap	swap	end swap

Data-bytes	1	2	3	4	5	6	7	8
CAN-frame	2 byte Intel format		byte 3	byte 4	4 byte Intel format			
PLC-data	2 byte Motorola format		byte 3	byte 4	4 byte Motorola format			

From this the format byte results in $8E_h$. If all eight bytes are to be swapped, for instance, value FE_h has to be specified for the format byte.
The lowest bit is generally without significance, because the telegram and therefore the formatting has been completed. The bit should always be set to 0.

Length RxPDOx/TxPDOx:
Here the length of the PDOs has to be entered.
0... data are not evaluated or transmitted
1...9 1...8 byte data are evaluated and transmitted
The configured length has to be increased by one byte, if *RX-Counter=yes* or *Tx-Counter=yes*.

Note:
The sum of all data bytes of all PDOs of one data direction always has to be the same (or smaller), than the number of data-bytes of the DP-slaves, defined in the fields *Length* and *Unit* (RxPDO: Inputs; TxPDO: Outputs).

Note:
If the parameter *Slave PDO-Orientation* is set to 'yes' in the parameter telegram, CAN frames are transmitted at the outputs with standard TxPDO-identifiers and CAN frames are received at the inputs with standard RxPDO-identifiers.
If the parameter *Slave PDO-Orientation* is 'no' (default), the CAN frames are transmitted via RxPDO-identifiers and they are received via TxPDO-identifiers.

Examples for the Entries in *Manufacturer-specific Data* for Slave-PDO Orientation = yes:

Entry in the field *Data of Specific Manufacturer* [hex]	Reasonable entries for PLC-slot length		Meaning
	input	output	
00,00	1...8	1...8	RxPDO1 und TxPDO1 with 1...8 byte each, no byte swapping
8E,00	1...8	1...8	RxPDO1 with 1...8 byte data and byte swapping as described in the example on page 41 and TxPDO1 with 1...8 byte data without byte swapping
00,00,01,02,00,00	2...9	3...10	no byte swapping, RxPDO1 with 1 byte length, TxPDO1 with 2 byte length, RxPDO2 with 1...8 byte length, TxPDO2 with 1...8 byte length
00,00,08,08,00,00,08, 08,00,00,08,08,00,00	32	32	no byte swapping, all PDOs with 8 byte data length

Examples for the Entries in *Manufacturer-specific Data* for Slave-PDO Orientation = yes, with Rx-Counter=yes or Tx-Counter=yes:

Entry in the field *Data of Specific Manufacturer* [hex]	Reasonable entries for PLC-slot length		Meaning
	input	output	
00,00	2...9	2...9	RxPDO1 und TxPDO1 with 1...8 byte each +1 byte Rx- or Tx-counter, no byte swapping
8E,00	2...9	2...9	RxPDO1 with 1...8 byte data +1 byte Rx-counter and byte swapping as described in the example on page 41 and TxPDO1 with 1...8 byte data +1 byte Tx-counter without byte swapping
00,00,02,03,00,00	4...11	5...12	no byte swapping, RxPDO1 with 1 byte length +1 byte Rx-counter, TxPDO1 with 2 byte length +1 byte Tx-counter, RxPDO2 with 1...8 byte length +1 byte Rx-counter, TxPDO2 with 1...8 byte length +1 byte Tx-counter
00,00,09,09,00,00,09, 09,00,00,09,09,00,00	36	36	no byte swapping, all PDOs with 8 byte data length +1 byte Rx- or Tx-counter

Examples for the Entries in *Manufacturer-specific Data* for Slave-PDO Orientation = no:

Entry in the field *Data of Specific Manufacturer* [hex]	Reasonable entries for PLC-slot length		Meaning
	input	output	
00,00	1...8	1...8	TxPDO1 und RxPDO1 with 1...8 byte each, no byte swapping
00, 8E	1...8	1...8	RxPDO1 with 1...8 byte data and byte swapping as described in the example on page 41 and TxPDO1 with 1...8 byte data without byte swapping
00,00,02,01,00,00	3...10	2...9	no byte swapping, TxPDO1 with 2 byte length, RxPDO1 with 1 byte length, TxPDO2 with 1...8 byte length, RxPDO2 with 1...8 byte length
00,00,08,08,00,00,08, 08,00,00,08,08,00,00	32	32	no byte swapping, all PDOs with 8 byte data length

Examples for the Entries in *Manufacturer-specific Data* for Slave-PDO Orientation = no, with Rx-Counter=yes or Tx-Counter=yes:

Entry in the field *Data of Specific Manufacturer* [hex]	Reasonable entries for PLC-slot length		Meaning
	input	output	
00,00	2...9	2...9	TxPDO1 und RxPDO1 with 1...8 byte each +1 byte Rx- or Tx-counter, no byte swapping
00, 8E	2...9	2...9	RxPDO1 with 1...8 byte data +1 byte Rx-counter and byte swapping as described in the example on page 41 and TxPDO1 with 1...8 byte data +1 byte Tx-counter without byte swapping
00,00,03,02,00,00	5...12	4...11	no byte swapping, TxPDO1 with 2 byte length +1 byte Tx-counter , RxPDO1 with 1 byte length +1 byte Rx-counter, TxPDO2 with 1...8 byte length +1 byte Tx-counter, RxPDO2 with 1...8 byte length +1 byte Rx-counter
00,00,09,09,00,00,09, 09,00,00,09,09,00,00	36	36	no byte swapping, all PDOs with 8 byte data length +1 byte Rx- or Tx-counter

6. Acyclic PROFIBUS Data Transfer to CANopen

6.1 Mode of Operation

The CANopen-DP/2 features the acyclic PROFIBUS data transfer for the transmission of parameters between PROFIBUS and CANopen (PROFIBUS-protocol extension DP-V1).
The principle of the data transmission for reading and writing of the parameter is identical for both directions:

- First a **DS-Write request** is transmitted, in which the direction of the transmission is defined (*Request Parameter* for reading of parameters and *Change Parameter* for writing of parameters respectively). Here you can specify if a SDO-read request or a SDO-write request is generated from the CANopen-DP/2.

- In the following **DS-Read request** the result of the operation is transmitted. For a *Request Parameter* (request ID = 01_h) a SDO-read request is generated, for a *Change Parameter* (request ID = 02_h) a SDO-write request is generated. As base of the parameter transmission the PROFIBUS profile *ProfiDrive* [2] is used. The complete procedure is already defined there.

The following figure shows the conversion of the PROFIBUS-access in a SDO-transfer:

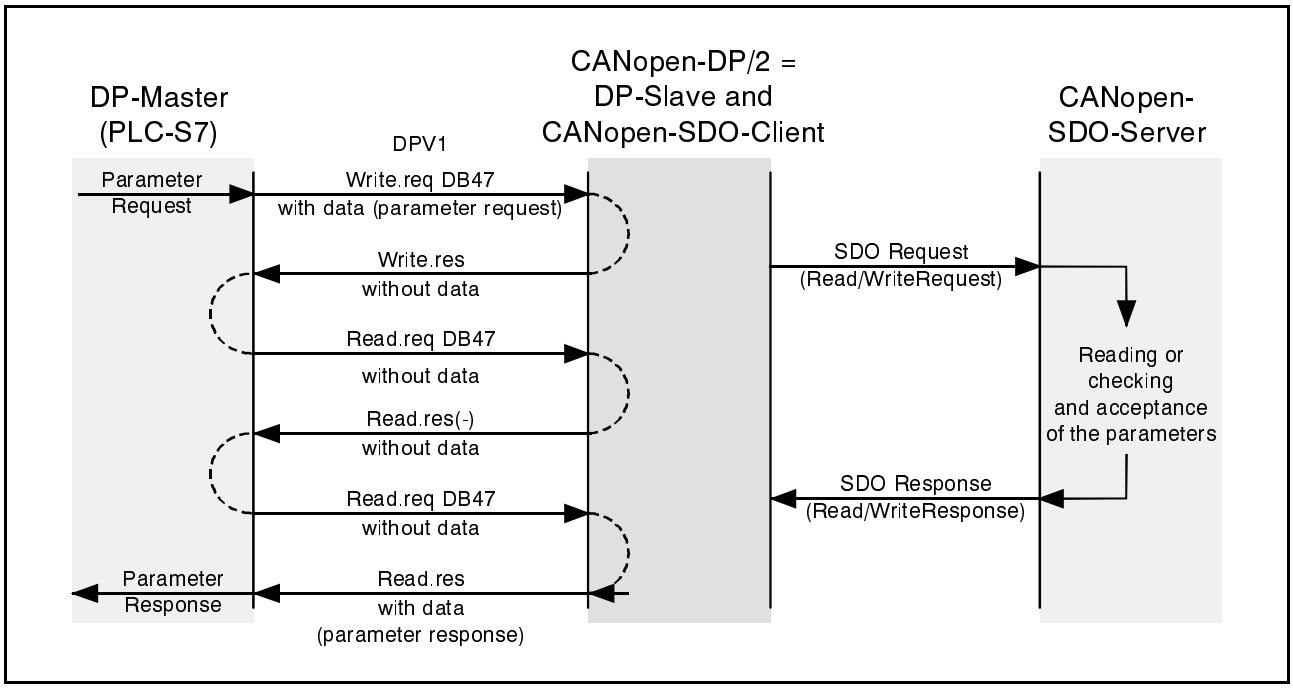

Fig. 12: Principle of the single acyclic parameter transmission

In a DS-Write-request also several objects can be written or requested.

The following figure shows the PROFIBUS access in n SDO-transfers:

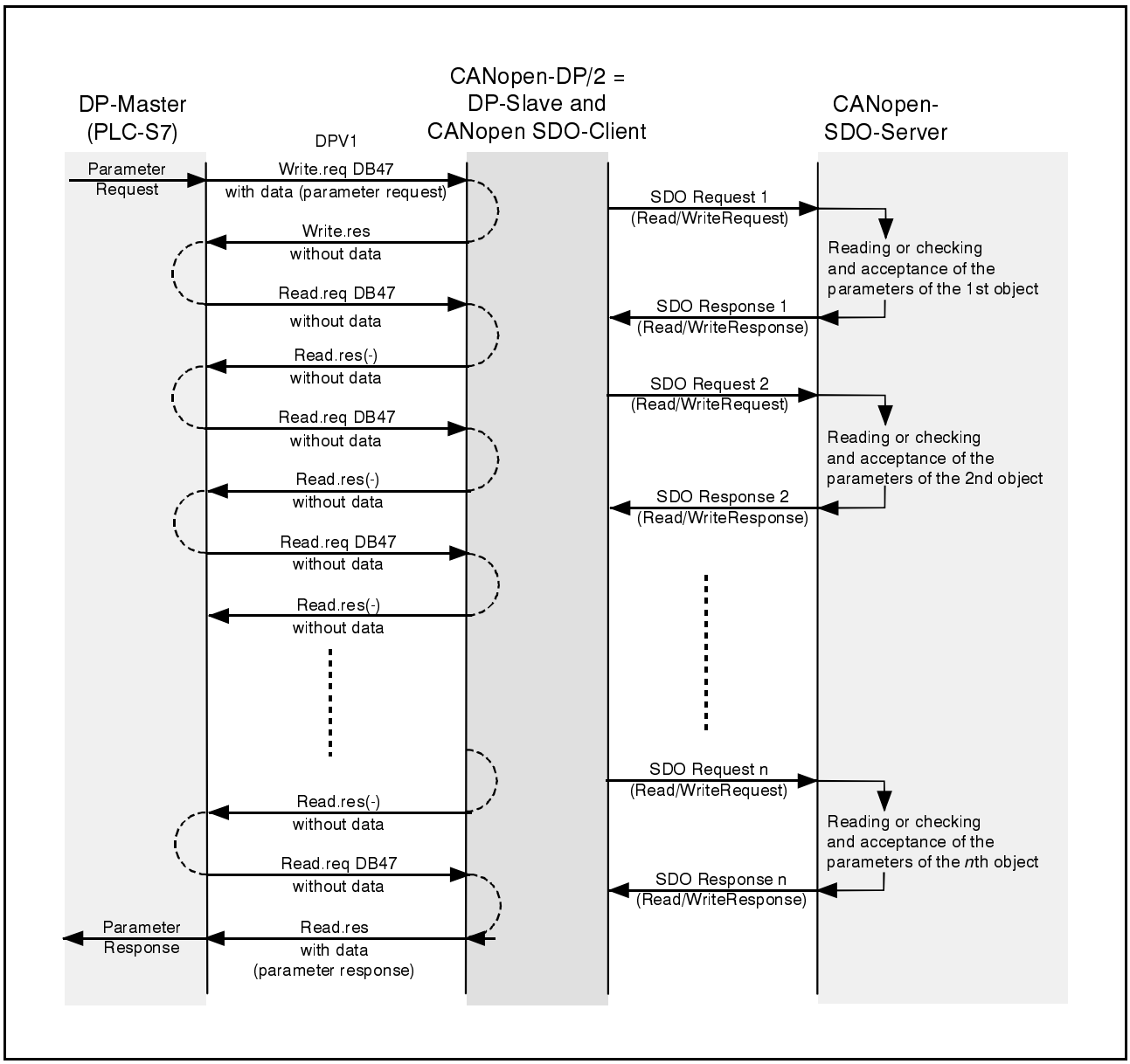

Fig. 13: Principle of the multiple acyclic parameter transmission

6.2 Structure of the PROFIBUS Write Request

For the write request a PROFIBUS telegram with variable length and two address extensions is used. The data field of the telegram has the following structure:

