CAN-DP/2

PROFIBUS-DP / CAN-Gateway

Software Manual

to Product: C.2907.02

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\CAN-DP2\Englisch\CAN-DP2_Software-Manual_en_11.wpd
Date of print:	2013-02-12

Described software version:	Command-File: DP/CANopen:	CBXDP_00 1.0.0

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 CAN-DP/2 Software Manual
1.1	10	New chapter: License notice to opensource FreeRTOS™ operating system

Technical details are subject to change without notice.

This page is intentionally left blank.

Contents

Page

Contents

Page

1. Overview

1.1 About this Manual

This manual describes the local firmware of the CAN-DP/2 module. The local firmware controls the data exchange between PROFIBUS-DP (abbreviated to PROFIBUS below) and CAN.

Layer 2 Implementation
The manual describes the Layer 2 implementation and the implemented CANopen functions.

Page Mode
Furthermore, the manual describes the Page Mode which was developed to allow more than 48 CAN identifiers to be controlled by one gateway. For a general understanding fundamental functions of the Page Mode will be described first, followed by descriptions of the function blocks (FBs) and data blocks (DBs), which are used to realize the Page Mode.

11-Bit and 29-Bit Identifier
The module CAN-DP/2 supports 11-bit and 29-bit CAN identifier (CAN2.0A/B).

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 CAN devices can be operated as PROFIBUS slaves.
The PROFIBUS output bytes are transmitted to the CAN bus. One to eight output bytes are assigned to an Tx-identifier and can optionally be transmitted cyclically . Rx-identifiers are assigned to the input bytes on CAN side. Received CAN data is treated as input data by the PROFIBUS.
The PROFIBUS station address is set directly at the CAN-DP/2 module by means of coding switches.

1.3 Configuration via PROFIBUS-DP

The CAN-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 which are assigned with further parameters such as the PLC address, data direction, data length and CAN identifier.

1.4 More addressable Identifiers via Page Mode

The Page Mode offers the chance to address more CAN identifiers than can be stored in one PROFIBUS telegram (that means more than 48). The number of the identifiers is only limited by the available memory range of the PLC and the CAN-Gateway.

Note:
Page Mode can only be used, if the Siemens SIMATIC Manager for S7 is used as configuration-tool!

2. Functionality of the Local Firmware

The following figure represents the functionality of the firmware.

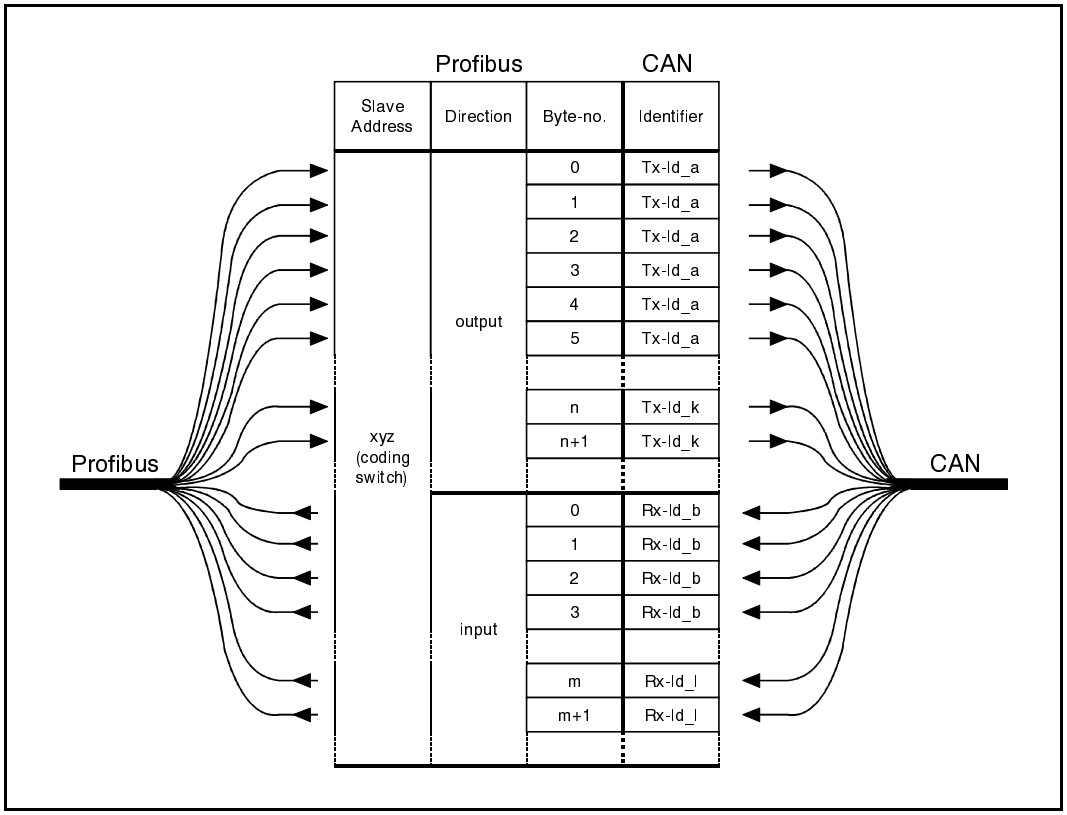

Fig. 2.1.1: Overview of functions of the CAN-DP/2 module

2.1 PROFIBUS-Slave Address

The CAN-DP/2 module simulates a slave module on the PROFIBUS side. The slave address is set by means of coding switches at the module. When switching on the module the hexadecimal PROFIBUS address set 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 (decimal) or smaller than 03_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 upper coding switch (HIGH) is used to set the MSBs, while the LSBs are set by means of the lower coding switch (LOW).

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 CAN-DP/2-module simulates a total of up to 300 bytes for the input direction and the output direction in the current software implementation. From these 300 bytes a maximum of 244 bytes can be selected for one data direction, otherwise the division into input bytes and output bytes is entirely up to the user. (Examples: 150 input bytes and 150 output bytes, or 244 input bytes and 56 output bytes).

One to eight bytes (16 bytes when using the communication window, see page 37) each are assigned to a Tx-or Rx-identifier. The same identifier cannot be used as Tx-*and* Rx-identifier.
Optionally the bytes, assigned to a Tx-Identifier, can be transmitted cyclically.

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 Diagnosis

The status of the LED displays and the DP-slave diagnosis can be used for diagnosis. The module supports five module-specific diagnostic bytes. The diagnosis will be described in more detail on page 15.

2.5 Parameter Telegram (CAN Bit Rate)

In addition to the seven standard bytes of the configuration, the CAN-DP/2 module supports eight module-specific bytes. Here, the DP master can e.g. change 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 have not yet been implemented.

2.7 PROFIBUS-DP Profiles

The PROFIBUS-DP profiles are not being supported yet.

2.8 More Addressable CAN Identifiers in Page Mode

The Page Mode offers the chance to address more CAN identifiers than can be stored in one PROFIBUS telegram (that means more than 48).

Because of the additional protocol expenditure the handling of the Page Mode is slightly more complicated than the standard operation of the gateway. The data exchange between PROFIBUS and CAN requires two cycles instead of one PLC cycle.

3. Implementing and Diagnosis

3.1 Prerequisites for Implementation

This chapter describes the implementation of the CAN-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 CAN-DP/2 module absolutely first with the PLC via the SIMATIC-Manager as described in chapter: "5. Configuration with the SIMATIC Manager". Only after the configuration is carried out, the CAN-DP/2 module can be identified as CAN device!

3.2 Implementation

3.2.1 Strategy

Please make the following steps to implement the module:

1	Install and wire the CAN-DP/2 module (power supply, CAN bus, see hardware manual).
2	Set the PROFIBUS address of the module by means of the coding switch.
3	Connect the PROFIBUS connector to the PROFIBUS interface of the CAN-DP/2 module.
4	Configure the settings of the CAN-DP/2 module in the PLC via the SIMATIC manager
5	Switch on the power supply for the CAN-DP/2. Now the module has to run. The CAN-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 CAN-DP/2 module starts automatically. It does not have its own mains switch.
During start-up LEDs “E” (PROFIBUS LED) and “D” (data exchange LED) flash. The PROFIBUS address set via the coding switches is read in.
The module receives projection data from the DP master and evaluates the specifications in them. If the projection complies with the structure, the CAN-DP/2 module starts the data transfer.

3.2.3 Data Transfer

If the module is configured, the data transfer starts automatically after start-up: If the PLC master changes transmission data of an identifier, the data is transmitted from the CAN-DP/2 module to the CAN bus. When the CAN-DP/2 module receives data, it provides these to the PLC master.
The configuration is described in chapter 5 ‘Configuration via the SIMATIC-Manager’ from page 24.

3.3 Diagnosis via LED Display

The function of LEDs has been defined by the firmware. In normal operation the LEDs D, P and C are never switched off, i.e. they either flash or are permanently on.

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

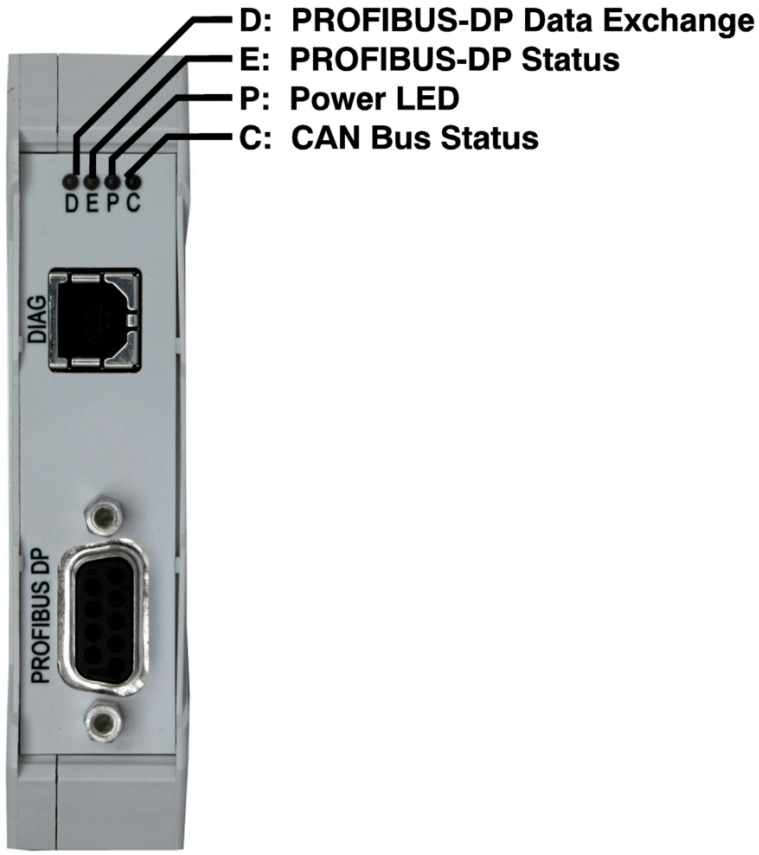

Fig. 3.3.1: Position of the LEDs

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. Diagnosis via SIMATIC-Manager or system function SFC13 (DPNRM_DG) (see chap. 3.4)
		4x short flashes	waiting for configuration telegram	configuration telegram is faulty. Diagnosis via SIMATIC-Manager or system function SFC13 (DPNRM_DG) (see chap. 3.4)
		on	malfunction	internal error
P (green)	Power LED	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 3.3.1: LED status

3.4 Slave Diagnosis

In addition to the six diagnostic bytes predefined in norm DIN EN 19245, part 3, the module supports three further module-specific diagnostic bytes.

The slave diagnosis 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 3.4.1: Function component for requesting the slave diagnosis

3.4.1 Diagnostic Bytes 0...5

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

The following designations will be used for this:

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

Table 3.4.2: Diagnostic bytes 0...5

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 CAN-DP/2? - bus connector correctly wired? - power supply available at CAN-DP/2? - power off/power on executed at CAN-DP/2 in order to read in DP address?
1	DP slave is not yet ready for data exchange	- wait until the CAN-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 CAN-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 9 and 10
7	DP slave has already been set by another master.	- this bit is always ‘1’, if you, e.g., just access the CAN-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 3.4.3: 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 according 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 diagnosis message applies. The DP slave cannot operate until the error has been removed (static diagnosis message).
2	This bit is always ‘1’.
3	The response monitoring for the CAN-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 3.4.4: Bits of station status 2

3.4.1.3 Station Status 3

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

3.4.1.4 Diagnostic Byte 3: 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 4 and 5: Manufacturer Identification

The manufacturer identification has been coded into two bytes. For the CAN-DP/2 module the designation $\mathbf{04A4_h}$ is returned.

3.4.2 External (Module-Specific) Diagnostic Bytes

The CAN-DP/2 module supports diagnostic bytes 6 to 10 for module-specific diagnosis messages.

Diagnostic bytes	Meaning
0...5	defined in the PROFIBUS specification (see previous chapter)
6	length specification for module-specific diagnosis information (here always 5)
7	header byte: bits 0...5 contain the block length including header (here always 4)
8	- DP service (SAP) which led to error (bByte 8 = $3D_h$, $3E_h$), or - bus state, if the flag *CAN-Diagnosis* = “yes” in parameterization: values of byte 8: 00_h OK 40_h WARN 80_h ERROR_PASSIVE $C0_h$ BUS_OFF
9	depending on status of byte 8: byte 8 = $3D_h$ setting (SAP61) faulty, byte 9 contains the number of the faulty setting byte byte 8 = $3E_h$ configuration (SAP62) faulty, byte 9 contains the number of the faulty PROFIBUS module (= address of the simulated PLC module) byte 8 = 00_h, 40_h, 80_h or C0h (bus state) byte 9 contains the IRQ_LOST_Counter of the built-in CAN driver of the CAN-DP/2 The IRQ_LOST_Counter counts the lost messages of the CAN controller. This counter is set by an error output of the CAN controller. It shows the number of lost CAN frames (receive or transmit messages).