Segment	Octet	Name	Description
-	1.	*Function_Num*	Bit 7: 0: Request/Response OK 1: Response Error Frame Bit 6...0: *Function_Num* ($5F_h$ for DS_Write)
	2.	*Slot_Number*	0 (not evaluated)
	3.	*Index*	always 47 (see [2])
	4.	*Length*	length of user data (from octet 5 up to the end)
Request Header	5.	*Request Reference, Handshake Counter*	consecutive number, increased with every request
	6.	*Request ID*	01_h: Request Parameter 02_h: Change Parameter
	7.	*Node-ID*	CANopen Node-ID
	8.	*No. of Parameters*	number (n) of the SDOs in this request (here: n = 1)
Parameter Address	9.	*Attribute*	always 10_h (see [2])
	10.	*No. of elements*	number of the elements, always 1
	11.	*Index, High-Byte*	parameter number, higher-order byte (is converted to CANopen index-high byte)
	12.	*Index, Low-Byte*	parameter number, lower-order byte (is converted in CANopen index-low byte)
	13.	*Subindex, High-Byte*	subindex , higher-order byte (not used in CANopen)
	14.	*Subindex , Low-Byte*	subindex, lower-order byte (is converted to CANopen subindex)
Parameter Values (only if Request-ID = 02_h)	15.	*Format*	data format (only existing, if request ID = 02_h change parameter), data format see page 51
	16.	*No. of values*	number of data bytes of this parameter (only existing, if request-ID = 02_h change parameter)
	17.	*Values*	values (only existing, if request-ID = 02_h change parameter)
	18.	*Values*	

Table 17: Content of the data field *PDU* at “Write-Request with Data” (parameter-request)

Octet	Name	Description
1.	*Function_Num*	Bit 7: 0: request/response OK 1: response error frame Bit 6...0: *Function_Num* ($5F_h$ for DS_Write)
2.	*Slot_Number*	0 (not evaluated)
3.	*Index*	47
4.	*Length*	length of the user data, mirrored

Table 18: Content of the data field *PDU* at “Write-Response without Data”

6.2.1 Example : Structure of the PROFIBUS Write Request for n = 5 SDOs

Segment	SDO	Octet	Name	Description
-	-	1.	*Function_Num*	Bit 7: 0: Request/Response OK 1: Response Error Frame Bit 6...0: *Function_Num* ($5F_h$ for DS_Write)
		2.	*Slot_Number*	0 (not evaluated)
		3.	*Index*	always 47 (see [2])
		4.	*Length*	length of the user data (from octet 5 up to the end, here 56)
Request Header	-	5.	*Request Reference, Handshake Counter*	consecutive number, increased with every request
		6.	*Request ID*	01_h: Request parameter 02_h: Change parameter
		7.	*Node-ID*	CANopen Node-ID
		8.	*No. of Parameters*	number (n = 5) of the SDOs in this request
Parameter Address for the SDOs 1 ... 5 (n)	1.	9.	*Attribute*	Parameter Address for the 1st SDO
		10.	*No. of elements*	
		11.	*Index, High-Byte*	
		12.	*Index, Low-Byte*	
		13.	*Subindex, High-Byte*	
		14.	*Subindex , Low-Byte*	
	2.	15.	*Attribute*	Parameter Address for the 2nd SDO
		16.	*No. of elements*	
		17.	*Index, High-Byte*	
		18.	*Index, Low-Byte*	
		19.	*Subindex, High-Byte*	
		20.	*Subindex , Low-Byte*	
	:	:	:	:
	5. (n)	33.	*Attribute*	Parameter Address for the 5th (*n*-th.) SDO
		34.	*No. of elements*	
		35.	*Index, High-Byte*	
		36.	*Index, Low-Byte*	
		37.	*Subindex, High-Byte*	
		38.	*Subindex , Low-Byte*	
Parameter Values for SDOs 1 ... 5 (n) (only if Request-ID = 02_h)	1.	39	*Format (Byte)*	Parameter Values for the 1st SDO (e.g..: Transmission of a byte)
		40	*No. of values (1)*	
		41	*Values (Byte 1)*	
		42	*Values (0)*	
	2.	43	*Format (Word)*	Parameter Values for the 2 nd SDO (e.g..: Transmission of a word)
		44	*No. of values (2)*	
		45	*Values (Byte 1)*	
		46	*Values (Byte 2)*	
	:	:	:	:
	5. (n.)	55	*Format (Double Word)*	Parameter Values for the 5th (*n*-th.) SDO (e.g.: Transmission of a double word)
		56	*No. of values (4)*	
		57	*Values (Byte 1)*	
		58	*Values (Byte 2)*	
		59	*Values (Byte 3)*	
		60	*Values (Byte 4)*	

Table 19: Example for the content of the data field *PDU* at “Write-Response without Data” (n =5)

The number of the octets for the parameter address and the parameter values depends on the number (n) of the SDOs in the request.
The content of the telegrams (values) is always transmitted as word or double word, with the Most Significant Byte first (Big Endian).
If the value consists of an odd number of bytes, always a zero byte is appended.

6.3 Structure of the PROFIBUS-Read-Request

For the read request a PROFIBUS telegram with variable length and two address extensions is used. The data field of the telegram is composed of:

Octet	Name	Description
1.	*Function_Num*	Bit 7: 0 Bit 6...0: *Function_Num* ($5E_h$ for DS_Read)
2.	*Slot_Number*	0 (not evaluated)
3.	*Index*	47
4.	*Length*	length of the user data, (maximum)

Table 20: Content of the data field *PDU* at "Read-Request without Data"

Segment	Octet	Name	Meaning
-	1.	*Function_Num*	Bit 7: 0: Request/Response OK 1: Response Error Frame Bit 6...0: *Function_Num* ($5E_h$ for DS_Read)
	2.	*Slot_Number*	0 (not evaluated)
	3.	*Index*	always 47 (see [2])
	4.	*Length*	length of the user data (from octet 5 up to the end)
Response Header	5.	*Request Reference, Handshake Counter*	consecutive number, increased with every request
	6.	*Request ID*	01_h: Request parameter 02_h: Change parameter
	7.	*Node-ID*	CANopen Node-ID
	8.	*No. of Parameters*	Number (n) of the SDOs in this request (here n = 1)
Parameter Address	9.	*Attribute*	always 10_h (see [2])
	10.	*No. of elements*	number of the elements, always 1
	11.	*Index, High-Byte*	parameter number, higher-order byte (is converted to CANopen index-high byte)
	12.	*Index, Low-Byte*	parameter number, lower-order byte (is converted to CANopen index-low-byte)
	13.	*Subindex, High-Byte*	subindex , higher-order byte (is not used in CANopen)
	14.	*Subindex , Low-Byte*	subindex, lower-order byte (is converted to CANopen subindex)
Parameter Values (only if Request-ID = 01_h)	15.	*Format*	data format (only existing, if request-ID = 01_h request parameter), data format see page 51
	16.	*No. of values*	number of data bytes of this parameter (only existing, if request-ID = 01_h request parameter)
	17.	*Values*	values (only existing, if request-ID = 01_h request parameter)
	18.	*Values*	

Table 21: Content of the data field *PDU* at "Read-Response" (parameter-response)

6.3.1 Example: Structure of the PROFIBUS-Read-Requests for n = 5 SDOs

The number of the octets of the parameter address and the parameter values depend on the number (n) of the SDOs in the request.

Segment	SDO	Octet	Name	Description
-	-	1.	*Function_Num*	Bit 7: 0: Request/Response OK 1: Response Error Frame Bit 6...0: *Function_Num* ($5E_h$ for DS_Read)
		2.	*Slot_Number*	0 (not evaluated)
		3.	*Index*	always 47 (see [2])
		4.	*Length*	length of the user data (from octet 5 up to the end, here:56)
Request Header	-	5.	*Request Reference, Handshake Counter*	consecutive number, increased with every request
		6.	*Request ID*	01_h: Request parameter 02_h: Change parameter
		7.	*Node-ID*	CANopen Node-ID
		8.	*No. of Parameters*	number (n = 5) of the SDOs in this request
Parameter Address for SDOs 1 ... 5 (n)	1.	9.	*Attribute*	Parameter Address for the 1st SDO
		10.	*No. of elements*	
		11.	*Index, High-Byte*	
		12.	*Index, Low-Byte*	
		13.	*Subindex, High-Byte*	
		14.	*Subindex , Low-Byte*	
	2.	15.	*Attribute*	Parameter Address for the 2nd SDO
		16.	*No. of elements*	
		17.	*Index, High-Byte*	
		18.	*Index, Low-Byte*	
		19.	*Subindex, High-Byte*	
		20.	*Subindex , Low-Byte*	
	:	:	:	:
	5. (n)	33.	*Attribute*	Parameter Address for the 5th (*n*-th) SDO
		34.	*No. of elements*	
		35.	*Index, High-Byte*	
		36.	*Index, Low-Byte*	
		37.	*Subindex, High-Byte*	
		38.	*Subindex , Low-Byte*	
Parameter Values for SDOs 1 ... 5 (n) (only if Request-ID = 01_h)	1.	39	*Format (Byte)*	Parameter Values for the 1st SDO (e.g..: Transmission of a byte)
		40	*No. of values (1)*	
		41	*Values (Byte 1)*	
		42	*Values (0)*	
	2.	43	*Format (Word)*	Parameter Values for the 2nd SDO (e.g..: Transmission of a word)
		44	*No. of values (2)*	
		45	*Values (Byte 1)*	
		46	*Values (Byte 2)*	
	:	:	:	:
	5. (n.)	55	*Format (Double Word)*	Parameter Values for the 5th (*n*-th) SDO (e.g..: Transmission of a double word)
		56	*No. of values (4)*	
		57	*Values (Byte 1)*	
		58	*Values (Byte 2)*	
		59	*Values (Byte 3)*	
		60	*Values (Byte 4)*	

Table 22: Example of the data field *PDU* at “Read-Request with Data” for n =5

6.4 Data Format

The data format has to be stated explicitly for every access. Depending on the format the conversion PLC-format <-> CANopen-format has to be carried out according to the rules listed below (the table below can e.g. be applied for SIEMENS S7, because the conversion of the data from little-endian to big-endian format is necessary).

Format [dec]	Format [hex]	Data Type PROFIBUS	Data Type CANopen	Conversion PLC-Format <-> CANopen-Format
1	01	Boolean	Boolean	no
2	02	Integer8	INTEGER8	no
3	03	Integer16	INTEGER16	yes (2 byte)
4	04	Integer32	INTEGER32	yes (4 byte)
5	05	Unsigned8	UNSIGNED8	no
6	06	Unsigned16	UNSIGNED16	yes (2 byte)
7	07	Unsigned32	UNSIGNED32	yes (4 byte)
8	08	FloatingPoint	REAL32	yes (4 byte)
9	09	VisibleString	VisibleString	no
10	0A	OctetString	OctetString	no
...	...	...	...	...
33	21	N2 Normalized value (16 bit)	INTEGER16	yes (2 byte)
34	22	N4 Normalized value (32 bit)	INTEGER32	yes (4 byte)
...	...	...	...	...
65	41	Byte	UNSIGNED8	no
66	42	Word	UNSIGNED16	yes (2 byte)
67	43	Double Word	UNSIGNED32	yes (4 byte)
68	44	Error	Error	no

Table 23: Data format

The table below shows the order of the data bytes, for the PLC using the Motorola-format (big-endian, high-byte first) and CANopen the Intel-format (little endian, low-byte first):

Data length [bytes]	Order PLC	Order CANopen	Example PLC	Example CANopen
1	byte 1	byte 1	12_h	12_h
2	byte 1 byte 2	byte 2 byte 1	$12\ 34_h$	$34\ 12_h$
4	byte 1 byte 2 byte 3 byte 4	byte 4 byte 3 byte 2 byte 1	$12\ 34\ 56\ 78_h$	$78\ 56\ 34\ 12_h$

Table 24: Comparison of the byte orientation little/big-endian

6.5 Error Codes of the PROFIBUS at Acyclic Transfers to/from CANopen

Octet No.	Name	Meaning PLC	Meaning CANopen (SDO-abort code)
1	Function_Num	bit 7: 1: response error frame bit 6...0: Function_Num ($5E_h$ for DS_Read, $5F_h$ for DS_Write)	-
2	Error_Decode	always 128 (DP-V1)	-
3	Error Code 1	bit 7... 4: error class bits bit 3... 0: error code bits	-
		error class = 0...9: reserved	-
		error class = 10: application error code = 0: read error error code = 1: write error error code = 8: version conflict	 not used not used not used
		error class=11: access error code = 0: invalid index error code = 2: invalid slot (Node-ID) error code = 3: type conflict error code = 4: invalid area error code = 9: invalid type error code = 12: timeout error error code = 13: service unsupported error code = 15: generic error	 INDEX != 47 slot (Node-ID) > 127 not used not used not used 0504 0000
		error class = 12: resource error code = 0: read constr. conflict error code = 1: write constr. conflict error code = 2: resource busy error code = 3: resource unavailable	 not used not used not used no CAN-Ack or read-request without preceding write-request
4	Error Code 2	always 0	always 0

Table 25: Error codes at acyclic transfers

If communication to a CANopen-node is not possible (node is nonexistent or the communication is disturbed), the CANopen-DP/2 Gateway transmits a SDO-abort according to CANopen-standard and reports the corresponding error (error class 11/error code 12).

All SDO-abort codes, which are not listed here and in table 26, cause a generic error (error class 11/error code 15).

If the *Node-ID* is 255 (see page 46), a MPDO (DAM-PDO, structure see [1]) is generated as broadcast. Because this service is unconfirmed, on PROFIBUS-side it is answered with response OK, or with error code error class 12/error code 3 (resource unavailable) in case of a CAN-Ack missing.

6.5.1 Error-Numbers in DPV1 Parameter Value

In case of an error (negative partial response) the parameter value has the following content:

Format = error
No. of values = 1
Value = 1. byte: error value = Error No. (error number, according to table 26)
2. byte: Number of the parameter, beginning with 0

Error No. (error number)	Error Name	Meaning CANopen (SDO-Abort-Code)
0x00	Impermissible index	6020000
0x01	Parameter value cannot be changed	06010000/1/2
0x02	Low or high limit exceeded	06090030/1/2
0x03	Faulty subindex, subindex can not be accessed	6090011
0x05	Incorrect data type	06070010/2/3
0x11	Request cannot be executed because of operating state	8000021
0x16	Parameter address impermissible	6040043
0x65	Other errors	all other abort-codes except 05040000

Table 26: Error numbers in DPV1 parameter responses

6.6 I&M0 for Identification

6.6.1 Write Request

For the write request a PROFIBUS telegram with variable length is used. The data field of the telegram has the following structure:

Segment	Octet	Name	Description
-	1.	*Function_Num*	$5F_h$ (DS_Write)
	2.	*Slot_Number*	0
	3.	*Index*	255 (I&M)
	4.	*Length*	Length of user data = 4 (from octet 5 up to 8)
I&M0 Data	5.	*Extended FN*	08_h (Call Request)
	6.	*reserved*	00_h
	7.	*FI_Index (high)*	FD_h (Index 65000)
	8.	*FI_Index (low)*	$E8_h$ (Index 65000)

Table 27: Content of the data field *PDU* at “Write-Request with Data” (I&M0-Request)

Octet	Name	Description
1.	*Function_Num*	$5F_h$ (DS_Write)
2.	*Slot_Number*	0
3.	*Index*	255 (I&M)
4.	*Length*	length of the user data, mirrored

Table 28: Content of the data field *PDU* at “Write-Response without Data” (I&M0)

6.6.2 Read Request

For the read request a PROFIBUS telegram with variable length is used. The data field of the telegram has the following structure:

Octet	Name	Description
1.	*Function_Num*	$5E_h$ (DS_Read)
2.	*Slot_Number*	0
3.	*Index*	255 (I&M)
4.	*Length*	Length of the user data, (maximum)

Table 29: Content of the data field *PDU* at “Read-Request without Data” (I&M0)

Segment	Octet	Name	Description
-	1.	*Function_Num*	$5E_h$ (DS_Read)
	2.	*Slot_Number*	0 (not evaluated)
	3.	*Index*	255 (I&M)
	4.	*Length*	Length of the user data = 64 (from octet 5 up to 68)
I&M0 Data	5.-14.	*Manufacturer-specific*	“CANopen-DP/2”
	15.-16.	*Manuf. ID*	$098E_h$
	17.-36.	*OrderID*	“C.2909.02"
	37.-52.	*Ser.Num*	e.g. “GF003219"
	53.-54.	*HW Rev.*	currently 0003_h
	55.-58.	*SW.Rev*	currently “V” 010000_h
	59.-60.	*Revision_Counter*	0000_h (not supported)
	61.-62.	*ProfileID*	0000_h (does not correspond to a profile)
	63.-64.	*Profile_Specific_type*	0000_h (does not correspond to a profile)
	65.-66.	*IM_Version*	0100_h (version 1.0)
	67.-68.	*IM_Supported*	0000_h (only supported by I&M0)

Table 30: Content of the data field *PDU* at “Read-Response” (I&M0-Response)

7. CANopen Introduction

Apart from basic descriptions of the CANopen, this chapter contains the most significant information about the implemented functions.
A complete CANopen description is too extensive for the purpose of this manual.
Further information can therefore be taken from the CANopen standard ‘CiA 301’ [1].

7.1 Definition of Terms

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

PDOs (Process Data Objects)
PDOs are used to transmit up to 8 byte process data.
In the ‘Receive’-PDO (RxPDO) process data is received.
In the ‘Transmit’-PDO (TxPDO) process data is transmitted.

SDOs (Service Data Objects)
SDOs are used to transmit module internal configuration- and parameter data. The object dictionary can be accessed via SDOs.
In opposition to the PDOs SDO-messages are confirmed. A write or read request on a data object is always answered by a response telegram with an error index.

NMT State Machine
For the control of the device functions all CANopen-devices feature an internal state machine. In the individual states only particular functions are permitted. The state transitions can be triggered by internal events (e.g. boot-up, error, reset) or by the NMT-master.

7.2 NMT Boot-up

After switching on every CANopen device obtains a phase of initialisation. The device automatically enters the Pre-Operational state directly after finishing the device initialisation.

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

7.3 CANopen Object Directory

The object directory is basically a (sorted) group of objects which can be accessed via the network. Each object in this directory is addressed with a 16-bit index. The index in the object directories is represented in hexadecimal format.

The index can be a 16-bit parameter in accordance with the CANopen specification or a manufacturer-specific code. By means of the MSBs of the index the object class of the parameter is defined.

Part of the object directory are among others:

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

Type and number of the supported objects depend on the corresponding CANopen module type.

7.4 Access to the Object Directory via SDOs

SDOs (Service Data Objects) are used to get access to the object directory of a device. They are used for initialisation of a device and for transmission of the parameters.
SDO-accesses are only possible if the CANopen module is in the state *operational* or *pre-operational*

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

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

Description of the SDOs:

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

Command code . . The command code transmitted consists amongst others of the command specifier and the length. Frequently required combinations are, for instance:

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

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

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

Frequently Used Command Codes

The following table summarizes frequently used command codes. The command frames must always contain eight data bytes. Notes on the syntax and further command codes can be found in CiA 301 [1], chapter “Service Data Object”.

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

Index, Sub-Index . Index and sub-index address the parameter in the object dictionary.

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

Error Codes of the SDO Domain Transfer

The following error codes might occur (according to CiA 301[1], chapter “Abort SDO Transfer Protocol”):

Error code [Hex]	Name	Description
0x05040001	SDO_CS_UNKNOWN	wrong command specifier
0x06010000	SDO_WRONG_ACCESS	wrong write access
0x06010001	SDO_WRITE_ONLY	attempt to read a write only object
0x06010002	SDO_READ_ONLY	attempt to write a read only object
0x06020000	SDO_WRONG_INDEX	wrong index
0x06040043	SDO_PARA_INCOMPATIBLE	general parameter incompatibility reason
0x06070010	SDO_WRONG_LENGTH	wrong number of data bytes
0x06070012	SDO_PARA_TO_LONG	length of service parameter too high
0x06070013	SDO_PARA_TO_SHORT	length of service parameter too low
0x06090011	SDO_WRONG_SUBIND	wrong sub-index
0x06090030	SDO_VALUE_EXCEEDED	transmitted parameter is outside the accepted value range
0x06090031	SDO_VALUE_TOO_HIGH	value of parameter written too high
0x06090032	SDO_VALUE_TOO_LOW	value of parameter written too low
0x08000000	SDO_OTHER_ERROR	general error
0x08000021	SDO_LOCAL_CONTROL	data cannot be transferred or stored to the application because of local control

7.5 Access to Process Data via PDOs

During operation, the process data of a CANopen device are exchanged via PDOs.

There are four standard-PDOs per data direction available. The communication parameters of these RxPDOs and TxPDOs can be programmed via SDOs. The programmable communication parameters can be e.g. the COB-ID and the transmission type (e.g. sync/async).
Also via the communication parameters the PDOs are assigned with the objects of the object dictionary, which contain the relevant process data. The CANopen protocol CiA 301 [1] defines default assignments of the PDOs with objects for some typical applications.

How many and which PDOs are supported by a CANopen device and which communication parameters can be changed depends on the individual CANopen-firmware of the devices.

7.6 Overview of Used CANopen Identifiers

The following table shows a short list of the significant general CANopen telegrams.

CAN-identifier [hex]	Name	Length	Data [hex]	Description
0	NMT	2	01 xx	Start (Pre-operational -> Operational)
0	NMT	2	80 xx	Operational -> Pre-operational
0	NMT	2	81 xx	Reset
0	NMT	2	82 xx	Reset communication
80_h	SYNC	0	-	SYNC to all
80_h + *Node-ID*	EMCY	0...8 bytes	error code	Emergency message

Node-ID... Node-ID of the responded CANopen module
xx... Node-ID of a CANopen-module or ‘00’ for message to all CANopen-participants

8. CANopen Object Dictionary of the CANopen-DP/2-Gateway

For a detailed description of the objects refer to CiA 301 [1].