Diagnostic bytes	Meaning
10	depending on status of byte 8: byte 8 = $3D_h$ setting (SAP61) faulty, byte 10 shows the correct values byte 8 = $3E_h$ configuration (SAP62) faulty 1 wrong I/O type: "out- input" or "blank") correct: "input" or "output" 2 wrong unit, such as "words" correct: unit = "byte" 3 wrong length correct: length = 1-8 or 16 4 only one byte has been specified for identifier 5 format specification is missing 6 wrong identifier byte 8 = 00_h, 40_h, 80_h or C0h (bus state) byte 10 contains the MSG_LOST_Counter of the built-in CAN-driver of the CAN-DP/2. The MSG_LOST_Counter counts the lost messages of the FIFOs. This counter is increased, if messages are lost because of a FIFO overrun (FIFO full).

Table 3.4.5: Module-specific status messages

4. GSD File

Below, the GSD file (Device Master Data) of the CAN-DP/2 module has been printed. The specification printed here are for orientation. Decisive is the data contained in the GSD file **CDPS04A4.GSD**, included in the product package.

```
;=============================================================================================
; (c) esd electronic system design GmbH Hannover
;
; PROFIBUS-DP Geraetestammdatei
; Version: 1.30
;
; Autor: Olaf Kruse
; Erstellungsdatum:         V1.0   30.04.1999  ok
; Aenderungen:              V1.01  03.08.1999  ok     baudrate 6 MBaud, MaxTsdr-times
;                           V1.02  11.08.1999  ok     baudrate 12 Mbaud, Min_Slave_Intervall,
;                                                     Max_Module, Max_Input_Len, Max_Output_Len, Max_Data_len
;                           V1.03  30.09.1999  ok     Min_Slave_Intervall = 20 (2msec)
;                           V1.04  02.11.1999  ok     MaxTsdr_45.45 = 60, MaxTsdr_1.5M = 150
;                           V1.05  20.12.1999  ok     user-parameter-data:
;                                                     byte 13 = wakeup-time ( 0: off; 0xff: not relevant )
;                                                     byte 14,15 = sync-time ( 0: off; 0xffff: not relevant )
;                           V1.06  10.04.2000  uh     menu structure for parameter
;                           V1.07  26.02.2001  uh     Min_Slave_Intervall back to 4 msec
;                           V1.10  22.10.2003  uh     Changed for new CAN-DP
;                           V1.20  02.03.2009  uh     Diagnosis and Data counter added
;                           V1.30  04.01.2013  uh     Changed for new CAN-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            = 1             ; (M ab GSD_Revision 1) (Unsigned8)
Vendor_Name             = "esd"         ; (M) Herstellername (Visible-String 32)
Model_Name              = "CAN-DP/2"    ; (M) Herstellerbezeichnung des DP-Geraetes (Visible-String 32)
Revision                = "V1.0"        ; (M) Ausgabestand des DP-Geraetes (Visible-String 32)
Revision_Number         = 1             ; (M ab GSD_Revision 1) (Unsigned8 (1 bis 63)) (1234)
Ident_Number            = 1188          ; (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        = "V1.00"       ; (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
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              = 48            ; (M falls modulare Station) Hoechstanzahl der Module einer Modularen Station
Max_Input_Len           = 244           ; (M falls modulare Station) Hoechstlaenge der Eingangsdaten einer Modularen Station
Max_Output_Len          = 244           ; (M falls modulare Station) Hoechstlaenge der Ausgangsdaten einer Modularen Station
Max_Data_Len            = 300           ; (O nur falls modulare Station) Groesste Summe der Ein- und Ausgangsdaten einer Modularen Station in Bytes
Max_Diag_Data_Len       = 16            ; max. 16 Byte Diagnosedaten
Modul_Offset            = 0             ; (D ab GSD_Revision 1) erste Steckplatznummer
Max_User_Prm_Data_Len   = 9
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 "RTR-Frames"
Bit(4) 0 0-1
Prm_Text_Ref=3
EndExtUserPrmData
ExtUserPrmData=4 "CANopen-Slave"
Bit(3) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
ExtUserPrmData=5 "CANopen-Master"
Bit(2) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
ExtUserPrmData=6 "Start-Frame"
Bit(1) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
ExtUserPrmData=7 "Page-Mode"
Bit(0) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
ExtUserPrmData=8 "ModuleID"
Unsigned8 1 1-127
EndExtUserPrmData
ExtUserPrmData=9 "WakeUp Time (0=Off, 255=Default)"
Unsigned8 255 0-255
EndExtUserPrmData
ExtUserPrmData=10 "Sync Time (0=Off, 65535=Default)"
Unsigned16 65535 0-65535
EndExtUserPrmData
ExtUserPrmData=11 "CAN-Diagnosis"
Bit(0) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
ExtUserPrmData=12 "Rx-Counter"
Bit(1) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
```

```
ExtUserPrmData=13 "Tx-Counter"
Bit(2) 0 0-1
Prm_Text_Ref=2
EndExtUserPrmData
Ext_User_Prm_Data_Const(0)=0x00,0x06,0x00,0x00,0x00,0x00,0xff,0xff,0xff
Ext_User_Prm_Data_Ref(1)=1
Ext_User_Prm_Data_Ref(2)=2
Ext_User_Prm_Data_Ref(2)=3
Ext_User_Prm_Data_Ref(2)=4
Ext_User_Prm_Data_Ref(2)=5
Ext_User_Prm_Data_Ref(2)=6
Ext_User_Prm_Data_Ref(2)=7
Ext_User_Prm_Data_Ref(3)=8
Ext_User_Prm_Data_Ref(4)=11
Ext_User_Prm_Data_Ref(4)=12
Ext_User_Prm_Data_Ref(4)=13
Ext_User_Prm_Data_Ref(6)=9
Ext_User_Prm_Data_Ref(7)=10
Slave_Family = 9@CAN@V01
OrderNumber                         = "C.2907.02"
```

5. Configuration via SIMATIC Manager

5.1 Course of Configuration

The CAN-DP/2 module is configured via the PROFIBUS.

Note:
Without correct configuration via the SIMATIC manager the CAN-DP/2 module and the CAN participants connected do not operate together and operation of the CAN participants connected can be disturbed.
In particular the CAN-Bitrate configured in the CAN-DP/2-module and the module-ID (at CANopen) must match the settings of the CAN participants connected!
If problems should occur, further information can be obtained with the diagnosis as described in the chapters "3.3 Diagnosis via LED Display" and "3.4 Slave Diagnosis".

Please follow the steps below to configure the CAN-DP/2 module:

1. **Select CAN-DP/2**
   Select menu *Hardware Catalogue* and there *Additional Field Devices* and *Other*. Select *GSD CAN-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 configuration settings by means of the parameter telegram as described in chapter 5.1.2 on page 26.

4. **Assignment of the Slots of the DP-slaves**
   Assign the slots as described in chapter 5.1.3 on page 30.

5. **Configuration of the Slots (PLC-Address)**
   Configure the slots as described in chapter 5.1.4 on page 31.

6. **Save settings on hard disk**
   Save the settings as described in chapter 5.1.5 on page 31.

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!

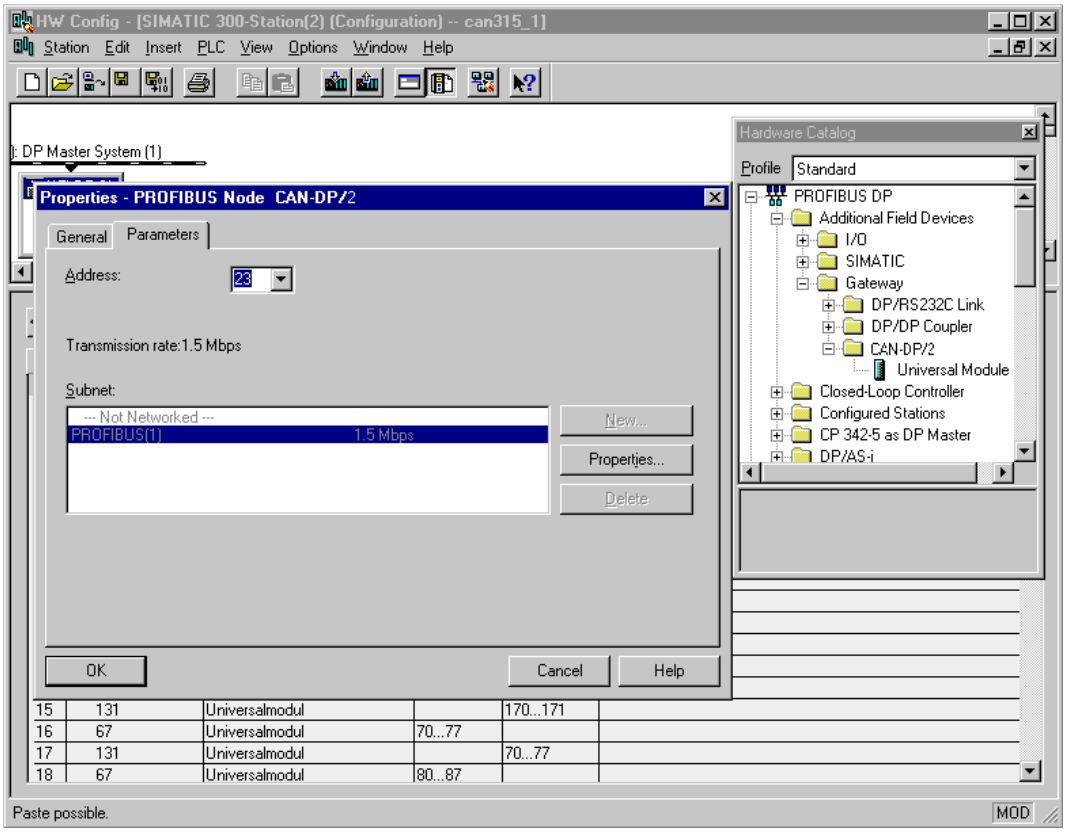

Fig. 5.1.2: Setting the PROFIBUS address of the CAN-DP/2

5.1.2 Parameter Telegram

In the configuration window the module ‘DP slave’ is now automatically added. If you desire another CAN bit rate than the standard setting of 125 Kbit/s, you can change it by means of the parameter telegram.

The module-specific bytes of the parameter telegram can be changed in the Properties window which opens, if the header of the DP-slave window is double clicked (here line ‘`(17) CAN-DP/2`’).

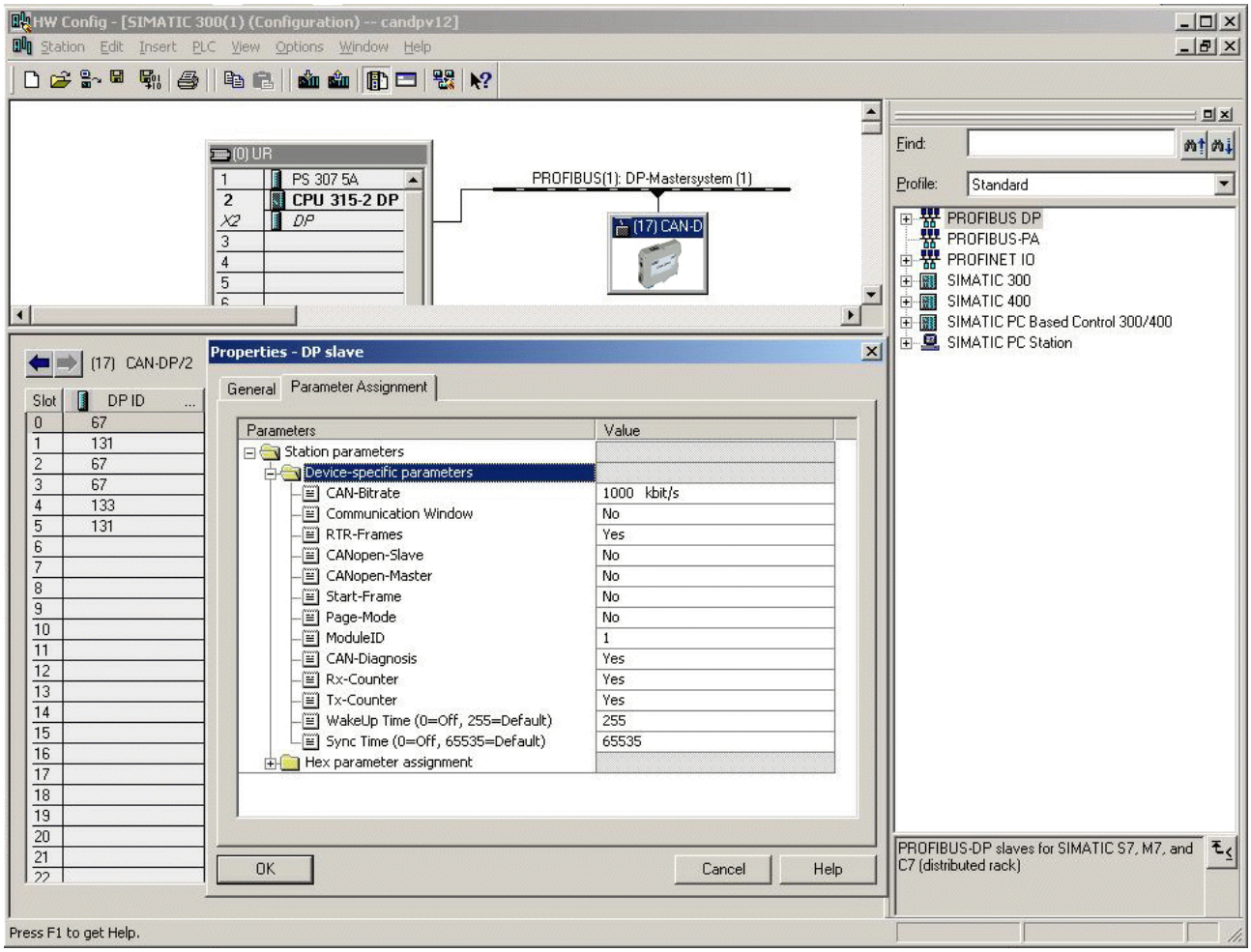

Fig. 5.1.2: Setting the parameters in the DP-slave properties window

Note:
By means of selection point *Hex-Parameter* the parameters can be specified by means of entering hexadecimal values, as in older software versions. More comfortable, however, is of course the specification in the format shown above. Here, the parameters can be configured ‘directly’. Therefore, in the following descriptions the configuration by means of hexadecimal values will not be considered.