Index [hex]	**Sub index** [dec]	**Name**	**Data type**	**R/W**	**Default value**
1000	-	Device Type	unsigned 32	ro	$0000\ 0000_h$
1001	-	Error Register	unsigned 8	ro	00_h
1005	-	COB-ID-Sync	unsigned32	rw	80_h
1008	-	Manufacturer Device Name	visible string	ro	“CANopen-DP/2”
1009	-	Manufacturer Hardware Version	visible string	ro	x.yy (depending on version)
100A	-	Manufacturer Software Version	visible string	ro	x.yy (depending on version)
100C	-	Guard Time	unsigned 16	rw	0000_h
100D	-	Life Time Factor	unsigned 8	rw	00_h
1010	-	Store Parameter	unsigned 32	rw	no function
1011	-	Restore Parameter	unsigned 32	rw	no function
1016	126	Consumer Heartbeat Time	array	rw	00_h
1017	-	Producer Heartbeat Time	unsigned 16	rw	00_h
1018	0	Identity Object	unsigned 8	ro	number of entries=4
	1		unsigned 32	ro	vendor ID=00000017_h
	2		unsigned 32	ro	prod. code=22909002_h => C.2909.002
	3		unsigned 32	ro	software revision number=40001_h
	4		unsigned 32	ro	serial number

Table 31: Implemented CANopen objects of the CANopen-DP/2-Gateway

9. CAN-Layer-2 Functions

9.1 Introduction

Besides the CANopen implementing the CANopen-DP/2 device supports the CAN-Layer-2 functions.

Note:
If only CAN-Layer-2 functions are required, the CANopen-DP/2 should not be used, but the unit ”CAN-DP/2”, which is optimized for CAN-Layer-2 applications.

Below the significant differences are listed, which result from the CAN-Layer-2-implementation of the CANopen-DP/2 in reference to the functions described so far:

External (module specific) diagnostic bytes:
- byte 11 / error 7 and 8 are not defined

Identifier related diagnostic bytes:
- are inapplicable for CAN-Layer-2

Procedure of configuration:
- the device related parameter for CANopen (*Slave PDO orientation*, *SDO Timeout*, *Heartbeat* und *MPDO Identifier*, see page 26), that can be set via the parameter telegram (in the “Properties DP-Slave”window), do not affect the CAN-layer-2 functions.

- all CAN identifiers in the 11-bit and in the 29-bit CAN identifier range can be addressed.

Restrictions for the CANopen implementation:
- For the CAN-layer-2 data transfer with CAN-identifiers simulated slots of the PROFIBUS slaves are assigned, as for the cyclic data transfer via CANopen-PDOs. A double assignment is not possible, i.e. a slot can only be assigned once. This must be considered at the selection of the CANopen Node-ID, because the Node-ID is defined via the slot number selected.

9.2 Configuration via SIMATIC Manager

9.2.1 Course of Configuration

Note:
If operation with simultaneous use of the CANopen and the CAN-Layer-2 function is required, the CANopen-configuration should always be carried out first, because the slot number is important here (= Node-ID). The slots which are not used for CANopen can be configured for the CAN-Layer-2 data transfer. The slot number is of minor importance for CAN-Layer-2 data transfer.

In the following the configuration of a CANopen-DP/2 is described which has not already been configured. If a CANopen configuration has already been carried out, continue with the configuration as described on page 70.

1. **Select CANopen-DP/2**
   Select menu *Hardware Catalogue* and then *Additional Field Devices* and *Gateway*.
   Select *esd CANopen-DP/2* there.

2. **Set PROFIBUS address**
   Set the PROFIBUS address as described in chapter 9.2.2 on page 65.

3. **Parameter Telegram (set CAN bit rate, general configuration and CANopen module ID)**
   Configure the settings by means of the parameter telegram as described in chapter 9.2.3 on page 66.

4. **Assignment of the Slots and setting the CANopen node-ID**
   Assign the slots as described in chapter 9.2.4 on page 69.

5. **Configuration of the Slots (PLC address)**
   Configure the slots as described in chapter 9.2.5 on page 70.

6. **Save settings on hard disk**
   Save the settings as described in chapter 9.2.6 on page 70.

9.2.2 Set PROFIBUS Address

A window opens in which you have to specify the PROFIBUS station address.

Attention!
The ***hexadecimal*** address set via the coding switches has to be ***converted*** into a ***decimal*** value and entered here!

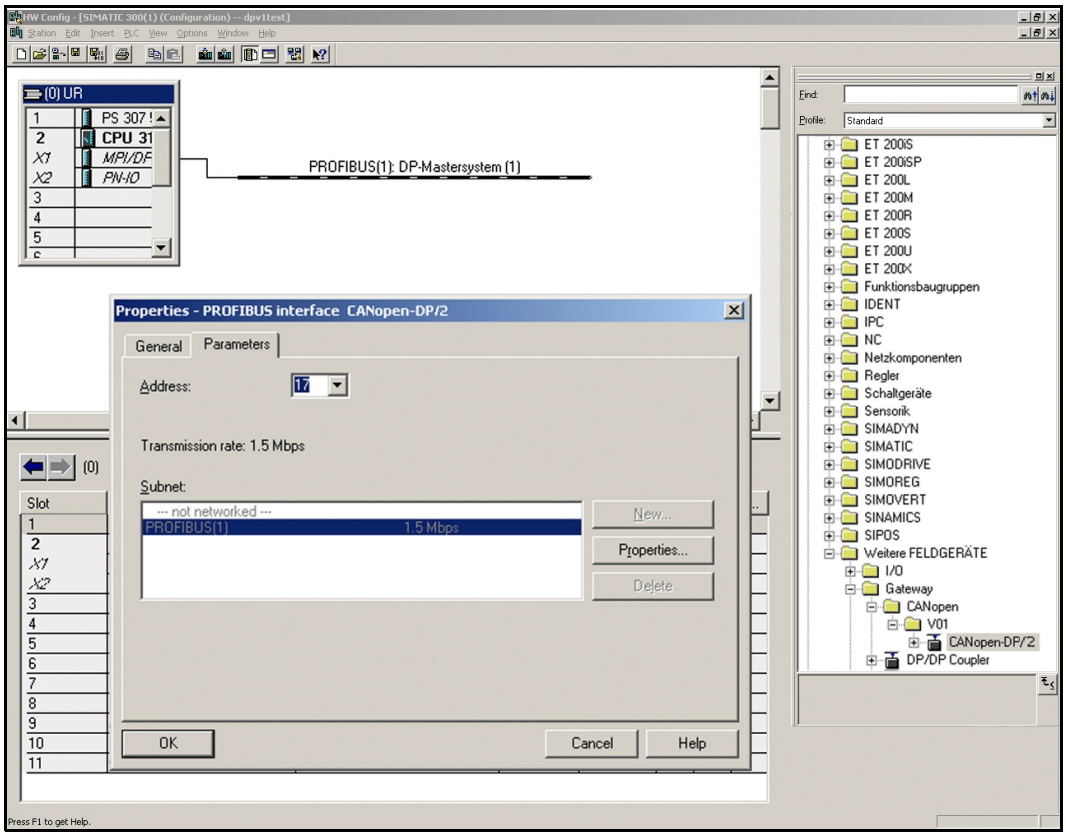

Fig. 14: Setting the PROFIBUS address of the CANopen-DP/2

9.2.3 Parameter Telegram (CAN-Bit Rate and Configuration)

The module ‘DP slave’ is now automatically inserted in the configuration window. Configuration settings can be changed by means of the parameter telegram.

The parameter setting of the DP-Slave can be done in the Properties window. To open the Properties window double click the header of the DP-slave window (here ‘`(17) CANopen-DP/2`’).

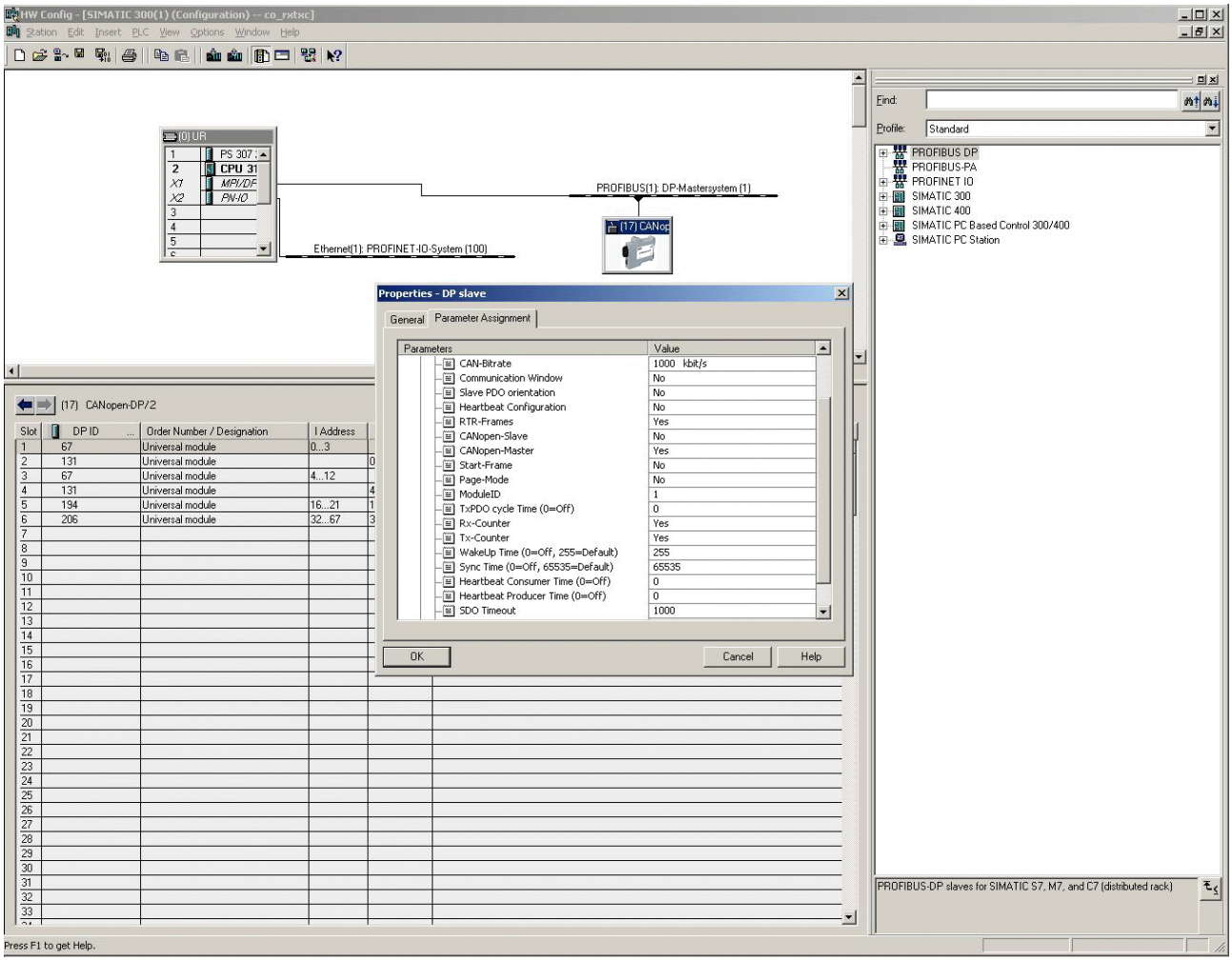

Fig. 15: Setting the parameters in the DP-slave properties window

Description of Parameters:

CAN-Bit rate **For the bit rate the following selections can be made:**

Bitrate [kbit/s]	*Bitrate* [kbit/s]
1000	100
666,6	66,6
500	50
333,3	33,3
250	20
166	12,5
125	10

Table 32: Setting the bit rate in 14 levels

Communication Window: (CW) This parameter activates the Communication Window. It is described in detail on page 76.

Slave PDO Orientation: (SO) This parameter is only relevant for CANopen applications (see page 27).