Description of Parameters:

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

Bit rate **[kbit/s]**
1000
666.6
500
333.3
250
166
125
100
66.6
50
33.3
20
12.5
10

Table 5.2.1: Setting the bit rate in 14 levels

Communication Window: (CW) This parameter activates the Communication Window. It is described in detail on page 37.

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 has expired, a start frame is transmitted, if the gateway is a master (autostart).

Page-Mode: (PM) Activate Page-Mode.

Permissible combinations:

CW	NR	CS	CM	AS	PM	Meaning
x	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: put out TxId, transmit RTR-frames on RxId
x	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: put out TxId
x	no	no	yes	no	no	- after wake-up time, put out TxId - transmit RTR-frames on RxId
x	yes	no	yes	no	no	- after wake-up time, put out TxId
x	no	no	yes	yes	no	- after wake-up time start frame, put out TxId, transmit RTR-frames on RxId
x	yes	no	yes	yes	no	- after wake-up time start frame, put out TxId
x	yes/no	no	no	yes/no	no	- the CANopen settings like wake-up time, RTR, start-frame and sync-time are ignored

Table 5.1.2: Example for permissible settings

Module-ID: Module-ID of the Gateway as CANopen slave.
The *Module-ID* under which the gateway is addressed is set via this byte, if the gateway has been configured as CANopen slave.

Value range: 1 ... 127 (decimal)

CAN-Diagnosis: If this flag is set to "yes", diagnosis messages are sent in case of an error on the CAN bus (see page 19).

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 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 to the CAN.

The *WakeUp Time* specified here, overwrites the value of *WakeUp Time* stored previously in the CAN-DP/2 gateway, if another value than '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 start 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]	Explanations
WakeUp Time	0	WakeUp-Time function off
	1...254	WakeUp Time in seconds
	255	Use current value from gateway (default)

Table 5.1.3: Function of parameter *WakeUp Time*

SYNC Time:

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

The specified cycle is specified in milliseconds.

SYNC Time is specified as a decimal value, here.

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

Table 5.1.4: Function of parameter *SYNC Time*

Attention!
SYNC Time can be set in two different ways:
1. As described above.
2. Via bytes 4 and 5 of the Communication Window (refer to page 43).

Both specifications are equal. That means that the last specification is valid!

5.1.3 Assigning the Slots of the DP Slave

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

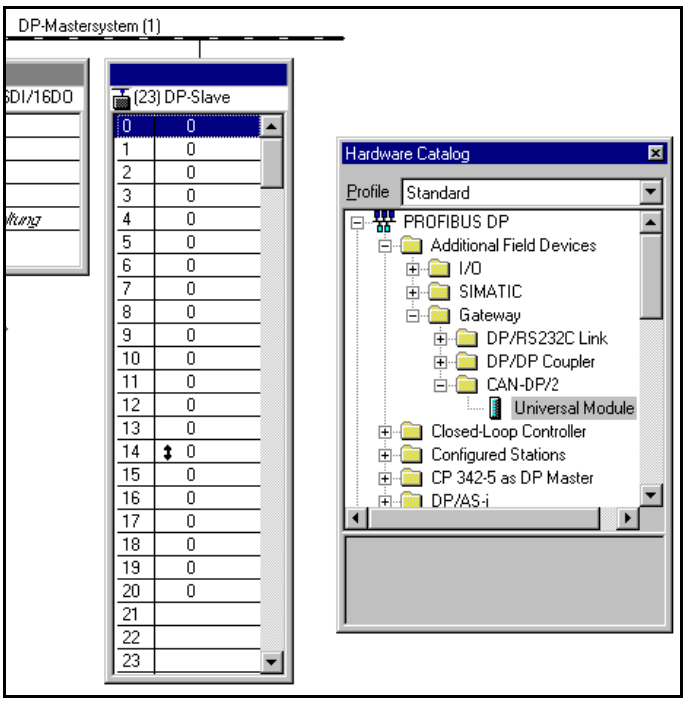

Fig. 5.1.3: Setting the parameters

5.1.4 Configuration of Slots

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
PLC address:	172 decimal
Length:	6
Unit:	byte
Consistent over:	whole length
Identifier:	0289 hexadecimal
Form byte:	B8 hexadecimal

Data direction:	output
PLC address:	172 decimal
Length:	6
Unit:	byte
Consistent over:	whole length
Identifier:	0309 hexadecimal
Form byte:	B8 hexadecimal
Cyclic transmission:	no

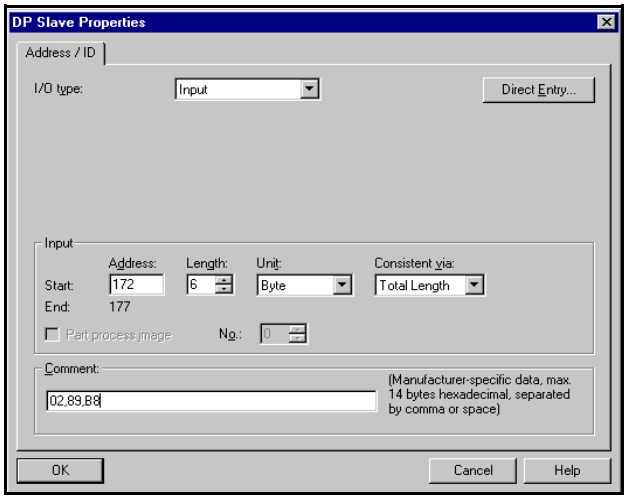

Fig. 5.1.4: Example: Configuration of input data

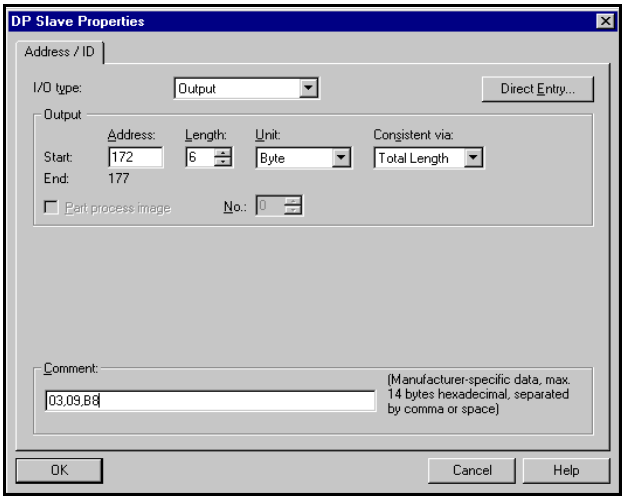

Fig. 5.1.5: Example: Configuration of output data

Achtung!
In order to guarantee that the module works perfectly, at least one output (any unit) has to be configured always. The PROFIBUS controller VPC3 does not trigger an interrupt, if no output is defined! If no CAN is to be assigned 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.

5.1.5 Save Settings to Hard Disk

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.*

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

I/O-Type Depending on the data direction, 'input' or 'output' has to be selected in this field. Other properties are not permissible.

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

Length, ***Unit*** By means of these fields the number of data bytes is specified.

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 this field shows whether the data is to be transmitted as individual unit (bytes, words, etc.) or as complete packet (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 over 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	

5.2.1 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 for example 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.

5.2.2 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).

Background: Messages which are longer than 1 byte are normally transmitted on 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 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 byte, 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 *form*	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**
hexa-decimal	8				E			
action	begin swap	end swap	un-changed	un-changed	begin swap	swap	swap	end swap

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

5.2.3 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 CAN-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 cyclical 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 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: $03E8_h$)		-

5.3 The Communication Window

5.3.1 Introduction

If the connected CANopen modules are addressed as described in chapter ‘5.1 Course of Configuration’, each CAN identifier needs its own PLC address. The Communication Window has the advantage that individual PLC addresses for different Tx-identifiers and different Rx-identifiers can be used. 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 it is recommendable to use the Communication Window for non-time-critical accesses such as writing the SDOs in CANopen nets after starting up the device.

The data length must always be 16 bytes in the configuration!

The identifier to be used is always ‘FFEF’ hexadecimal!

The Communication Window will be described in detail on the following pages.

5.3.2 Configuring the Communication Window

The Communication Window is configured via PROFIBUS. An entry for each the transmission and reception of data via the Communication Window is required. More than these two properties are not accepted by the firmware.

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 over the whole length has always to be specified! A shared PLC address or different PLC addresses are permissible for input and output direction.

Data direction:	input
PLC address:	any (example: 30)
Length:	16
Unit:	byte
Consistent over:	whole length!
Identifier:	FFEF hexadecimal
Form byte:	00 hexadecimal

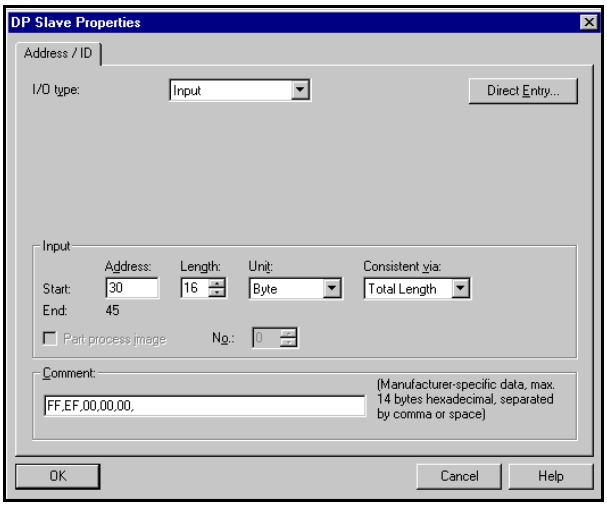

Fig. 5.3.1: Configuring the input path of the Communication Window

Data direction:	output
PLC address:	any (example: 30)
Length:	16
Unit:	byte
Consistent over:	whole length!
Identifier:	FFEF hexadecimal
Form byte:	00 hexadecimal

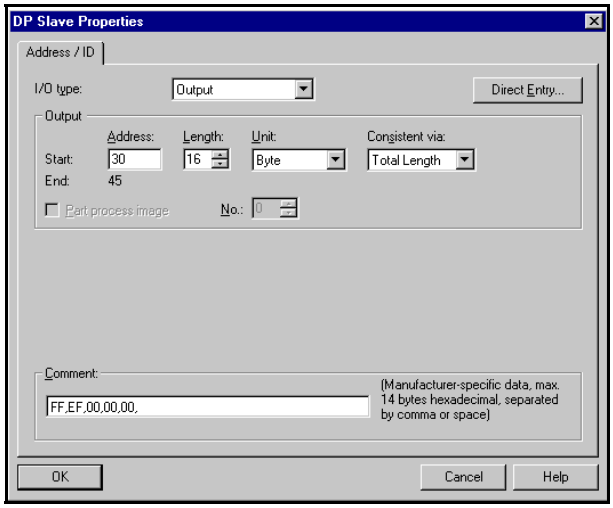

Fig. 5.3.2: Configuring the output path of the Communication Window

5.3.3 Format of Communication Window

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

5.3.3.1 Write Bytes of the Communication Window

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

Bytes of Communication Window	Contents
0 1	high byte of CAN identifier (identifier bits [15] 10...8) low byte of CAN identifier (identifier bits 7...0)
 2 3	with 11-bit CAN identifier byte 2 and 3 always ‘0’ with 29-bit CAN identifier byte 2: identifier bits 28...24 byte 3: identifier bits 23...16
4 5 6 7 8 9 10 11	data byte 0 data byte 1 data byte 2 data byte 3 data byte 4 data byte 5 data byte 6 data byte 7
12	data length for transmission jobs (Tx)
13	PLC loop counter (has to be incremented in pulse with OB1 in order to tell the gateway the OB1 cycle)
14	sub command (always set to ‘0’)
15	command (description refer page 41)

Table 5.4.1: Write bytes of the Communication Window

5.3.3.2 Read Bytes of the Communication Window

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

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

Table 5.4.2: 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’, 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 5.4.3 Commands of Communication Window

Attention!
A command is only completely processed, if, when reading the Communication Window, byte 13 of the CAN-DP/2-module provides the value of the PLC-loop counter which was specified during the command call.
Before the following command is called, it is therefore advisable to check byte 13 first!

Explanations to the commands:

Command 1: Send data

In order to send data via the Communication Window the CAN identifier has to be specified in bytes 0 and 1 (or 0...3 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 CAN-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 40. The Rx-data is received asynchronously to the PLC-cycle. Until valid data has been received you will be returned the value 'EEEE'$_h$ 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 13.