Heartbeat Configuration: (HC) This parameter is only relevant for CANopen applications (see page 27).

RTR-Frames: (NR) Transmit RTR-frames for the Rx-identifiers configured via PROFIBUS.

CANopen-Slave: (CS) see page 27

CANopen-Master: (CM) see page 27

Start-Frame: (AS) see page 27

Page-Mode: (PM) Activate Page-Mode.
Refer to the CAN-DP/2 software manual [4] for a detailed description of Page-Mode.

Module-ID: see page 28

TxPDO cycle time: This parameter is only relevant for CANopen applications (see page 28).

RX-Counter=yes: See page 29

Tx-Counter=yes: See page 29

Wakeup Time See page 29

SYNC Time: see page 30

9.2.4 Assigning the Slots of the DP-Slave

The number of slots to be used by the DP-slave for data exchange is set by double clicking the device ‘Universal Module’ in the *hardware catalog* with activated DP-slave window for each byte.
In the DP-slave window the assigned slots are represented by a ‘0’.

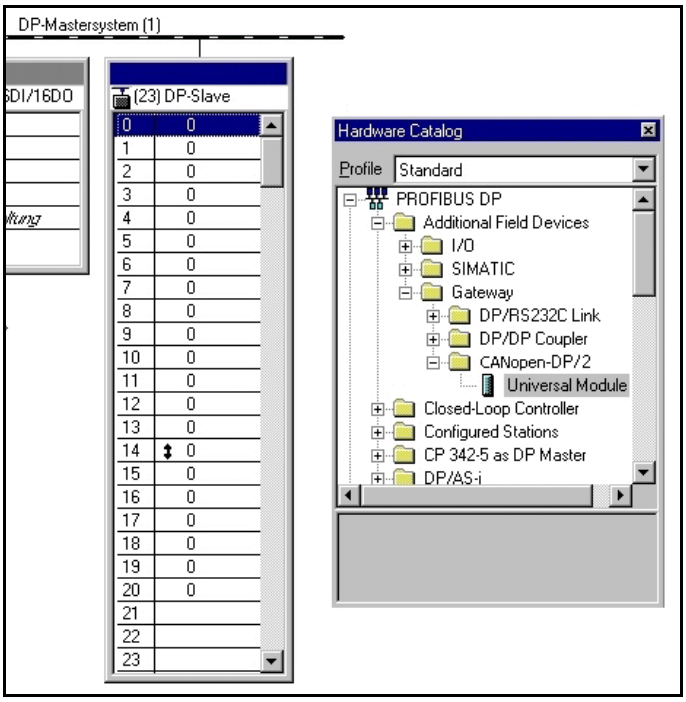

Fig. 16: Assignment of the DP-slave slots

9.2.5 Configuration of Slots for the CAN-Layer-2 Data Exchange

In order to configure the slots the slot entry has to be double clicked. A properties window opens in which the simulated PLC-slots are configured. Below, two examples with 11-bit identifiers are shown:

Data direction:	input	Data direction:	output
PLC address:	172_d	PLC address:	172_d
Length:	6	Length:	6
Unit:	byte	Unit:	byte
Consistent for:	whole length	Consistent for:	whole length
Identifier:	0289_h	Identifier:	0309_h
Form byte:	$B8_h$	Form byte:	$B8_h$

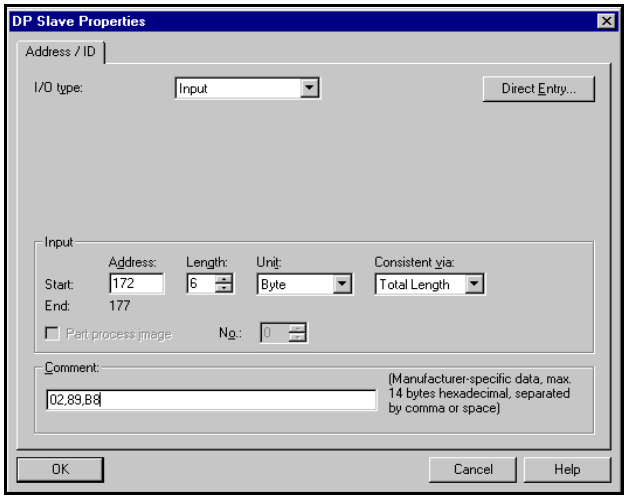

Fig. 5.2.3: Example: Configuration of input data

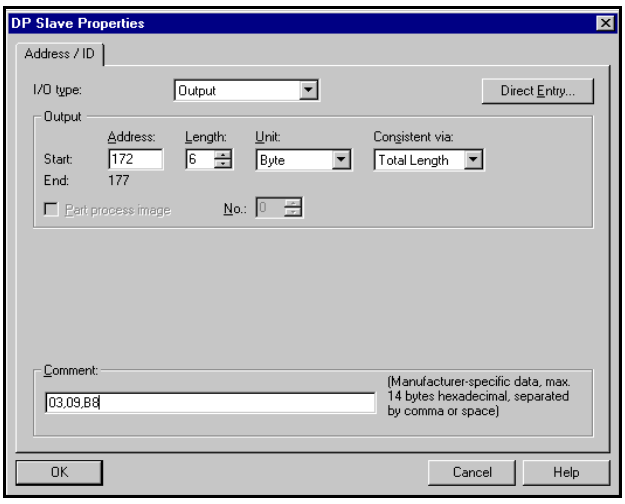

Fig. 5.2.4: Example: Configuration of output data

Attention!
In order to guarantee that the module operates faultless, always at least one output (any unit) has to be configured. The PROFIBUS controller does not trigger an interrupt, if no output is defined! If no CAN-assignment is to be made when an output is defined, it is permissible to specify the value $07F8_h$ as an identifier, here.

The individual parameters of the properties window will be explained in detail in the following chapter.

9.2.6 Save Settings to Hard Disc

Now you have to save the settings via menu points *Station/Save* to hard disc. Afterwards the settings are transmitted to the PLC by means of menu points *Target System/Load in Unit.*

9.2.7 Description of the Window '*Properties - DP-Slave*'

I/O-Type

In the field ***I/O-Type*** 'Input' or 'Output' has to be selected depending on the data direction required for CAN-Layer-2 applications. Other entries are not permissible. (The firmware distinguishes between CANopen- and CAN-Layer-2 data exchange by evaluation of this parameter.)

Address

In the ***Address*** field the PLC-I/O-address is entered as a **decimal value**.

Length, ***Unit***

By means of the fields ***Length*** and ***Unit*** the number of data bytes is defined. (Length ≤ 8 bytes and 4 words respectively).

Attention!:
If ***RX-Counter=yes***, the configured length for each received CAN message must be increased by one.
(If for example a message with 8-byte data length shall be received, in the input window *Properties DP- Slave* the entry must be *length* = 9.)

If ***Tx-Counter=yes,*** the configured length for each CAN message to be sent must be increased by one. (If for example a message with 8-byte data length shall be sent, in the input window *Properties DP- Slave* the entry must be *length* = 9).

Consistent over

The entry in the field ***Consistent over*** specifies whether the data is to be transmitted as individual units (bytes, words, etc.) or as complete package (1...8 bytes or 16 bytes in Communication-Window) during a PLC-cycle. This function is only to be set to 'whole length' if required, because the transmission as 'Unit' is faster.

Note:
If the data is to be transmitted consistently for the entire length, you have to specify this here *and* you have to use SFC14 and SFC15 (refer to Step7-PLC Manual).

Comment

In the first two (four) bytes of this field the CAN identifier and then the control byte *form*, each divided by commas, are transmitted.
For outputs the format byte can be optionally followed by 2 bytes *cycle* for the cyclic transmission of the CAN telegram defined in this slot. The two *cycle* bytes give the cycle time in milliseconds.
The data format for all properties is **hexadecimal** (!).
The entries of the ***Comment*** field are described in detail in the following chapters.

	Comment bytes of **11-bit** identifiers					
Byte No.	**1**	**2**	3	4	5	6 ... 15
Content:	**CAN identifier**		format byte	cycle time (optional for outputs)		not used
	ID_high	ID_low	form	Cycle_high	Cycle_low	

	Comment bytes of **29-bit** identifiers							
Byte No.	**1**	**2**	**3**	**4**	5	6	7	8 ... 15
Content:	**CAN identifier**				format byte	cycle time (optional for outputs)		not used
	ID_UU	ID_UL	ID_LU	ID_LL				
	bit31...bit24	bit23...bit16	bit15...bit08	bit07...bit00	form	Cycle_high	Cycle_low	

9.2.7.1 Enter the CAN Identifier in the *Comment* Field

The **CAN identifier** is transferred in the first two (four) bytes of the ***Comment*** field.
The bytes have to be entered separated by commas.

Example: The 11-bit identifier 0309_h shall be entered in the comment field.
The two bytes of the 11-bit identifier have to be entered into the *comment* field as: 03,09,

	Comment bytes of **11-bit** identifiers					
Byte No.	**1**	**2**	3	4	5	6 ... 15
Content:	**CAN identifier**		format byte	cycle time (optional for outputs)		not used
	ID_high	ID_low	form	Cycle_high	Cycle_low	
Example:	**03,**	**09,**	B8,	-	-	-
	(CAN identifier: 0309_h)		(format byte: $B8_h$)	no cyclic transmission		-

Note:
A 29-bit identifier requires four bytes and bit 29 must be set to ‘1' (counted 0...31 bits), in order to enable the module to distinguish between 11-bit and 29-bit identifiers.

The value range for the entry of the four bytes of a 29-bit identifier lies between 20,00,00,00 and 3F,FF,FF,FF.

Example: The 29-bit identifier 123456_h shall be entered in the comment field.
Please note that bit 29 has to be set to ‘1’ for 29-bit identifiers!
The four bytes of the 29-bit identifier have to be entered into the comment field as:
20,12,34,56

	Comment bytes of **29-bit** identifiers							
Byte No.	**1**	**2**	**3**	**4**	5	6	7	8 ... 15
Content:	**CAN identifier**				format byte	cycle time (optional for outputs)		not used
	ID_UU	ID_UL	ID_LU	ID_LL				
	bit31...bit24	bit23...bit16	bit15...bit08	bit07...bit00	form	Cycle_high	Cycle_low	
Example:	**20,**	**12,**	**34,**	**56,**	B8,	-	-	-
	(CAN identifier: 123456_h)				(format byte: $B8_h$)	no cyclic transmission		-

If ‘input’ has been selected in the *I/O-Type* field, the CAN identifier entered there is regarded as an Rx-identifier by the PLC. If ‘output’ has been selected in the *I/O-Type*, the CAN identifier entered here is a Tx-identifier.

Attention!
No Rx-identifier must be assigned twice!

If the *same* Rx-identifier has inadmissibly been selected on PLC-address 50 and address 51, no new Rx-data would be received on address 50 after the Rx-identifier has been assigned.
The data received last remained unchanged.

This Rx-identifier rule is also valid for the Rx-identifier activated via the Communication Window.

9.2.7.2 Setting the Data Format with the Control Byte *form*

The control byte ***form*** is transferred in the ***Comment*** field behind the first two (four, for 29-bit identifier) bytes for the **CAN identifier**.
form is used to convert the user data from Motorola format (high byte first) into Intel format (low byte first).

Attention!
The *form* byte always has to be entered in hexadecimal form!

Background: Messages which are longer than 1 byte are normally transmitted via a CANopen network in Intel notation, while the Siemens PLC operates in Motorola format.

Starting with bit 7 of the format byte you can decide whether the following byte is to be converted as well, i.e. swapped, or not. If a '1' is specified for a byte, the following bytes are converted up to and inclusive the next '0' transmitted. The functionality can be explained best by means of an example.