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

The command is transmitted as shown in the table 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 38).

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 (refer to page 26)
2. Via byte 4 and 5 of the Communication window (refer to page 38)

These specifications are equal. That means that the last specification is valid!

5.3.4 Examples on the Communication Window

5.3.4.1 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 CAN-DP/2-gateway (see page 27)
Communication Window: yes

1.2 Definition of the 16 input and output bytes of the Communication Window (see page 38) 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
Consistent over:	entire length!	Consistent over:	entire length!
Identifier:	FFEF hexadecimal (always)	Identifier:	FFEF hexadecimal (always)
Form byte:	00 hexadecimal	Form byte:	00 hexadecimal

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

	PLC-Cycle (Pseudo Code):
...	...
1	Read Byte 13 (returned loop counter) of '**Read** Bytes of Communication-Windows' (refer to page 40)
2	Compare Byte 13 of the '**Read** Bytes of Communication-Windows' with PLC-loop counter byte 13 of the '**Write** Bytes of Communication-Windows'(refer to page 39), if unequal go to 6., if equal go to 3.
3	Increase PLC-loop counter (Byte 13) of '**Write** Bytes of Communication-Window' (refer to page 39)
4	Evaluation of '**Read** Bytes of Communication-Windows' (refer to page 40), 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 39) 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	high byte of CAN-identifier (identifier bit [15] 10...8) low byte of CAN-identifier (identifier bit 7...0)	**00** **12**
2 3	bytes 2 and 3 always '0' for 11-bit identifier	**00** **00**
4 5 6 7 8 9 10 11	data byte 0 data byte 1 data byte 2 data byte 3 data byte 4 data byte 5 data byte 6 data byte 7	**00** **01** **02** **03** **04** **05** **06** **07**
12	data length for transmission commands	**08**
13	PLC-loop counter	**8-bit counter**
14	sub-command (always set to '0')	**00**
15	command 'transmit data'	**01**

The data bytes 00, 01, 02, 03, 04, 05, 07 are transmitted on Tx-identifier 0012_h.

In order to acknowledge the execution of the command a read access to byte 13 of the Communication Window should follow. It has to have the same value of the PLC-loop counter as when the command was called.

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

In order to acknowledge the execution of the command a read access of byte 13 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	high byte of CAN-identifier (identifier bit [15] 10...8) low byte of CAN-identifier (identifier bit 7...0)	**01** **23**
2 3	bytes 2 and 3 always ‘0’ for 11-bit identifier	**00** **00**
4 5 6 7 8 9 10 11	data byte 0 data byte 1 data byte 2 data byte 3 data byte 4 data byte 5 data byte 6 data byte 7	**00** **00** **00** **00** **00** **00** **00** **00**
12	data length for transmission commands (Tx)	**00**
13	PLC-loop counter	**8-bit counter + n**
14	sub-command (always set to ‘0’)	**00**
15	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	high byte of CAN-identifier (identifier bit [15] 10...8) low byte of CAN-identifier (identifier bit 7...0)	**01** **23**
2 3	bytes 2 and 3 always ‘0’ for 11-bit identifier	**00** **00**
4 5 6 7 8 9 10 11	received data byte 0 received data byte 1 received data byte 2 received data byte 3 received data byte 4 received data byte 5 received data byte 6 received data byte 7	**AA** **BB** **CC** **DD** **EE** **FF** **00** **11**
12	data length	**08**
13	PLC-loop counter	**8-bit counter + n**
14	returned sub-command (without significance)	**00**
15	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	high byte of CAN-identifier (identifier bit [15] 10...8) low byte of CAN-identifier (identifier bit 7...0)	**01** **23**
2 3	bytes 2 and 3 always ‘0’ for 11-bit identifiers	**00** **00**
4 5 6 7 8 9 10 11	data byte 0 data byte 1 data byte 2 data byte 3 data byte 4 data byte 5 data byte 6 data byte 7	**00** **00** **00** **00** **00** **00** **00** **00**
12	data length for transmission commands (Tx)	**00**
13	PLC-loop counter	**8-bit counter + m**
14	sub-command (always set to ‘0’)	**00**
15	command ‘Disable Rx-Identifier’	**05**

6. Page Mode

Note:
Page Mode can only be used, if the configuration tool Siemens SIMATIC Manager for S7 is used!

6.1 Properties

The Page Mode offers the chance to address more CAN identifiers than can be stored in a PROFIBUS telegram (that means more than 48). The number of possible identifiers is only limited by the free memory available on the PLC and the CAN gateway.
By means of the Communication Window, too, more than 48 identifiers can be transmitted. You can only transmit one CAN frame each per PLC cycle, however, via the Communication Window, therefore it is generally more suitable for infrequent accesses, such as one-time configurations.

Because of the additional protocol expenditure the handling of the Page Mode is slightly more complicated than the standard operation of the gateway. The data exchange between PROFIBUS and CAN requires two cycles instead of one PLC cycle, because of the required handshake.

In order to simplify the handling of the Page Mode, function blocks and data blocks which control the Page Mode are contained in the package.

6.2 Activation

Before you activate the Page Mode you have to integrate the according functional and data blocks into your PLC program. Please read the following chapters carefully to get an insight into the mode of operation and be able to use the contained functional and data blocks according to your demands.

The Page Mode is activated via the SIMATIC manager (SIEMENS PLC, S7).

6.3 Communication Window in Page Mode

When specifying *Communication Window* the Communication Window can be activated in the DP-salve properties window while configuring the gateway (see page 27).

The Communication Window is set up and handled like in normal operation (see page 37). The Communication Window must be defined in the last segment, however.

Note:
Using the Communication Windows (*CW*) is only useful to configure the connected CAN-devices. If the connected CAN-devices have been configured, the normal page mode (*PM*) is to be preferred.

6.4 Mode of Operation

6.4.1 Overview

In order to provide more CAN identifiers than can be stored in a PROFIBUS telegram, a protocol-controlled data exchange between PLC and gateway is necessary.

For the communication so-called *pages* are defined in which the parameters and data are exchanged. On PLC side an input and an output area are reserved for the transmission of the pages.
After the system has been started a page with setup data is exchanged between PLC and gateway. In the following pages the PLC transmits the configuration of the Tx- and Rx-identifiers. These pages contain the identifier numbers used for the CAN, the number of bytes and information about the data format.

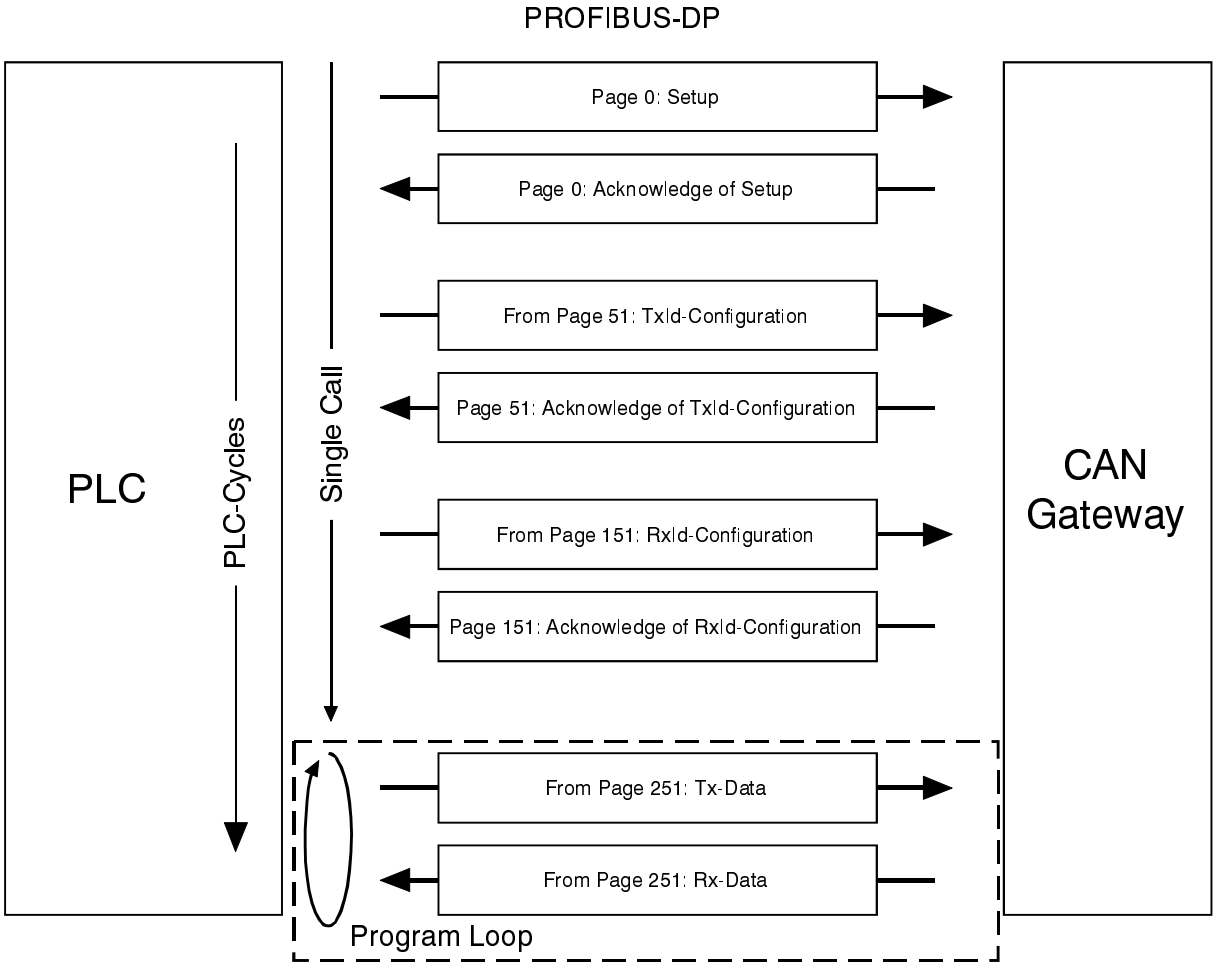

Fig. 6.4.1: Exchange of parameters and data in Page Mode (overview)

If the setup has been completed, data can be exchanged. With each PLC cycle an input and an output page is transmitted. If more identifiers have to be provided than can be stored in a page, the following identifiers will be handled in the following PLC cycles. With rising number of identifiers and depending on the length of data to be transmitted per identifier, more PLC cycles are required, therefore, to transmit all data. In order to keep the number of PLC cycles low, input and output page should be selected as large as possible.

In an example below, 127 motors are controlled by a SIEMENS SIMATIC S7-PLC. Together these motors use 127 Tx- and 127 Rx-identifiers. In this example 20 PLC-cycles are needed to supply all identifiers (20 cycles are required for 10 pages).

The product package contains function blocks (FB) and data blocks (DB) with which the transmission of the pages can be controlled. Users do not have to program the control of the pages themselves, therefore!

The FB and DB will be described from page 60.

6.4.2 Definition of PLC-Addresses

The Page Mode needs input and output addresses. The number of addresses used is limited to the top only by the PLC. The inputs need at least a page size of 32 bytes so that the setup can be made via page 0 and page 1.

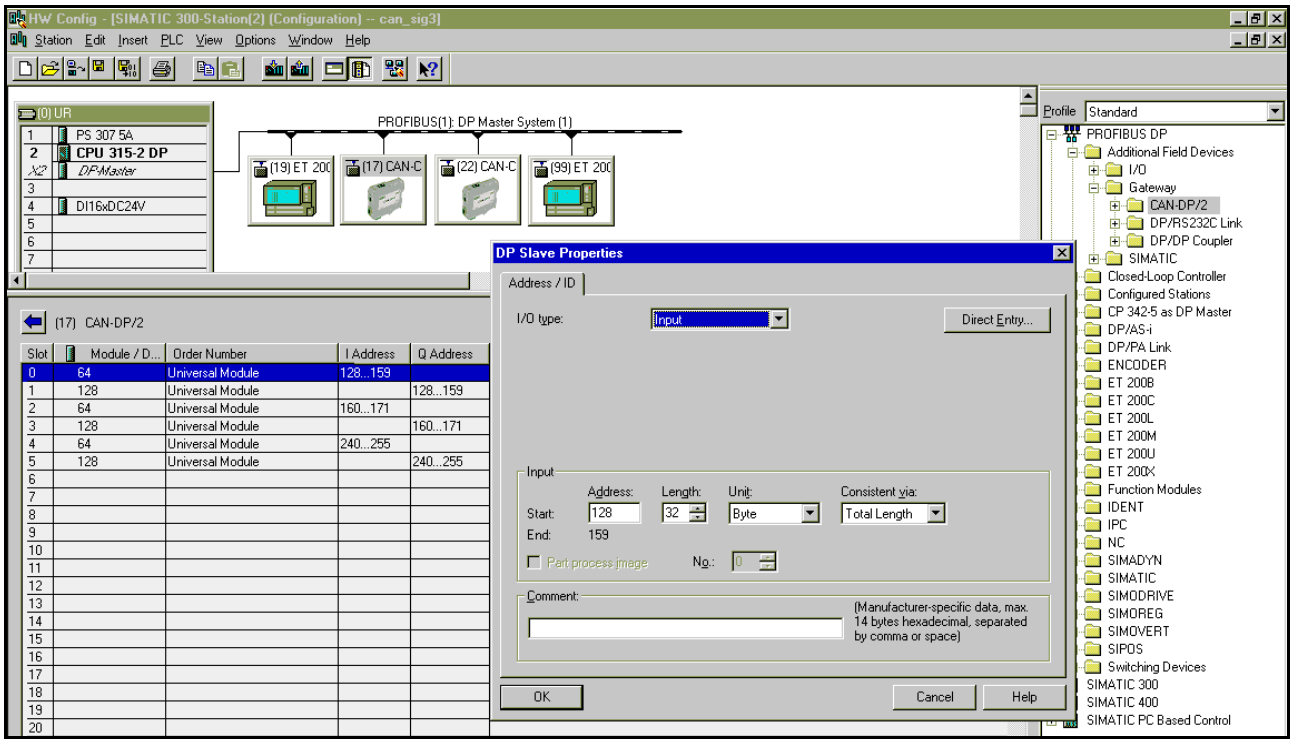

Fig. 6.4.2: Example 1: Configuring the PLC-addresses in Page Mode

Example 1:
The figure above represents the assignment of a SIMATIC-S7-300 PLC for the Page Mode. 105 bytes have been specified for the Page Mode and 16 bytes for the Communication Window. With these the S7-300 is completely occupied, because it offers a maximum of 122 bytes.