Example: A CAN telegram has got a date in Intel format in the first word, followed by two bytes which are not to be swapped and a long word in the last four bytes in Intel format again. Binary the following representation results for the format byte:

Bit No.	7	6	5	4	3	2	1	0
Bit of *format-byte*	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**
hex	8				E			
action	begin swap	end swap	unchanged	unchanged	begin swap	swap	swap	end swap

Data-bytes	1	2	3	4	5	6	7	8
CAN-frame	2 byte Intel format		byte 3	byte 4	4 byte Intel format			
PLC-data	2 byte Motorola format		byte 3	byte 4	4 byte Motorola format			

From this the format byte results in $8E_h$. If all eight bytes are to be swapped, for instance, value FE_h is specified for the format byte.

The lowest bit is generally without significance, because the telegram and therefore the formatting has been completed. The bit should always be set to 0.

Note:
The parameter '*form*' must always be set, even if no byte swapping is necessary. In this case the parameter has to be set to '00'.

9.2.7.3 Setting for the Cyclic Transmission via *Cycle*

For outputs the format byte in the ***Comment*** field can be optionally followed by 2 bytes, which trigger the CANopen-DP/2 to cyclically transmit the CAN telegram defined in this slot every *cycle* milliseconds. The cyclic transmission is then carried out independent of changes in the telegram.

The entry of the two bytes *cycle* for the cycle time must be placed in the ***Comment*** field behind the two (four, for 29-bit identifiers) bytes of the ***CAN identifier*** and the following command byte ***form***.
If the cyclic transmission shall not be used, the two cycle bytes can be left out.
The first of the two bytes for cyclic transmission is the higher order byte.
The data format in the ***Comment*** field is **hexadecimal** (!).

Example: A defined CAN identifier shall be cyclically transmitted every 1 second.
The cycle time *cycle* has to be entered hexadecimal in milliseconds. The value must be converted as follows:

$$1s = 1000ms = 03E8_h$$

For the cycle time *cycle* the values 03_h and $E8_h$ have to be entered in the *Comment* field as byte 4 and 5 (or byte 6 and 7 for 29-bit identifier respectively).

	Comment bytes of **11-bit** identifiers					
Byte No.	1	2	3	**4**	**5**	6 ... 15
Content:	CAN identifier		format byte	**cycle time**		not used
	ID_high	ID_low	form	Cycle_high	Cycle_low	
Example:	03,	09,	B8,	**03,**	**E8,**	-
	(CAN identifier: 0309_h)		(format byte: $B8_h$)	(cycle time: $03E8_h$)		-

	Comment bytes of **29-bit** identifiers							
Byte No.	1	2	3	4	5	**6**	**7**	8 ... 15
Content:	CAN identifier				format byte	**cycle time**		not used
	ID_UU	ID_UL	ID_LU	ID_LL				
	bit31...bit24	bit23...bit16	bit15...bit08	bit07...bit00	form	Cycle_high	Cycle_low	
Example:	20,	12,	34,	56,	B8,	**03,**	**E8,**	-
	(CAN identifier: 123456_h)				(format byte: $B8_h$)	(cycle time: $03E8_h$)		-

9.3 The Communication Window

9.3.1 Introduction

The cyclic PROFIBUS-accesses address the CANopen PDO-objects of the configured CANopen-nodes. Via acyclic accesses the CANopen SDO-accesses can be carried out.

The *Communication Window* features the option to access all CAN-identifiers on the layer-2-level, without CANopen-support.

In the Communication Window only one PLC-address is needed to access different Tx-identifiers and different Rx-identifiers. This is possible, because the identifiers of the CANopen-modules are transmitted as parameters together with the data at each access.

The disadvantage of the Communication Window is the lower data flow, though. Therefore we recommend to use the Communication Window for non-time-critical accesses only (e.g. parameterisation).

Attention!
The Communication Window does not support CANopen-functions, but operates with CAN-Layer-2 functions.

Attention!
The data length must always be 16 bytes in the configuration!
The identifier to be used is always $FFEF_h$!

The Communication-Window is described in detail on the following pages.

9.3.2 Configuration of the Communication Window

The Communication Window is configured via PROFIBUS. An entry for each of the transmission and reception of data via the Communication Window is required. The firmware only accepts these two entries.

The following two pictures show the required properties. **Apart from the PLC-address and the specifications for the SYNC time in the comment bytes 4 and 5, all parameters have been specified**. Even the identifier cannot be selected freely! Consistency has to be defined via total length. A shared PLC address or different PLC addresses are permissible for input and output direction.

Data direction:	input	Data direction:	output
PLC address:	any (example: 30)	PLC address:	any (example: 30)
Length:	16 (always)	Length:	16 (always)
Unit:	byte	Unit:	byte
Consistent for:	whole length!	Consistent for:	whole length!
Identifier:	$FFEF_h$ (always)	Identifier:	$FFEF_h$ (always)
Form byte:	00_h	Form byte:	00_h
Sync time:	$00\ 00_h$	Sync time:	$00\ 00_h$

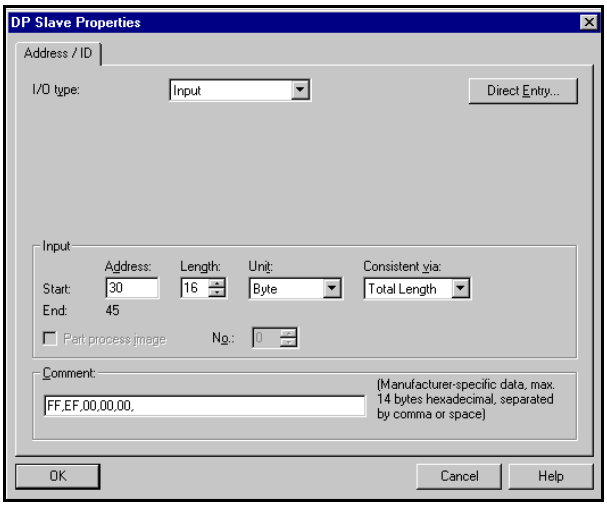

Fig. 5.4.1: Configuring the input path of the Communication Window

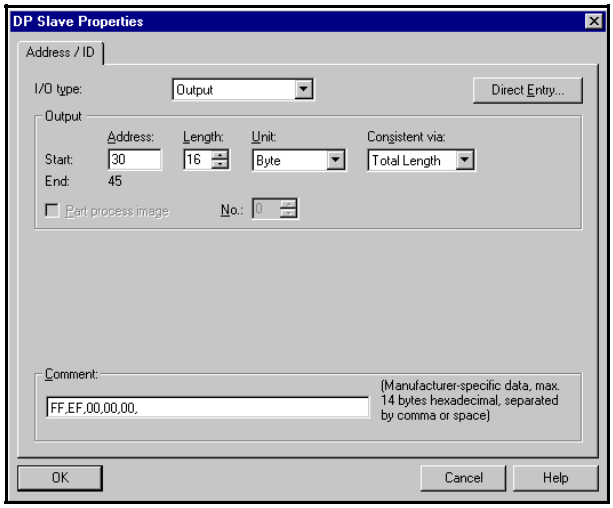

Fig. 5.4.2: Configuring the output path of the Communication Window

9.3.3 Format of Communication Window

The 16 bytes of the Communication Window are assigned differently, according to data direction.

9.3.3.1 Write Bytes of the Communication Window

(command setting and transmitting of data PLC -> CANopen -> CAN)

Bytes of Communication Window	Contents
1 2	high byte of CAN identifier (identifier bits [15] 10...8) low byte of CAN identifier (identifier bits 7...0)
3 4	with 11-bit CAN identifier byte 3 and 4 always ‘0’ with 29-bit CAN identifier byte 3: identifier bits 28...24 byte 4: identifier bits 23...16
5 6 7 8 9 10 11 12	data byte 1 data byte 2 data byte 3 data byte 4 data byte 5 data byte 6 data byte 7 data byte 8
13	data length for transmission jobs (Tx)
14	PLC loop counter (has to be incremented in pulse with OB1 in order to tell the gateway the OB1 cycle)
15	sub command (always set to ‘0’)
16	command (description refer page 80)

Table 33: Write bytes of the Communication Window

9.3.3.2 Read Bytes of the Communication Window

(command acknowledge and reception of data CAN -> CANopen -> PLC)

Bytes of the Communication Window	Contents
1 2	as long as no received data are available $EEEE_h$, otherwise high byte of CAN identifier (identifier bits [15] 10...8) low byte of CAN identifier (identifier bits 7...0)
3 4	with 11-bit CAN identifier byte 3 and 4 always ‘0’ with 29-bit CAN identifier byte 3: identifier bits 28...24 byte 4: identifier bits 23...16
5 6 7 8 9 10 11 12	data byte 1 data byte 2 data byte 3 data byte 4 data byte 5 data byte 6 data byte 7 data byte 8
13	number of data bytes received
14	return of the PLC-loop counter which has been transmitted to the gateway via the last PROFIBUS telegram
15	return of the sub command
16	error code of the read function (not supported at the moment)

Table 34: Read bytes of the Communication Window

The following table shows commands which are currently being supported. The sub command is not yet being evaluated and should always be set to ‘0’ at command call, therefore.

Command	Function
1	transmit data
3	receive data at enabled Rx-identifiers
4	enable Rx-identifier for data reception
5	deactivate reception (command 4)
6	transmits an RTR frame
7	executes command 4 and command 6
(11)	(reserved)
20	If the gateway is configured as CANopen master: Cyclical transmission of the CANopen SYNC command (ID 80_h, len = 0)

Table 35: Commands of Communication Window

Attention!
A command is only completely processed, if, when reading the Communication Window, byte 14 of the CANopen-DP/2-module provides the value of the PLC-loop counter which was assigned at the command call.
Before the following command is called, it is therefore necessary to check byte 14 first!

Explanations of the commands:

Command 1: Send data

In order to send data via the Communication Window the CAN identifier has to be specified in bytes 1 and 2 (or 1...4 for 29-bit identifiers). In addition to the number of bytes to be transmitted, a PLC-loop counter has to be specified. The loop counter has to be realised by the user. It is required to provide the CANopen-DP/2-gateway with the OB1-cycle of the PLC.

Command 3: Reception on enabled Rx-identifiers

The reception of data requires the CAN-Rx-identifiers which are to receive data to be enabled (see command 4).
After reception command 3 has been written, read accesses to the Communication Window will give you the data structure shown on page 79. The Rx-data is received asynchronously to the PLC-cycle. Until valid data has been received the value $EEEE_h$ will be returned in the first bytes in read accesses. Only after valid data has been received the Rx-identifier of the read frame in the first bytes becomes readable. In addition, the read command which requested the reception of data is assigned by means of the returned PLC-loop counter in byte 14.

The CANopen-DP/2 module has got a FIFO-memory for 255 CAN-frames to buffer the received Rx-data. If several Rx-frames are to be received on one Rx-identifier, or if frames of various Rx-identifiers enabled for reception are received, the data is not lost, as long as the PLC reads out the FIFO memory quicker than it is being filled.

Command 4: Enabling Rx-identifiers for reception

By means of this command the Rx-identifier whose data is to be received has to be enabled. More than one Rx-identifier can be enabled at the same time. For this, the command has to be called an according number of times.

Command 5: Deactivate reception (command 4)

After this command has been called no data is received any longer on the specified Rx-identifiers.

Command 6: Sending an RTR-frame

By means of this command a remote-request frame is transmitted. Prior to the transmission the reception on the Rx-identifier has to be enabled by command 4.

Command 7: Executes command 4 and command 6

See there.

Command 20: Cyclical transmission of the CANopen command SYNC

The CANopen-DP/2 device can cyclically transmit the command SYNC for simple CANopen applications.

The command is transmitted as shown in the tables above. The cycle is specified e.g in the properties window in bytes 4 and 5 when the Communication Window is configured (refer to page 77).

The cycle is specified in milliseconds.
Value range: 0...FFFE $_h$ (0...65534 ms)

Attention!
In order to guarantee that all CANopen users have received their new data when they receive the SYNC command, the cyclical transmission command of the SYNC command cannot interrupt transmission of a DP-telegram on the CAN. That means that the SYNC command is delayed until the DP-telegram has been transmitted, if its transmission and the transmission of a SYNC command coincide.
This can result in slight changes of time in the cyclical transmission of the SYNC command.

Attention!
SYNC Time can be set in two different ways:
1. In the parameter telegram in the DP-properties window (see page 30)
2. Via byte 4 and 5 of the Communication window (see page 77)

These specifications are equal, i.e. the last specification made is valid!

9.3.4 Examples on the Communication Window

9.3.4.1 Transmitting Data

1. Basic Setting of the Communication Window
   The basic settings have to be made only once when setting up the Communication Window.

1.1 Activating the Communication Window during the configuration of the DP-gateway (see page 67)
Communication Window: yes

1.2 Definition of the 16 input and output bytes of the Communication Window (see page 77) e.g.

Data direction:	**input**	Data direction:	**output**
PLC-address:	e.g. here: 30	PLC-address:	e.g. here: 30
Length:	16 (always)	Length:	16 (always)
Unit:	byte	Unit:	byte
Consistently via:	entire length!	Consistently via:	entire length!
Identifier:	$FFEF_h$ (always)	Identifier:	$FFEF_h$ (always)
Form byte:	00_h	Form byte:	00_h
Sync time:	$00\ 00_h$	Sync time:	$00\ 00_h$

1.3 Program PLC-loop counter
8-bit loop counter for handshake function between PLC and gateway

	PLC-Cycle (Pseudo Code):
...	...
1	Read Byte 14 (returned loop counter) of '**Read** Bytes of Communication-Windows' (refer to page 79)
2	Compare Byte 14 of the '**Read** Bytes of Communication-Windows' with PLC-loop counter byte 14 of the '**Write** Bytes of Communication-Windows'(refer to page 78), if unequal go to 6., if equal go to 3.
3	Increase PLC-loop counter (Byte 14) of '**Write** Bytes of Communication-Window' (refer to page 78)
4	Evaluation of '**Read** Bytes of Communication-Windows' (refer to page 79), i.e. the evaluation of the answer to the last command or received CAN frame (depending on the application).
5	Send new '**Write** Bytes of Communication-Window' (refer to page 78) with increased loop-counter value of 3. and if necessary new application data.
6	Continue PLC program (new request at the next program cycle)

2. Start Transmission Command by Writing the 16 Bytes of the Communication Window

Byte of Communication Window	Contents	**Example here [hex]**
1 2	high byte of CAN-identifier (identifier bit [15] 10...8) low byte of CAN-identifier (identifier bit 7...0)	**00** **12**
3 4	bytes 3 and 4 always ‘0’ for 11-bit identifier	**00** **00**
5 6 7 8 9 10 11 12	data byte 1 data byte 2 data byte 3 data byte 4 data byte 5 data byte 6 data byte 7 data byte 8	**00** **01** **02** **03** **04** **05** **06** **07**
13	data length for transmission commands (Tx)	**08**
14	PLC-loop counter	**8-bit counter**
15	sub-command (always set to ‘0’)	**00**
16	command ‘transmit data’	**01**

The data bytes 0...8 are transmitted on Tx-identifier 0012_h .

In order to acknowledge the execution of the command a read access to byte 14 of the Communication Window should follow. It has to have the same value of the PLC-loop counter as when the command was called.

9.3.4.2 Receiving Data

1. Basic Setting of the Communication Window

The basic settings of the Communication Window have already been described in the example above ‘Transmitting Data’.

2. Receiving Data

2.1 Enabling the Rx-Identifier for Reception

In this example the data of the Rx-identifier 0123_h are to be received.

Byte of Communication Window	Contents	**Example here [hex]**
1 2	high byte of CAN-identifier (identifier bit [15] 10...8) low byte of CAN-identifier (identifier bit 7...0)	**01** **23**
3 4	bytes 3 and 4 always ‘0’ for 11-bit identifier	**00** **00**
5 6 7 8 9 10 11 12	data byte 1 data byte 2 data byte 3 data byte 4 data byte 5 data byte 6 data byte 7 data byte 8	**00** **00** **00** **00** **00** **00** **00** **00**
13	data length for transmission command (Tx)	**00**
14	PLC-loop counter	**8-bit counter**
15	sub-command (always set to ‘0’)	**00**
16	command ‘Enable Rx-Identifier’	**04**

In order to acknowledge the execution of the command a read access of byte 14 of the Communication Window should be made with every command call. It has to have the same value of the PLC-loop counter as it had when the command was called.

2.2 Initiate Reception of Data of the Enabled Rx-Identifier

Byte of Communication Window	Contents	**Example here [hex]**
1 2	high byte of CAN-identifier (identifier bit [15] 10...8) low byte of CAN-identifier (identifier bit 7...0)	**01** **23**
3 4	bytes 3 and 4 always ‘0’ for 11-bit identifier	**00** **00**
5 6 7 8 9 10 11 12	data byte 1 data byte 2 data byte 3 data byte 4 data byte 5 data byte 6 data byte 7 data byte 8	**00** **00** **00** **00** **00** **00** **00** **00**
13	data length for transmission commands (Tx)	**00**
14	PLC-loop counter	**8-bit counter + n**
15	sub-command (always set to ‘0’)	**00**
16	command ‘Read Rx-Identifier’	**03**

2.3 Reading the Data

After an undetermined time the Rx-data is received and can be accessed by reading the Communication Window. Since the data is received asynchronously to the PLC-cycles the Communication Window has to be read again and again until the data was received (polling). By comparing the values of the PLC-loop counter you can determine, whether the data received is the correct data from the read command.

A read access returns the following bytes:

Byte of Communication Window	Contents	**Example here [hex]**
1 2	high byte of CAN-identifier (identifier bit [15] 10...8) low byte of CAN-identifier (identifier bit 7...0)	**01** **23**
3 4	bytes 3 and 4 always ‘0’ for 11-bit identifier	**00** **00**
5 6 7 8 9 10 11 12	data byte 1 data byte 2 data byte 3 data byte 4 data byte 5 data byte 6 data byte 7 data byte 8	**AA** **BB** **CC** **DD** **EE** **FF** **00** **11**
13	data length	**08**
14	PLC-loop counter	**8-bit counter + n**
15	returned sub-command (without significance)	**00**
16	error code of the read function (without significance)	**00**

2.4 Deactivate Reception of Data on this Rx-Identifier

If no further data is to be received on this identifier, the reception is to be disabled again.

Byte of Communication Window	Contents	**Example here [hex]**
1 2	high byte of CAN-identifier (identifier bit [15] 10...8) low byte of CAN-identifier (identifier bit 7...0)	**01** **23**
3 4	bytes 3 and 4 always ‘0’ for 11-bit identifiers	**00** **00**
5 6 7 8 9 10 11 12	data byte 1 data byte 2 data byte 3 data byte 4 data byte 5 data byte 6 data byte 7 data byte 8	**00** **00** **00** **00** **00** **00** **00** **00**
13	data length for transmission commands (Tx)	**00**
14	PLC-loop counter	**8-bit counter + m**
15	sub-command (always set to ‘0’)	**00**
16	command ‘Disable Rx-Identifier’	**05**

10. Editing the GSD-File with a Text Editor

We recommend to configure the module with a PROFIBUS configuration tool, as e.g. the SIMATIC manager.
Not every PROFIBUS configuration software supports the “Universal Module” (see chapter: “5.1 Course of Configuration via SIMATIC Manager”).
If the “Universal Module” is not supported, the GSD-file has to be adapted via a text editor.

The configuration of a module is made by means of a configuration frame, whose content is entered in the GSD-file.

The frame of the configuration is sub-divided in three octets (see also PROFIBUS-Specification, Normative Part 8, page 738, Fig. 16):

Octet 1: *Number_of_the_manufacturer-specific data*
Octet 2: *Number_of_output_bytes*
Octet 3: *Number_of_input_bytes*
Octet 4: *Manufacturer-specific_configuration_byte*
:
:
Octet 17: *Manufacturer-specific_configuration_byte*

The octets have the following meaning:

Octet 1: *Number_of_manufacturer-specific_data*

Because the CANopen-DP/2 always uses a specific ID-format to represent a connected CAN-module, the identifier-byte has the following structure (see also PROFIBUS-Specification-Normative-Part-8, page 737):

MSB LSB

Bit-No.:	7	6	5	4	3	2	1	0
Content:	00: free place 01: Input 10: Output 11: In/Output		always 0	always 0	Length of the manufacturer-specific data: 0010 for the PDO1 0110 for the PDO1 and PDO2 1010 for the PDO1, PDO2 and PDO3 1110 for the PDO1, PDO2., PDO3 and PDO4			

Example Octet 1:

Bit-No.:	7	6	5	4	3	2	1	0
Content:	1	1	0	0	1	0	1	0

= 0xCA Module is an input and output (0xCO) and 10 bytes manufacturer-specific data (PDO1, PDO2 and PDO3) will follow.

Octet 2: *Number_of_output_bytes*

Octet 2 gives the consistency, the structure (byte/word) and the number of the output bytes.
Length bytes of the output as seen from the PROFIBUS master (see also PROFIBUS-Specification-Normative-Part-8, page 738)

MSB LSB

Bit-No.:	7	6	5	4	3	2	1	0
Content:	Consistency over 0: byte or word 1: complete length	Length-format 0: byte-structure 1: word-structure	Number of output bytes					

Bit					Meaning
5	4	3	2	1	
0	0	0	0	0	1 byte, resp. 1 word
:					:
1	1	1	1	1	64 bytes, resp. 64 words

Note:
If the parameter *Slave PDO-Orientation* is set to ‘yes’ in the parameter telegram, CAN frames are transmitted at the outputs with standard TxPDO-identifiers.
If the parameter *Slave PDO-Orientation* is ‘no’ (default), the CAN frames are transmitted via RxPDO-identifiers.

The number of the output bytes is the summation of all RxPDO-lengths (for SlavePDO-Orientation ‘no’), or summation of all TxPDO-lengths (for SlavePDO-Orientation ‘yes’)

Example Module 1:
The parameter *Slave PDO-Orientation* is set to ‘no’ in the parameter telegram.
The length of RxPDO1 is 8 bytes,
the length of RxPDO2 is 5 bytes and
the length of RxPDO3 is 3 bytes

Bit-No.:	7	6	5	4	3	2	1	0
Content:	0	0	0	0	1	1	1	1

= 0x0F 16 bytes RxPDO output data

Example Module 2:
The parameter *Slave PDO-Orientation* is set to ‘yes’ in the parameter telegram.
The length of TxPDO1 is 7 bytes,
the length of TxPDO2 is 4 bytes and
the length of TxPDO3 is 6 bytes

Bit-No.:	7	6	5	4	3	2	1	0
Content:	0	0	0	1	0	0	0	0

= 0x10 17 bytes TxPDO output data

Octet 3: *Number_of_input_bytes*

Octet 3 gives the consistency, the structure (byte/word) and the number of the input bytes.
Length bytes of the input as seen from the PROFIBUS master (see also PROFIBUS-Specification-Normative-Part-8, page 738)

MSB LSB

Bit-No.:	7	6	5	4	3	2	1	0
Content:	Consistency over 0: byte or word 1: complete length	Length-format 0: byte-structure 1: word-structure	Number of output bytes					

Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Meaning
0	0	0	0	0	1 byte, resp. 1 word
:					:
1	1	1	1	1	64 bytes, resp. 64 words

The number of the input bytes is the summation of all TxPDO-lengths (for SlavePDO-Orientation ‘no’), or summation of all RxPDO-lengths (for SlavePDO-Orientation ‘yes’)

Example Module 1:
The parameter *Slave PDO-Orientation* is set to ‘no’ in the parameter telegram.
The length of TxPDO1 is 7 bytes,
the length of TxPDO2 is 4 bytes and
the length of TxPDO3 is 6 bytes

Bit-No.:	7	6	5	4	3	2	1	0
Content:	0	0	0	1	0	0	0	0

= 0x10 17 bytes TxPDO input data

Example Module 2:
The parameter *Slave PDO-Orientation* is set to ‘yes’ in the parameter telegram.
The length of RxPDO1 is 8 bytes,
the length of RxPDO2 is 5 bytes and
the length of RxPDO3 is 3 bytes

Bit-No.:	7	6	5	4	3	2	1	0
Content:	0	0	0	0	1	1	1	1

= 0x0F 16 bytes RxPDO input data

Octet 4, 5 ... 17: *Manufacturer-specific_configuration_byte*

The content of octet 4 -17 depends on the setting of parameter *Slave PDO-Orientation*.