Below the specification of a PLC-slot will be called page *segment.*

In example 1 a data length of 32 bytes for each segment and the consistency for the entire 32 bytes have been set. The data length has not been chosen larger, because the S7-300 cannot transmit more than 32 bytes consistently. This, however, is absolutely necessary for the Page Mode.

Generally a segment is to be specified with 32 bytes. Given that at least 32 bytes have already been specified for the input data, it is also permissible to use any length between 0 and 32 bytes for the last segment. The length of the input data might differ from the length of the output date. It is absolutely necessary, however, that the input addresses of successive slots are sequentially, and that the output addresses of successive slots are sequentially.

Example 2:
For the output page 32 bytes have been specified at slot 0 from address 128. Slot 1 has also 32 bytes and therefore covers addresses 160...191. Slot 2 has only 18 bytes and covers addresses 192...209. A maximum size of 82 bytes results for the output page.

The following figure shoes the page in the address range of the PLC. For the application example the assignment with the Tx-configuration page (page 51) has been specified. With a size of 82 bytes 11 Tx-identifiers could be configured on one page. In the last four bytes the end identifier is specified. If more Tx-identifiers are required, Tx-pages 52, 53, etc. are transmitted afterwards.

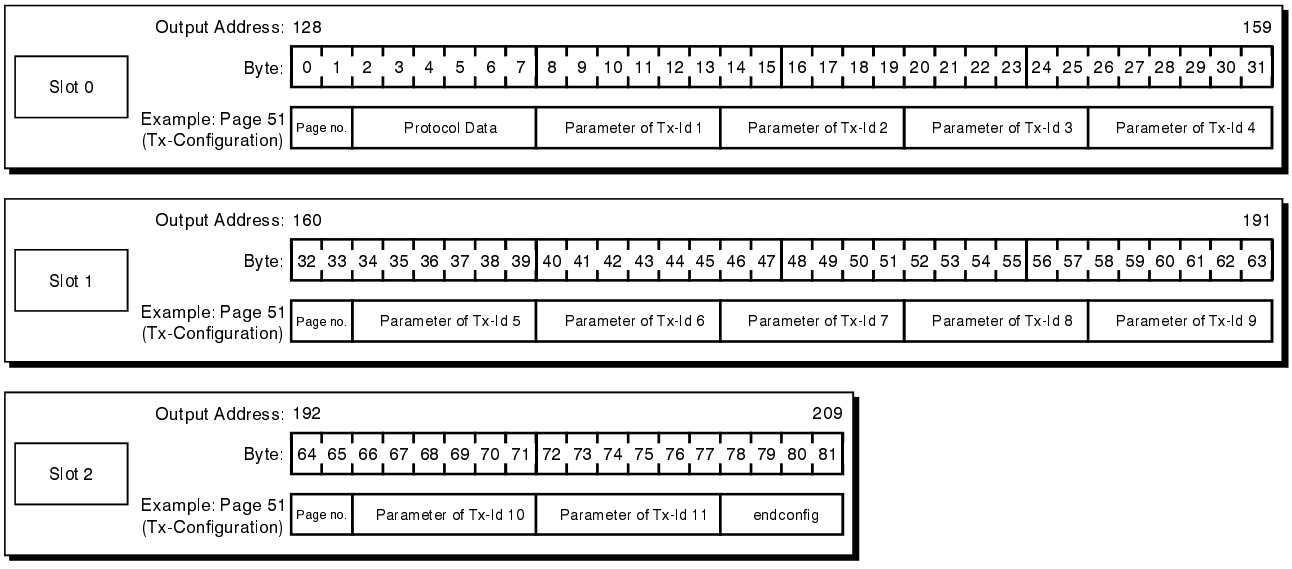

Fig. 6.4.4: Example 2: Output page with a length of 82 bytes

If the Page Mode is used with Communication Window, the Communication Window must be defined in the segment which is assigned to the **last** PLC-slot.

The following table summarizes the rules for the assignment of addresses in PLC Page Mode:

Rules for the assignment of addresses in Page Mode	
1	Define at least 32 input bytes !
2	Segment length always = 32 bytes ! Exception: last segment ≤ 32 bytes !
3	Consistency over the entire length !
4	Sequential addressing of segments of inputs and outputs !
5	Communication Window into the last two segments (if desired) !

Table 6.4.1: Rules for the assignment of addresses in Page Mode

6.4.3 Page Structure

The maximum length of the page depends on the configuration of addresses, made by the user (see page 51).

On all pages the first eight bytes contain information which is required for the protocol-controlled exchange of pages between PLC and gateway. They are followed by the ‘user data’ of the page. During configuration this data contains, e.g., the definition of identifiers, during operation the data of the identifiers.

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
Length [bytes]	2		6						depending on page no.								...	
Content	page no.		protocol data						user data								...	
Example	51		...						e.g. Tx-identifier definition								...	

Table 6.4.2: Structure of pages

The first two bytes of each segment of a page specify the page number. The page number marks the page to be transmitted and the type of page. The following table shows the page numbers, page types and the functional and data blocks which are available.

Page number	Page type	Function block	Data block	Formal operand ENABLE
0	setup page	FB2	-	0
51...150	Tx-configuration		DB94	1
151...250	Rx-configuration		DB95	
≥ 251	data exchange		output: DB96 input: DB97	

Table 6.4.3: Overview of pages

The contents of bytes 3 to 7, the ‘protocol data’ will not be referred to. Please use the function block (FB2) contained in the product package to control the transmission of pages. It contains the commands required for the protocol control.

The following chapters will describe the page types.

6.4.4 Setup via Page 0 and 1

After the system has been started, the gateway has to transmit the length of the previously configured page to the PLC. This is made by means of the so-called *page 1*.

For this the PLC has to transmit *page 0* to the gateway first. The gateway then returns the setup data in page 1.

The product package contains a function block which is responsible for the transmission and reception of pages 0 and 1 (FB2). We recommend that you use this function block. If you use function block FB2, you do not have to configure further parameters.

The setup requires some time. Therefore it is recommendable to delay the transmission of the next page for about 5 sec. It is, for example, possible to program a PLC timer which considers the delay.

6.4.5 Tx-Configuration via Pages 51...150

The Tx-identifiers are configured via pages 51 to 150 (decimal). The page structure is as follows:

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...
Length [bytes]	2		6						4				1	1	4				1	1	...	
Contents	page no.		protocol data						*TxId_value*				*form*	*length*	*TxId_value*				*form*	*length*	...	
									parameters of Tx-identifier 1						parameters of Tx-identifier 2							
Example	≥ 51		...						301				B8	6	303				B8	8	...	

Table 6.4.4: Structure of pages 51...150

Bytes 0 to 7 contain the protocol information already mentioned above (refer also to page 54).

From byte 8 in the first segment (byte 2 in the following segments) the definition of the desired Tx-identifiers is transmitted to the CAN gateway. For each Tx-identifier 6 bytes are required:

TxId_value These four bytes specify the numeric value of the Tx-identifier.

form Via this byte you can choose whether the output data is to be converted from Motorola data format of the PLC into Intel data format of the CAN network or not. Byte *form* has already been described in detail on page 32.

length Here the number of data bytes of the Tx-identifier is specified. Entries between 1 and 8 are permissible.

6.4.6 Rx-Configuration via Pages 151...250

The Rx-identifiers are configured via pages 151 to 250 (decimal). The page structure is as follows:

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...
Length [bytes]	2		6						4				1	1	4				1	1	...	
Contents	page no.		protocol data						*RxId_value*				*form*	*length*	*RxId_value*				*form*	*length*	...	
									parameters of Rx-identifier 1						parameters of Rx-identifier 2							
Example	≥ 151		...						301				B8	6	303				B8	8	...	

Table 6.4.5: Structure of pages 151...250

Bytes 0 to 7 contain the protocol information already mentioned above (refer also to page 54).

From byte 8 in the first segment (byte 2 in the following segments) the definition of the desired Rx-identifiers is transmitted to the CAN gateway. For each Rx-identifier 6 bytes are required:

RxId_value These four bytes specify the numeric value of the Rx-identifier.

form Via this byte you can choose whether the output data is to be converted from Motorola data format of the PLC into Intel data format of the CAN network or not. Byte *form* has already been described in detail on page 32.

length Here the number of data bytes of the Rx-identifier is specified. Entries between 1 and 8 are permissible.

6.4.7 Data Exchange via Pages 251...n

The user data is read and written via page 251 (decimal) and following. The maximum number of data pages is 65285.

The structure of the page for output data can differ from the page structure for input data, because the number of Rx-data can differ from the number of Tx-data.

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...
Length [bytes]	2		6						1	1...8 (here: 4)				1	1...8 (here: 6)						...	
Contents	page no.		protocol data						*force*	*Tx_user_data*				*force*	*Tx_user_data*						...	
									data of Tx-identifier 1					data of Tx-identifier 2								
Example	≥ 251		...						2	12 34 56 78 (hex)				1	BA 98 76 54 32 10 (hex)						...	

Table 6.4.6: Example of a data page for output data

Bytes 0 to 7 contain the protocol information already mentioned above (refer also to page 54).

Starting with byte 8 in the first segment, the data of the first identifier are transmitted to the gateway. The data of the next identifier follow directly, that means that only as many data bytes are transmitted each per identifier as have been defined in *length*!
In the second segment the transmission of data already starts with byte 2, because bytes 2 to 7 do not contain protocol information.

force In this byte you can specify the time when the Tx-data is to be transmitted:

force	Transmission
0	data is not put out as CAN frame
1	data is always (following each PROFIBUS telegram) put out as CAN frame
2	data is only put out as CAN frame, if data was changed
3	data is put out as CAN frame once *)
4	data is put out as CAN frame once *)

*) Change between 3 and 4 causes a direct output of data

Table 6.4.7: Specifying the cause for transmitting Tx-data

Tx_user_data Here the user data of this Tx-identifier to be transmitted are specified.

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...
Length [bytes]	2		6						1	1...8 (here: 6)						1	1...8 (here: 4)				...	
Contents	page no.		protocol data						*count_in 1*	*Rx_user_data*						*count_in 2*	*Rx_user_data*				...	
									data of Rx-identifier 1							data of Rx-identifier 2						
Example	≥ 251		...						14	11 22 33 44 55 66 (hex)						15	99 88 77 66 (hex)				...	

Table 6.4.8: Example of a data page for input data

count_in x In this byte the gateway specifies an input counter. The input counter is incremented with each Rx-frame received. It can be used by the user, for example, to program a guarding protocol.

Rx_user_data Here the received user data of this Rx-identifier are entered.

6.5 Using the Page Mode with FBs and DBs

The previous chapter has described the principal function of the Page Mode and the assignment of pages to show the functionality. The product package of the gateway includes function blocks and data blocks as source codes, which you should include in your PLC program, if you wanted to use the Page Mode.

6.5.1 Function Block FB 2: Configuration and Data Exchange

By means of function block FB2 all configurations and data transfers of the Page Mode can be executed. The types of data blocks which are used by FB2 will be shown in the following example call.

Calling FB2 (example):

```
CALL  FB     2 , DB102
  FREIGABE        :=#BIT1
  WRITE_ADDRESS   :=#WRITE_ADDRESS
  WRITE_CONFIG_DB:=#WRITE_CONFIG_DB
  WRITE_DB        :=#WRITE_DB
  READ_ADDRESS    :=#READ_ADDRESS
  READ_CONFIG_DB  :=#READ_CONFIG_DB
  READ_DB         :=#READ_DB
  RET_VALUE       :=#t016
```

Explanation of data blocks and parameters:

Data block/parameter	Function	For a detailed description refer to page
FREIGABE	Enable after setup via page 0 and 1.	61
WRITE_ADDRESS	Start address of the first output segment.	61
WRITE_CONFIG_DB	Data block to define the Tx-identifiers.	61
WRITE_DB	Data block to write the output data.	65
READ_ADDRESS	Start address of the first input segment.	62
READ_CONFIG_DB	Data block to define the Rx-identifiers.	63
READ_DB	Data block to read the input data.	67
RET_VALUE	Message about the handling of the present page.	68

Table 6.5.1: Function of data blocks used by FB2

FREIGABE Enable after basic set-up via pages 0 and 1.
The module is initialized by means of function block FB2 by bit *FREIGABE* = 0. For all other operations you have to set it to '1'. FB2 only needs an instance DB for the setup.

WRITE_ADDRESS **Start address of the first output segment.**
Via this parameter the PLC start address of the first segment of the output page is transferred to the PLC.