Note:
If the parameter *Slave PDO-Orientation* is set to 'yes' in the parameter telegram, CAN frames are transmitted at the outputs with standard TxPDO-identifiers and CAN frames are received at the inputs with standard RxPDO-identifiers.
If the parameter *Slave PDO-Orientation* is 'no' (default), the CAN frames are transmitted via RxPDO-identifiers and they are received via TxPDO-identifiers.

Octet	Manufacturer-specific data	
	Slave PDO Orientation: **no**	*Slave PDO Orientation:* **yes**
4	Format byte TxPDO1	Format byte RxPDO1
5	Format byte RxPDO1	Format byte TxPDO1
6	Length TxPDO1	Length RxPDO1
7	Length RxPDO1	Length TxPDO1
8	Format byte TxPDO2	Format byte RxPDO2
9	Format byte RxPDO2	Format byte TxPDO2
10	Length TxPDO2	Length RxPDO2
11	Length RxPDO2	Length TxPDO2
12	Format byte TxPDO3	Format byte RxPDO3
13	Format byte RxPDO3	Format byte TxPDO3
14	Length TxPDO3	Length RxPDO3
15	Length RxPDO3	Length TxPDO3
16	Format byte TxPDO4	Format byte RxPDO4
17	Format byte RxPDO4	Format byte TxPDO4

10.1 Example Module 1: Manufacturer-specific Data (*SO:* no)

Octet	Manufacturer-specific Data	Entry	Meaning
4	Format byte TxPDO1	0x00	no byte-swapping
5	Format byte RxPDO1	0x00	no byte-swapping
6	Length TxPDO1	0x07	-
7	Length RxPDO1	0x08	-
8	Format byte TxPDO2	0x00	no byte-swapping
9	Format byte RxPDO2	0x00	no byte-swapping
10	Length TxPDO2	0x04	-
11	Length RxPDO2	0x05	-
12	Format byte TxPDO3	0x00	no byte-swapping
13	Format byte RxPDO3	0x00	no byte-swapping

The configuration-frame for module 1 has the following structure and has to be inserted into the GSD-file.

Example for manual GSD-file entries:

```
...

Module="Name of the Module"  0xCA, 0x0F, 0x10, 0x00, 0x00, 0x07, 0x08, 0x00,
                             0x00, 0x04, 0x05, 0x00, 0x00
1
EndModule

...
```

Meaning of the entries:

1... Unambiguous reference number (1...65535)

entries under `Module=`:

“Name of the module” ... Comment to name the module

Octet 1: 0xCA... Module is an in- and output (0xC0) and 10 bytes of manufacturer- specific data (1. PDO, 2. PDO und 3. PDO) will follow.

Octet 2: 0x0F... Consistency over byte, the length-format is byte-structure (0x0F) and 16 bytes output data are transferred (0x0F = 16...1).

Octet 3: 0x10... Consistency over byte, the length-format is byte-structure (0x10) and 17 bytes input-data are transferred (0x10 = 17...1).

Manufacturer-specific data:

Octet 4: 0x00...	Format byte TxPDO1	0x00	no byte-swapping, i.e. the order of the data is not changed
Octet 5: 0x00...	Format byte RxPDO1	0x00	no byte-swapping
Octet 6: 0x07...	Length TxPDO1	0x07	length of TxPDO1: 7 bytes
Octet 7: 0x08...	Length RxPDO1	0x08	length of RxPDO1: 8 bytes
Octet 8: 0x00...	Format byte TxPDO2	0x00	no byte-swapping
Octet 9: 0x00...	Format byte RxPDO2	0x00	no byte-swapping
Octet 10: 0x04...	Length TxPDO2	0x04	length of TxPDO2: 4 bytes
Octet 11: 0x05...	Length RxPDO2	0x05	length of RxPDO2: 5 bytes
Octet 12: 0x00...	Format byte TxPDO3	0x00	no byte-swapping
Octet 13: 0x00...	Format byte RxPDO3	0x00	no byte-swapping

The length of TxPDO3 results from the length byte for input less the lengths of TxPDO1 and TxPDO2:
17 bytes input data
- 7 bytes TxPDO1
- 4 bytes TxPDO2
==========
6 bytes for TxPDO3

The length of RxPDO3 results from the length byte for output less the lengths of RxPDO1 and RxPDO2:
16 bytes input data
- 8 bytes RxPDO1
- 5 bytes RxPDO2
==========
3 bytes for RxPDO3

Attention!
Please note, that the GSD-file has to be renamed. The file name may be maximum 8 characters long. Some configuration-software for the PROFIBUS Master does not operate with longer file names.

10.2 Example Module 2: Manufacturer-specific Data (*SO:* yes)

Octet	Manufacturer specific data	Entry	Meaning
4	Format byte RxPDO1	0x00	no byte-swapping
5	Format byte TxPDO1	0x00	no byte-swapping
6	Length RxPDO1	0x08	-
7	Length TxPDO1	0x07	-
8	Format byte RxPDO2	0x00	no byte-swapping
9	Format byte TxPDO2	0x00	no byte-swapping
10	Length RxPDO2	0x05	-
11	Length TxPDO2	0x04	-
12	Format byte RxPDO3	0x00	no byte-swapping
13	Format byte TxPDO3	0x00	no byte-swapping

The configuration-frame for module 2 has the following structure and has to be inserted into the GSD-file.

Example for manual GSD-file entries:

```
...

Module="Name of the Module"   0xCA, 0x10, 0x0F, 0x00, 0x00, 0x08, 0x07, 0x00,
                              0x00, 0x05, 0x04, 0x00, 0x00
2
EndModule

...
```

Meaning of the entries:

2... Unambiguous reference number (1...65535)

entries under `Module=`:

“Name of the module” ... Comment to name the module

Octet 1: 0xCA... Module is an in- and output (0xC0) and 10 bytes of manufacturer-specific data (1. PDO, 2. PDO und 3. PDO) will follow.

Octet 2: 0x10... Consistency over byte, the length-format is byte-structure (0x10) and 17 bytes output data are transferred (0x10 = 17...1).

Octet 3: 0x0F... Consistency over byte, the length-format is byte-structure (0x0F) and 16 bytes input data are transferred (0x0F = 16...1).

Manufacturer-specific data:

Octet: 0x00...	Format byte RxPDO1	0x00	no byte-swapping, , i.e. the order of the data is not changed
Octet 5: 0x00...	Format byte TxPDO1	0x00	no byte-swapping
Octet 6: 0x08...	Length RxPDO1	0x08	length of RxPDO1: 8 bytes
Octet 7: 0x07...	Length TxPDO1	0x07	length of TxPDO1: 7 bytes
Octet 8: 0x00...	Format byte RxPDO2	0x00	no byte-swapping
Octet 9: 0x00...	Format byte TxPDO2	0x00	no byte-swapping
Octet 10: 0x05...	Length RxPDO2	0x05	length of RxPDO2: 5 bytes
Octet 11: 0x04...	Length TxPDO2	0x04	length of TxPDO2: 4 bytes
Octet 12: 0x00...	Format byte RxPDO3	0x00	no byte-swapping
Octet 13: 0x00...	Format byte TxPDO3	0x00	no byte-swapping

The length of TxPDO3 results from the length byte for output less the lengths of TxPDO1 and TxPDO2:
17 bytes output-data
- 7 bytes TxPDO1
- 4 bytes TxPDO2
==========
6 bytes for TxPDO3

The length of RxPDO3 results from the length byte for input less the lengths of RxPDO1 and RxPDO2:
16 bytes input-data
- 8 bytes RxPDO1
- 5 bytes RxPDO2
==========
3 bytes for RxPDO3

Attention!
Please note, that the GSD-file has to be renamed. The file name may be maximum 8 characters long. Some configuration-software for the PROFIBUS Master does not operate with longer file names.

11. Glossary

PROFIBUS-DP/PLC-Terms

Master . . . . . Master devices control the data transfer on the PROFIBUS. A master can transmit messages without request.

Slave . . . . . Slave devices are peripheral devices such as in-/output devices, valves, drives and transducers. They have no bus access authorization, i.e. they can only accept received messages or transmit messages to the master on request.

Class 1 Master . . . . . PROFIBUS controller which controls its decentral peripheral devices via cyclic data transfer.

Class 2 Master . . . . . PROFIBUS controller which acyclically exchanges data with a slave.

CAN/CANopen Terms

CAN . . . . . Controller Area Network

CAN-Layer-2 . . . . . Layer of the OSI-Layer model
On the level of CAN-Layer-2 control and management structures are not supported as under CANopen which is classified as CAN-Layer-7.

CANopen Node . . . . CANopen module
A CANopen node can feature master or/and slave functions.

CiA . . . . . CAN in Automation e.V.
Association for the development of CAN in industrial applications.

COB . . . . . Communication Object

Communication Window . . . . . Window for the transmission of CAN data on CAN-Layer-2 level

Frame . . . . . Also called message
General term for a CAN message which consists of control and user data.

DS-xxx . . . . . Draft Standard
Some CAN-standards can be downloaded directly from the CiA-homepage:
www.can-cia.org

Emergency Data Object . . . . . Emergency data

Heartbeat . . . . . Cyclical recurring message for the monitoring of the operability of a CANopen module.

Master . . . . . CANopen Master
CANopen module which carries out control tasks on the CAN-bus.

Node-ID . . . . . CANopen-node number
Number for identification of a CANopen-node ($1...127_d$). The PDO-identifier result from the Node-ID.

NMT . . . . . Network Management (Master)

PDO . . . . . Process Data Object
The real-time data transfer of process data is performed by PDOs. The data transfers are not acknowledged.

RTR . . . . . Remote Request
CAN-frame which requests another CAN-module to transmit data.

Rx . . . . . receive

MPDO . . . . . Multiplexed PDO
Via this PDO all modules can be addressed with a single CAN-frame simultaneously (Broadcast).

Page Mode . . . . . CAN-Layer-2 function to control more than 48 CAN-identifier in a PROFIBUS-telegram, see [4]

SDO . . . . . Service Data Object
With SDOs communication and parameter data can be transferred. The data transfers are acknowledged.

Slave . . . . . CANopen-Slave

START . . . . . Start-frame to start the CANopen-module

SYNC . . . . . Synchronisation-frame is used to synchronise the CANopen-modules

Tx . . . . . transmit

12. References

[1] CiA 301
CANopen Application Layer and Communication Profile
Version 4.2.0, February 21, 2011
www.can-cia.org

[2] PROFIBUS Specification,
PROFIdrive-Profil Drive Technology
Version 3.1.2, September 2004
Order No: 3.172
www.profibus.com

[3] CAN-DP/2/CANopen-DP/2,
PROFIBUS-DP / CAN-Gateway,
Hardware Manual
Rev. 1.0
esd electronic system design gmbh
www.esd.eu

[4] CAN-DP/2,
PROFIBUS-DP / CAN-Gateway,
Software Manual
Rev. 1.0, January 16, 2013
esd electronic system design gmbh
www.esd.eu

13. License

The CANopen-DP/2 gateway uses the opensource FreeRTOS™ operating system. The FreeRTOS source code is published in terms of the GNU public license (GPL)
For the full license text please see esd's "3rd party lisensor notice" document that is part of the product's documentation on the enclosed CD.

Please contact esd for the full source code of the FreeRTOS used for the CANopen-DP/2.

The web address of FreeRTOS is: http://www.freertos.org/