WRITE_CONFIG_DB **Data block to define the Tx-Identifiers.**
(DB94) In the PLC-source code, included in the product package, the *WRITE-CONFIG-DB* has been realized as data block DB94.
In *WRITE-CONFIG-DB* 6 bytes are required for each Tx-identifier to be written:

Address	Bytes 0...3	Byte 4	Byte 5	Explanation
0	*Tx-Identifier 1*	*form 1*	*length 1*	definition of Tx-Id 1
6	*Tx-Identifier 2*	*form 2*	*length 2*	definition of Tx-Id 2
12	*Tx-Identifier 3*	*form 3*	*length 3*	definition of Tx-Id 3
:	:	:	:	:
n · 6	*endconfig*	-	-	marking the end of the DB or the Tx-configuration

Table 6.5.2: Structure of the *WRITE-CONFIG-DB*

Tx-Identifier x Here the value of the Tx-identifier has to be specified.
11-Bit CAN-ID 0 ... 2047
29-Bit CAN-ID 0 ... 536870911

form x In parameter *form* you choose whether the output data is to be converted from Motorola format of the PLC into the Intel format of the CAN network or not. Byte *form* has already been described in detail on page 32.

length x This byte specifies the number of data bytes which are to be transmitted on the Tx-identifier which is defined here.

endconfig

The PLC has to be told whether another data block is required for the definition of the Tx-identifiers and when the Tx-definition will be finished.

- If another data block is required, the hexadecimal value 'DDDDDDDD' has to be specified as last Tx-identifier definition. FB2 will then continue with the handling of the following DB.
- If the last Tx-identifier has been defined, this is indicated to the FB2 by specifying the hexadecimal value 'EEEEEEEE' as last Tx-identifier definition. FB2 will then continue with the configuration of the Rx-identifiers.

The length of the data blocks differs. The required length can be determined from the number of required Tx-identifiers plus the four bytes for the end flag.

Example:
You have to define 16 Tx-identifiers via DB11.

DB11 defines Tx-Ids 1 ... 10,
therefore requires a length of (10 · 6 + 4) = 64 bytes
End flag = $DDDDDDDD_h$

DB12 defines Tx-Ids 11 ... 16,
therefore requires a length of (6 · 6 + 4) = 40 bytes
End flag = $EEEEEEEE_h$

Note:
In FB2 the bit *FREIGABE* has to be set = 1!

READ_ADDRESS

Start address of the first input segment.
Via this parameter the PLC-start address of the first segment of the input page is transferred to the PLC.

READ_CONFIG_DB
(DB95)

Data block for defining the Rx-identifiers.
In the PLC-source code included in the product package, the *READ-CONFIG-DB* has been realized as data block DB95.
In *READ-CONFIG-DB* 6 bytes are required for each Rx-identifier to be written:

Address	Bytes 0...3	Byte 4	Byte 5	Explanation
0	*Rx-Identifier 1*	*form 1*	*length 1*	definition of Rx-Id 1
6	*Rx-Identifier 2*	*form 2*	*length 2*	definition of Rx-Id 2
12	*Rx-Identifier 3*	*form 3*	*length 3*	definition of Rx-Id 3
:	:	:	:	:
n · 6	*endconfig*	-	-	marking the end of the DB or the Rx-configuration

Table 6.5.3: Structure of the *READ-CONFIG-DB*

Rx-Identifier x Here the value of the Rx-identifier has to be specified.
11-Bit CAN-ID 0 ... 2047
29-Bit CAN-ID 0 ... 536870911

form x In parameter *form* you choose, whether the input data is to be converted from Intel format of the CAN network to the Motorola format of the PLC or not. Byte *form* has already been described in detail on page 32.

length x This byte specifies the number of data bytes which are to be received by this Rx-identifier.

endconfig The PLC has to be told whether another data block is required for the definition of the Rx-identifiers and when the Rx-definition will be finished.
- If another data block is required, the hexadecimal value ‘DDDDDDDD’ has to be specified as last Rx-identifier definition. FB2 will then continue with the handling of the following DB.
- If the last Rx-identifier has been defined, this is indicated to the FB2 by specifying the hexadecimal value ‘EEEEEEEE’. FB2 will then continue with the transmission of user data.

The length of the data blocks differs. The required length can be determined from the number of Rx-identifiers required plus the four bytes for the end flag.

Example:
You have to define 19 Rx-identifiers, starting with DB26.
DB26 defines Rx-Ids 1 ... 10,
therefore requires a length of (10 · 6 + 4) = 64 bytes
End flag = $DDDDDDDD_h$
DB27 defines Rx-Ids 11 ... 19,
therefore requires a length of (9 · 6 + 4) = 58 bytes
End flag = $EEEEEEEE_h$

Note:
In FB2 the bit *FREIGABE* has to be set = 1!

WRITE_DB
(DB96)

Data block for writing the output data

The output data is stored in the data block according to identifier number (TxId1, TxId2, etc.). For each Tx-identifier the length (number of data bytes + *force*-byte) is stored in one byte, the parameter *force* is stored in another byte and then the user data is stored. The number of user data can differ from 1 to 8 bytes. The data of the following Tx-identifier always come directly after the previous one. The address from which the data of a Tx-identifier is stored has to be determined from the data of the previous Tx-identifiers, therefore.

Address	1 byte	1 byte	*n* bytes	Explanation
0	*length 1*	*force 1*	*user data 1*	user data of Tx-Id 1

Address	1 byte	1 byte	*m* bytes	Explanation
n + 2	*length 2*	*force 2*	*user data 2*	user data of Tx-Id 2

Address	1 byte	1 byte	*l* bytes	Explanation
n + 2 + *m* + 2	*length 3*	*force 3*	*user data 3*	user data of Tx-Id 3

:

Address	1 byte	-	Explanation
xxx	*enddata*	-	marking the end of the DB or the output data

Table 6.5.4: Structure of *WRITE_DB*

length x This byte specifies the number of data bytes which are to be transmitted on the Tx-identifier defined here (+1 for the *force*-byte):

lenght = (number of data bytes) + 1

force x

Via this byte you can specify the time when the data of the Tx-identifier are to be transmitted to the CAN.

force	Function
0	Data is not put out as CAN frame.
1	Data is always (following each PROFIBUS telegram) put out as CAN frames.
2	Data is only put out as CAN frame if data has changed.
3	Data is only put out as CAN frame once.
4	Data is only put out as CAN frame once.

Table 6.5.5: Meaning of parameter *force*

In order to transmit the CAN frame with the user data once, parameter *force* has to be set to value '3'. If the parameter is set to '3' again in the following cycle, the frame will not be transmitted. In order to transmit more than once *force* has to be set to the value '4' in the following cycle. Each further switch between the values triggers a transmission of frames.

userdata x

The user data between 1 to 8 bytes is specified after parameter *force.*

enddata

This parameter tells the PLC whether another data block with user data will follow, or whether this was the last user data to be transmitted.

- If another data block is required, the hexadecimal value 'DD' has to be specified for *length* following the definition of the last user data. FB2 will then continue to handle the following DB.
- If the last user data of this application has been specified, FB2 will be told by entering the hexadecimal value 'EE' in cell *length.* FB2 will then continue to transmit the user data of the first `WRITE_DB`.

Note:
Bit `FREIGABE` has to be set = 1 in FB2, if the output data is to be written!

READ_DB
(DB97)

Data block for reading the input data.
The input data is stored in the data block according to identifier number (RxId1, RxId2, etc.). For each Rx-identifier the length (number of data bytes + *count_in*-byte) is stored in one byte, the input counter *count_in* is stored in another byte and then the user data is stored. The number of user data can differ from 1 to 8 bytes. The data of the following Rx-identifier always comes directly after the previous one. The address from which the data of an Rx-identifier is stored has to be determined from the data of the previous Rx-identifiers, therefore.

Address	1 byte	1 byte	*n* bytes	Explanation
0	*length 1*	*count_in 1*	*userdata 1*	user data of Rx-Id 1

Address	1 byte	1 byte	*m* bytes	Explanation
n + 2	*length 2*	*count_in 2*	*userdata 2*	user data of Rx-Id 2

Address	1 byte	1 byte	*l* bytes	Explanation
n + 2 + m + 2	*length 3*	*count_in 3*	*userdata 3*	user data of Rx-Id 3

:

Address	1 byte	-	Explanation
xxx	*enddata*	-	marking the end of the DB or the input data

Table 6.5.6: Structure of *WRITE_DB*

length x This byte specifies the number of data bytes to be received via this Rx-identifier (+1 for the *count_in*-byte):

length = (number of data bytes) + 1

count_in x In this byte an input counter is specified by the gateway. The input counter is incremented with every Rx-frame received. It can, for example, be used to program a guarding protocol.

userdata x The user data between 1 to 8 bytes is specified after parameter *count-in.*

enddata This parameter tells the PLC whether another data block with user data will follow, or whether this was the last user data to be received.

- If another data block is required, the hexadecimal value 'DD' has to be specified for *length* following the definition of the last user data. FB2 will then continue to handle the following DB.
- If the last user data of this application has been specified, FB2 will be told by entering the hexadecimal value 'EE' in cell *length*. FB2 will then continue to transmit the user data of the first *READ_DB*.

Note:
Bit *FREIGABE* has to be set = 1 in FB2, if the input data is to be read!

RET_VALUE **Returning status of handling of current page.**
This parameter is '0', if bit *FREIGABE* = '0'. If *FREIGABE* = '1', RET_VALUE contains a number which specifies the page type which is being handled at the moment:

RET_VALUE (at *FREIGABE* = 1)	Page type currently being transmitted
0	no page transmission
1	reserved
2	Tx-configuration via pages 51...150
3	Rx-configuration via pages 151...250
4	data pages 251...n

Table 6.5.7: Return parameter *RET_VALUE*

6.6 Methodology

The following list gives a step-by-step instruction for the configuration and operation of the Page Mode.

1. Hardware Configuration

1.1 Determine PLC-address range for Page Mode (inputs/outputs), i.e.
- configure segments: n · 32 bytes + x bytes (x ≤ 32)
- successive addresses!

1.2 If required: Communication Window (at the end)

2. PLC-Program

2.1 Include FB2:

WRITE_ADDRESS:	start address of first output segment
READ_ADDRESS:	start address of first input segment

2.2 Data blocks:

WRITE_CONFIG_DB:	generate and preset (determine length of DB!)
READ_CONFIG_DB:	generate and preset (determine length of DB!)
WRITE_DB:	generate and supply with data during program
READ_DB:	generate and read data during program

3. Include More FBs

3.1 FB4: data exchange via Communication Window
3.2 FB1: initializing CANopen modules by means of a list (*INIT_LIST_DB*, *INIT_DB*)
3.3 FB3: controlling 127 uniform CANopen devices

7. 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 “Universalmodule” (see chapter: “5. Configuration with the SIMATIC Manager”).
If the “Universalmodule” 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-devided 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_or_inputbytes*
Octet 3: *Manufacturer-specific_configuration_byte*
:
:

The octets have the following meaning:

Octet 1: ***Number_of_manufacturer-specific_data***

Because the CAN-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		always 0	always 0	Length of the manufactuerer-specific data: 0011 11-bit identifier 0101 29-bit identifier 0101 Communication window			

Example Octet 1:

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

= 0x83 Output, 3 byte manufactuerer-specific data (11-bit identifier)

Octet 2: ***Number_of_output_or_input_bytes***

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

MSB LSB

<table>
<tr><td>Bit-No.:</td><td>7</td><td>6</td><td>5</td><td>4</td><td>3</td><td>2</td><td>1</td><td>0</td></tr>
<tr><td>Content:</td><td>Consistency over
0: byte or word
1: complete length</td><td>Length-format
0: byte-structure
1: word-structure</td><td colspan="6">Number of inputs/outputs
<table><tr><th>Bit
5 4 3 2 1 0</th><th>Meaning</th></tr><tr><td>0 0 0 0 0 0</td><td>1 byte, resp. 1 word</td></tr><tr><td>:</td><td>:</td></tr><tr><td>1 1 1 1 1 1</td><td>64 bytes, resp. 64 words</td></tr></table></td></tr>
</table>

Example Module 1:

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

= 0x05 6 byte data

Octet 3, 4, 5: ***Manufacturer-specific_configuration_byte***

Octet 3 and Octet 4 = CAN-identifier example: Identifier 0x0203
Octet 5 = Form byte

Example Module 1

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" 0x83, 0x05, 0x02, 0x03, 0x00
EndModule

...
```

Meaning of the entries under "Name of the module":

"Name of the module" ...	Comment to name the module		
0x83...	Module is an output (0x80) and three manufacturer-specific configuration-byte (0x03) will follow.		
0x05...	Consistency over byte, the length format is byte-structure (0x05) and 6 byte data are transferred (0x05 = 6...1).		
0x02...	Manufacturer-specific data:	0x02	
0x03...	Manufacturer-specific data:	0x03	Identifier = 0x0203 (e.g.: CANopen Rx PDO for Modul-ID 3)
0x00...	Manufacturer-specific data:	0x00	no byte-swapping, i.e. the sequence of the data will not be changed

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.

8. Application Example with Page Mode

The following summary shows the FBs and DBs required to control the CANopen devices:

```
Example for calling FB 1
--------------------------------
( Setup of CANopen module by means of a list, see below)

Network 7: motor setup
-----------
      O     #BIT15                      // domain-transfer is on ?
      O     #BIT14
      SPB   M033                        // yes --> jump
      L     #INIT_LIST_DB
      T     #t016
      AUF   DB [#t016]                  // open DB with init-list
M012: L     127                         // maximum 127 motors
      L     #MOTOR
      +     1
      T     #MOTOR
      <I                                // 127 < motor ? -> yes ==> configuration ready
      SPB   M013                        // ==> jump to the end
      +     -1                          // motor 1 starts at byte 0, motor 2 starts at byte 8, ...
      SLW   3                           // means * 8: motor-index -> byte-number
      SLW   3                           // means * 8: byte-number -> bit-address
      T     #t000
      L     0                           // means: motor not present
      L     DBW [#t000]                 // get DB-number to init this motor
      T     #INIT_DB
      ==I
      SPB   M012                        // to next motor
      L     W#16#FFFF                   // means: motor not needed to initialize
      ==I
      SPB   M012                        // to next motor
      L     #t000
      +     16
      T     #t000
      L     DBW [#t000]                 // get offset in the actual init-DB
      T     #INIT_OFFSET
      L     #MOTOR                      // motor no.
      L     W#16#600
      +I
      T     #TX_ID
      L     #MOTOR                      // motor no.
      L     W#16#580
      +I
      T     #RX_ID
      UN    #BIT15
      S     #BIT15
M033: CALL  FB     1 , DB101
       transfer        :=#BIT15
       tx_id           :=#TX_ID
       rx_id           :=#RX_ID
       write_address_cw:=#WRITE_ADDRESS_CW
       read_address_cw :=#READ_ADDRESS_CW
       init_db         :=#INIT_DB
       offset          :=#INIT_OFFSET
       ret_value       :=#t016
      U     #BIT15
      S     #BIT14
      L     W#16#0                      // means: configuration in FB1 is off
      L     #t016                       // status of setup
      ==I
      SPB   M034
      UN    #FREIGABE
      SPB   M014
      L     #INIT_LIST_DB
      T     #t008
      AUF   DB [#t008]                 // open DB with init-list
      L     #MOTOR
      +     -1                          // motor 1 starts at byte 0, motor 2 starts at byte 8, ...
      SLW   3                           // means * 8: motor-index -> byte-number
      SLW   3                           // means * 8: byte-number -> bit-address
      +     L#32
      T     #t000
```

```
      L     #t016                       // get status of setup
      T     DBW [#t000]                 // save in init-list-DB
      L     W#16#FFFF                   // means: configuration in FB1 allways runs
      L     #t016                       // status of setup
      ==I
      SPB   M014
      L     W#16#FFFE                   // means: configuration in FB1 is ready
      L     #t016                       // status of setup
      ==I
      SPB   M011
      SPA   M014
M034: U     #BIT14
      R     #BIT14
      SPA   M014
M011: U     #BIT15
      R     #BIT15
      SPA   M014
M013: UN    #BIT1
      S     #BIT1                        // say motor-configuration is ready
      L     2
      T     #RET_VALUE                   // say: configuration of Tx-ID
M014: SPA   M035
M015: NOP   0

******************************************************************************

Calling FB 2: Data exchange via Page Mode
-----------------------------------------------
(
  1. : Page  0     -> reading the lengths (absolutely necessary !!!)
  2. : Page  51 ff -> Tx-configuration ( once )
  3. : Page 151 ff -> Rx-configuration ( once )
  4. : Page 251 ff -> data exchange: output and input ( cyclically )
        ( Page 251, 252, 253, ... xyz, 251, 252, ... xyz (depending on the number of pages required)
)
)

Network 9: page-mode-output and page-mode-input
-----------
      CALL  FB     2 , DB102
       FREIGABE       :=#BIT1               // muss zuerst NULL sein (s.o.)
       WRITE_ADDRESS  :=#WRITE_ADDRESS
       WRITE_CONFIG_DB:=#WRITE_CONFIG_DB
       WRITE_DB       :=#WRITE_DB
       READ_ADDRESS   :=#READ_ADDRESS
       READ_CONFIG_DB :=#READ_CONFIG_DB
       READ_DB        :=#READ_DB
       RET_VALUE      :=#t016

******************************************************************************

Calling FB 4: give commands via Communication Window
----------------------------------------------------------------
(after setup via FB 1, because FB 1 operates via the Communication Window as well)

Network 5:
-----------
      U     M     95.0
      SPB   M401
      L     0                           // CAN-ID = 0
      T     MW     0
      T     MW     4
      T     MB    12
      T     MB    14                    // subcommand
      T     MB    15                    // command = 0:
      SPA   M499
M401: U     M     95.1                  // start-frame ready ?
      SPB   M402
      L     0                           // CAN-ID = 0 (for start-frame)
      T     MW     0
      L     W#16#100                    // CAN-data = 0x01,0x00 (start-frame)
      T     MW     4
      L     2
      T     MB    12
      L     0
      T     MB    14                    // subcommand
      L     1
      T     MB    15                    // command = 1: send frame
      SPA   M499
```

```
M402: U     M     95.2                  // sync-time ready ?
      SPB   M403
      L     0
      T     MW     0
      L     W#16#200                   // time = 512 msec
      T     MW     4
      L     0
      T     MB    12
      L     0
      T     MB    14                   // subcommand
      L     20
      T     MB    15                   // command = 20: set sync-frame-time
      SPA   M499
M403: U     M     95.3
      SPB   M404
      L     W#16#181                   // 1.PDO of motor 1
      T     MW     0
      L     0
      T     MW     4
      T     MB    12
      T     MB    14                   // subcommand
      L     4
      T     MB    15                   // command = 4: activate reception
      SPA   M499
M404: U     M     95.4
      SPB   M405
      L     W#16#18A                   // 1.PDO of motor 10
      T     MW     0
      L     0
      T     MW     4
      T     MB    12
      T     MB    14                   // subcommand
      L     4
      T     MB    15                   // command = 4: activate reception
      SPA   M499
M405: U     M     95.5
      SPB   M406
      L     W#16#183                   // 1.PDO of motor 3
      T     MW     0
      L     0
      T     MW     4
      T     MB    12
      T     MB    14                   // subcommand
      L     4
      T     MB    15                   // command = 4: activate reception
      SPA   M499
M406: U     M     95.6
      SPB   M407
      L     W#16#184                   // 1.PDO of motor 4
      T     MW     0
      L     0
      T     MW     4
      T     MB    12
      T     MB    14                   // subcommand
      L     4
      T     MB    15                   // command = 4: activate reception
      SPA   M499
M407: U     M     95.7
      SPB   M408
      L     W#16#187                   // 1.PDO of motor 7
      T     MW     0
      L     0
      T     MW     4
      T     MB    12
      T     MB    14                   // subcommand
      L     4
      T     MB    15                   // command = 4: activate reception
      SPA   M499
M408: U     M     96.0
      SPB   M499
      L     W#16#188                   // 1.PDO of motor 8
      T     MW     0
      L     0
      T     MW     4
      T     MB    12
      T     MB    14                   // subcommand
      L     4
      T     MB    15                   // command = 4: activate reception
      SPA   M499
```

```
M499: NOP   0
      U     E    125.7
      SPB   M498
      UN    E    125.7
      =     M     99.7
      =     M     99.6
      L     0
      T     MW     0
      T     MW     4                    // time = 0 => sync-frame off
      L     0
      T     MB    12
      L     0
      T     MB    14                    // subcommand
      L     20
      T     MB    15                    // command = 20: set sync-frame-time
M498: NOP   0


Network 6:
----------
      CALL  FB     4 , DB104
       FREIGABE         :=M99.6
       WRITE_ENABLE     :=M99.7
       READ_ENABLE      :=M99.7
       WRITE_CAN_ID     :=MW0
       WRITE_DATA0      :=MB4
       WRITE_DATA1      :=MB5
       WRITE_DATA2      :=MB6
       WRITE_DATA3      :=MB7
       WRITE_DATA4      :=MB8
       WRITE_DATA5      :=MB9
       WRITE_DATA6      :=MB10
       WRITE_DATA7      :=MB11
       WRITE_LEN        :=MB12
       WRITE_SUBCOMMAND:=MB14
       WRITE_COMMAND    :=MB15
       WRITE_ADDRESS    :=W#16#F0
       READ_ADDRESS     :=W#16#F0
       TRANSFER_READY   :=M99.5
       READ_CAN_ID      :=MW0
       READ_DATA0       :=MB4
       READ_DATA1       :=MB5
       READ_DATA2       :=MB6
       READ_DATA3       :=MB7
       READ_DATA4       :=MB8
       READ_DATA5       :=MB9
       READ_DATA6       :=MB10
       READ_DATA7       :=MB11
       READ_LEN         :=MB12
       READ_FIFO_COUNT :=MB14
       READ_COMMAND     :=MB15
       READ_RET_VAL     :=MW14
       WRITE_RET_VAL    :=MW16
      L     W#16#181
      L     MW     0
      >I
      SPB   M601
      L     W#16#1FF
      L     MW     0
      <I
      SPB   M601
      L     W#16#181
      -I
      SLD   4
      SLD   3
      T     MD    14
      AUF   DB    92
      L     MW     0                    // CAN-ID
      T     DBW [MD    14]
      L     MD    14
      +     L#16
      T     MD    14
      L     0                           // reserve byte 2 + 3
      T     DBW [MD    14]
      L     MD    14
      +     L#16
      T     MD    14
      L     MD     4                    // data-byte 0 - 3
      T     DBD [MD    14]
```

```
      L     MD    14
      +     L#32
      T     MD    14
      L     MD     8                    // data-byte 4 - 7
      T     DBD [MD    14]
      L     MD    14
      +     L#32
      T     MD    14
      L     MD    12                    // length, counter, fifo-counter, command
      T     DBD [MD    14]
M601: UN    M     99.6
      S     M     99.6
      U     M     99.5
      R     M     99.6

      U     M     99.5
      U     M     95.7
      S     M     96.0

      U     M     99.5
      U     M     95.6
      S     M     95.7

      U     M     99.5
      U     M     95.5
      S     M     95.6

      U     M     99.5
      U     M     95.4
      S     M     95.5

      U     M     99.5
      U     M     95.3
      S     M     95.4

      U     M     99.5
      U     M     95.2
      S     M     95.3

      U     M     99.5
      U     M     95.1
      S     M     95.2

      U     M     99.5
      U     M     95.0
      S     M     95.1

      U     M     99.5
      S     M     95.0



*****************************************************************************

Calling FB3
=======================
( in FB 3 FB 1 (motor set-up) and FB 2 (data exchange) are called )

      CALL  FB     3 , DB103
       FREIGABE :=E125.7
       KONFIG_DB:=W#16#5D               // DB93: initialize motors 1,3,4,7,8 ; 1,4,8
       DATEN_DB :=W#16#64
       RET_VALUE:=MW16
      L     MW    16
      L     2
      >=I
      =     M     99.7
                  // motors initiliazed => Communication Window can be used
```

```
DB93: CONFIG_DB = configuration data block
===========================================
Address  Name  Type Initial val.  Actual val. Comment
-------  ----  ---  -----------  ----------  ---------
    0.0  v000  WORD  W#16#5E      W#16#5E     Number of first DB with Tx-configuration
    2.0  v002  WORD  W#16#2FE     W#16#2FE    Length of DBs with Tx-configuration
    4.0  v004  WORD  W#16#0       W#16#0      Number of DBs with Tx-configuration
    6.0  v006  WORD  W#16#5F      W#16#5F     Number of first DB with Rx-configuration
    8.0  v008  WORD  W#16#2FE     W#16#2FE    Length of DBs with Rx-configuration
   10.0  v010  WORD  W#16#0       W#16#0      Number of DBs with Rx-configuration
   12.0  v012  WORD  W#16#60      W#16#60     Number of first DB with output data
   14.0  v014  WORD  W#16#3F9     W#16#3F9    Length of DBs with output data
   16.0  v016  WORD  W#16#0       W#16#0      Number of DBs with output data
   18.0  v018  WORD  W#16#61      W#16#61     Number of first DB with input data
   20.0  v020  WORD  W#16#3F9     W#16#3F9    Length of DBs with input data
   22.0  v022  WORD  W#16#0       W#16#0      Number of DBs with input data
   24.0  v024  WORD  W#16#80      W#16#80     PLC-start address of output page
   26.0  v026  WORD  W#16#80      W#16#80     PLC-start address of input page
   28.0  v028  WORD  W#16#F0      W#16#F0     PLC-start address of output-communication window
   30.0  v030  WORD  W#16#F0      W#16#F0     PLC-start address of input-communication window
   32.0  v032  WORD  W#16#62      W#16#62     Number of DB with Init list


Set-up motors 1, 3, 4, 7 and 8 ==> DB 94, DB 95, DB 96 and DB 97

DB94: TX_CONFIG = Data block with configuration of Tx-identifiers
=================================================================
Address     Name       Type    Initial val.   Actual val.     Comment
-------     ----       ---     -----------    ----------      ---------
    0.0     CAN_ID1    DWORD   DW#16#0        DW#16#301       CAN-identifier
    4.0     FORMAT1    BYTE    B#16#0         B#16#B8         format-byte
    5.0     LENGTH1    BYTE    B#16#0         B#16#6          length
    6.0     CAN_ID2    DWORD   DW#16#0        DW#16#303       CAN-identifier
            FORMAT2    BYTE    B#16#0         B#16#B8         format-byte
            LENGTH2    BYTE    B#16#0         B#16#6          length
            CAN_ID3    DWORD   DW#16#0        DW#16#304       CAN-identifier
            FORMAT3    BYTE    B#16#0         B#16#B8         format-byte
            LENGTH3    BYTE    B#16#0         B#16#6          length
            CAN_ID4    DWORD   DW#16#0        DW#16#307       CAN-identifier
            FORMAT4    BYTE    B#16#0         B#16#B8         format-byte
            LENGTH4    BYTE    B#16#0         B#16#6          length
            CAN_ID5    DWORD   DW#16#0        DW#16#308       CAN-identifier
            FORMAT5    BYTE    B#16#0         B#16#B8         format-byte
            LENGTH5    BYTE    B#16#0         B#16#6          length
   30.0     CAN_ID6    DWORD   DW#16#0        DW#16#EEEEEEEE  CAN-identifier (end flag)

            FORMAT6    BYTE    B#16#0         B#16#0          format-byte
            LENGTH6    BYTE    B#16#0         B#16#0          length
            CAN_ID7    DWORD   DW#16#0        DW#16#0         CAN-identifier
            FORMAT7    BYTE    B#16#0         B#16#0          format-byte
            LENGTH7    BYTE    B#16#0         B#16#0          length

DB95: RX_CONFIG = Data block with configuration of Rx-identifiers
=================================================================
Address     Name       Type     Initial val.  Actial val.     Comment
-------     ----       ---      -----------   ----------      ---------
    0.0     CAN_ID1    DWORD    DW#16#0       DW#16#281       CAN-identifier
    4.0     FORMAT1    BYTE     B#16#0        B#16#B8         format-byte
    5.0     LENGTH1    BYTE     B#16#0        B#16#6          length
    6.0     CAN_ID2    DWORD    DW#16#0       DW#16#283       CAN-identifier
            FORMAT2    BYTE     B#16#0        B#16#B8         format-byte
            LENGTH2    BYTE     B#16#0        B#16#6          length
            CAN_ID3    DWORD    DW#16#0       DW#16#284       CAN-identifier
            FORMAT3    BYTE     B#16#0        B#16#B8         format-byte
            LENGTH3    BYTE     B#16#0        B#16#6          length
            CAN_ID4    DWORD    DW#16#0       DW#16#287       CAN-identifier
            FORMAT4    BYTE     B#16#0        B#16#B8         format-byte
            LENGTH4    BYTE     B#16#0        B#16#6          length
            CAN_ID5    DWORD    DW#16#0       DW#16#288       CAN-identifier
            FORMAT5    BYTE     B#16#0        B#16#B8         format-byte
            LENGTH5    BYTE     B#16#0        B#16#6          length
   30.0     CAN_ID6    DWORD    DW#16#0       DW#16#EEEEEEEE  CAN-identifier

            FORMAT6    BYTE     B#16#0        B#16#0          format-byte
            LENGTH6    BYTE     B#16#0        B#16#0          length
            CAN_ID7    DWORD    DW#16#0       DW#16#0         CAN-identifier
            FORMAT7    BYTE     B#16#0        B#16#0          format-byte
            LENGTH7    BYTE     B#16#0        B#16#0          length
```

DB96: OUTPUT_DB = Data block for output data

Address	Name	Type	Initial val.	Actual val.	Comment
0.0	laenge1	BYTE	B#16#0	B#16#7	Length
1.0	force1	BYTE	B#16#0	B#16#2	Force byte
2.0	data01	BYTE	B#16#0	B#16#0	Data byte 0
3.0	data11	BYTE	B#16#0	B#16#0	Data byte 1
4.0	data21	BYTE	B#16#0	B#16#0	Data byte 2
5.0	data31	BYTE	B#16#0	B#16#0	Datenbyte 3
6.0	data41	BYTE	B#16#0	B#16#0	Data byte 4
7.0	data51	BYTE	B#16#0	B#16#0	Data byte5
8.0	laenge2	BYTE	B#16#0	B#16#7	Length
9.0	force2	BYTE	B#16#0	B#16#1	Force byte
10.0	data02	BYTE	B#16#0	B#16#0	Data byte 0
11.0	data12	BYTE	B#16#0	B#16#0	Data byte 1
	data22	BYTE	B#16#0	B#16#0	Data byte 2
	data32	BYTE	B#16#0	B#16#0	Data byte 3
	data42	BYTE	B#16#0	B#16#0	Data byte 4
	data52	BYTE	B#16#0	B#16#0	Data byte5
	laenge3	BYTE	B#16#0	B#16#7	Length
	force3	BYTE	B#16#0	B#16#4	Force byte
	data03	BYTE	B#16#0	B#16#0	Data byte 0
	data13	BYTE	B#16#0	B#16#0	Data byte 1
	data23	BYTE	B#16#0	B#16#0	Data byte 2
	data33	BYTE	B#16#0	B#16#0	Data byte 3
	data43	BYTE	B#16#0	B#16#0	Data byte 4
	data53	BYTE	B#16#0	B#16#0	Data byte5
	laenge4	BYTE	B#16#0	B#16#7	Length
	force4	BYTE	B#16#0	B#16#0	Force byte
	data04	BYTE	B#16#0	B#16#0	Data byte 0
	data14	BYTE	B#16#0	B#16#0	Data byte 1
	data24	BYTE	B#16#0	B#16#0	Data byte 2
	data34	BYTE	B#16#0	B#16#0	Data byte 3
	data44	BYTE	B#16#0	B#16#0	Data byte 4
	data54	BYTE	B#16#0	B#16#0	Data byte5
	laenge5	BYTE	B#16#0	B#16#7	Length
	force5	BYTE	B#16#0	B#16#3	Force byte
	data05	BYTE	B#16#0	B#16#0	Data byte 0
	data15	BYTE	B#16#0	B#16#0	Data byte 1
	data25	BYTE	B#16#0	B#16#0	Data byte 2
	data35	BYTE	B#16#0	B#16#0	Data byte 3
	data45	BYTE	B#16#0	B#16#0	Data byte 4
	data55	BYTE	B#16#0	B#16#0	Data byte5
40.0	laenge6	BYTE	B#16#0	B#16#EE	Length
	force6	BYTE	B#16#0	B#16#0	Force byte

DB97: INPUT_DB = Data block for input data

Address	Name	Type	Initial val.	Actual val.	Comment
0.0	laenge1	BYTE	B#16#0	B#16#7	Length
1.0	zaehler1	BYTE	B#16#0	B#16#0	Counter
2.0	data01	BYTE	B#16#0	B#16#0	Data byte 0
3.0	data11	BYTE	B#16#0	B#16#0	Data byte 1
4.0	data21	BYTE	B#16#0	B#16#0	Data byte 2
5.0	data31	BYTE	B#16#0	B#16#0	Data byte 3
6.0	data41	BYTE	B#16#0	B#16#0	Data byte 4
7.0	data51	BYTE	B#16#0	B#16#0	Data byte 5
8.0	laenge2	BYTE	B#16#0	B#16#7	Length
	zaehler2	BYTE	B#16#0	B#16#0	Counter
	data02	BYTE	B#16#0	B#16#0	Data byte 0
	data12	BYTE	B#16#0	B#16#0	Data byte 1
	data22	BYTE	B#16#0	B#16#0	Data byte 2
	data32	BYTE	B#16#0	B#16#0	Data byte 3
	data42	BYTE	B#16#0	B#16#0	Data byte 4
	data52	BYTE	B#16#0	B#16#0	Data byte 5
	laenge3	BYTE	B#16#0	B#16#7	Length
	zaehler3	BYTE	B#16#0	B#16#0	Counter
	data03	BYTE	B#16#0	B#16#0	Data byte 0
	data13	BYTE	B#16#0	B#16#0	Data byte 1
	data23	BYTE	B#16#0	B#16#0	Data byte 2
	data33	BYTE	B#16#0	B#16#0	Data byte 3
	data43	BYTE	B#16#0	B#16#0	Data byte 4
	data53	BYTE	B#16#0	B#16#0	Data byte 5
	laenge4	BYTE	B#16#0	B#16#7	Length
	zaehler4	BYTE	B#16#0	B#16#0	Counter

```
          data04            BYTE    B#16#0          B#16#0          Data byte 0
          data14            BYTE    B#16#0          B#16#0          Data byte 1
          data24            BYTE    B#16#0          B#16#0          Data byte 2
          data34            BYTE    B#16#0          B#16#0          Data byte 3
          data44            BYTE    B#16#0          B#16#0          Data byte 4
          data54            BYTE    B#16#0          B#16#0          Data byte 5
          laenge5           BYTE    B#16#0          B#16#7          Length
          zaehler5          BYTE    B#16#0          B#16#0          Counter
          data05            BYTE    B#16#0          B#16#0          Data byte 0
          data15            BYTE    B#16#0          B#16#0          Data byte 1
          data25            BYTE    B#16#0          B#16#0          Data byte 2
          data35            BYTE    B#16#0          B#16#0          Data byte 3
          data45            BYTE    B#16#0          B#16#0          Data byte 4
          data55            BYTE    B#16#0          B#16#0          Data byte 5
   40.0   laenge6           BYTE    B#16#0          B#16#EE         Length
          zaehler6          BYTE    B#16#0          B#16#0          Counter


DB98: INIT_LIST_DB = Data block with list of motors which are available / are being initialized
=============================================================================================
Address  Name          Type Initial val. Actual val. Comment
-------  ----          ---  -----------  ----------  ---------
    0.0  init_db1      WORD  W#16#63      W#16#63     Motor 1 available and initialize
    2.0  init_offset1  WORD  W#16#0       W#16#0      --> list from DB 99 data word 0
    4.0  init_status1  WORD  W#16#0       W#16#0
    6.0  reserve1      WORD  W#16#0       W#16#0
    8.0  init_db2      WORD  W#16#0       W#16#0      Motor 2 not available
   10.0  init_offset2  WORD  W#16#0       W#16#0
   12.0  init_status2  WORD  W#16#0       W#16#0
   14.0  reserve2      WORD  W#16#0       W#16#0
   16.0  init_db3      WORD  W#16#FFFF    W#16#FFFF   Motor 3 available
   18.0  init_offset3  WORD  W#16#0       W#16#0
   20.0  init_status3  WORD  W#16#0       W#16#0
   22.0  reserve3      WORD  W#16#0       W#16#0
   24.0  init_db4      WORD  W#16#63      W#16#63     Motor 4 available and initialize
   26.0  init_offset4  WORD  W#16#82      W#16#82     --> list from DB 99 data word 130
   28.0  init_status4  WORD  W#16#0       W#16#0
   30.0  reserve4      WORD  W#16#0       W#16#0
   32.0  init_db5      WORD  W#16#0       W#16#0      Motor 5 not available
   34.0  init_offset5  WORD  W#16#0       W#16#0
   36.0  init_status5  WORD  W#16#0       W#16#0
   38.0  reserve5      WORD  W#16#0       W#16#0
   40.0  init_db6      WORD  W#16#0       W#16#0      Motor 6 not available
   42.0  init_offset6  WORD  W#16#0       W#16#0
   44.0  init_status6  WORD  W#16#0       W#16#0
   46.0  reserve6      WORD  W#16#0       W#16#0
   48.0  init_db7      WORD  W#16#FFFF    W#16#FFFF   Motor 7 available
   50.0  init_offset7  WORD  W#16#0       W#16#0
   52.0  init_status7  WORD  W#16#0       W#16#0
   54.0  reserve7      WORD  W#16#0       W#16#0
   56.0  init_db8      WORD  W#16#63      W#16#63     Motor 8 available and initialize
   58.0  init_offset8  WORD  W#16#0       W#16#0      --> list from DB 99 data word 0
   60.0  init_status8  WORD  W#16#0       W#16#0
   62.0  reserve8      WORD  W#16#0       W#16#0
   64.0  init_db9      WORD  W#16#0       W#16#0      Motor 9 not available
   66.0  init_offset9  WORD  W#16#0       W#16#0
   68.0  init_status9  WORD  W#16#0       W#16#0
   70.0  reserve9      WORD  W#16#0       W#16#0
     .      .            .      .           .
     .      .            .      .           .
     .      .            .      .           .
                         (always 0 => motor xxx not available)


DB99: INIT_DB = Data block with setup list
==========================================================
Address    Name      Type    Initial val. Actual val.     Comment
-------    ----      ---     -----------  ----------      ---------
    0.0    v010      WORD    W#16#1000    W#16#1000       index
    2.0    v012      BYTE    B#16#0       B#16#0          subindex
    3.0    v013      BYTE    B#16#2       B#16#2          ccs = 2 => read (domain upload)
    4.0    v014      BYTE    B#16#0       B#16#0          length
    5.0    v015      BYTE    B#16#0       B#16#0          data 0
    6.0    v016      BYTE    B#16#0       B#16#0          data 1
    7.0    v017      BYTE    B#16#0       B#16#0          data 2
    8.0    v018      BYTE    B#16#0       B#16#0          data 3
    9.0    v019      BYTE    B#16#0       B#16#0          reserve
   10.0    v020      WORD    W#16#6040    W#16#6040       index
```

```
12.0    v022      BYTE    B#16#0        B#16#0          subindex
        v023      BYTE    B#16#1        B#16#1          ccs = 1 => write (domain download)
        v024      BYTE    B#16#2        B#16#2          length
        v025      BYTE    B#16#0        B#16#0          data 0
        v026      BYTE    B#16#6        B#16#6          data 1
        v027      BYTE    B#16#0        B#16#0          data 2
        v028      BYTE    B#16#0        B#16#0          data 3
        v029      BYTE    B#16#0        B#16#0          reserve
        v0101     WORD    W#16#6040     W#16#6040       index
        v0121     BYTE    B#16#0        B#16#0          subindex
        v0131     BYTE    B#16#1        B#16#1          ccs
        v0141     BYTE    B#16#2        B#16#2          length
        v0151     BYTE    B#16#0        B#16#0          data 0
        v0161     BYTE    B#16#7        B#16#7          data 1
        v0171     BYTE    B#16#0        B#16#0          data 2
        v0181     BYTE    B#16#0        B#16#0          data 3
        v0191     BYTE    B#16#0        B#16#0          reserve
        v0102     WORD    W#16#1002     W#16#1002       index
        v0122     BYTE    B#16#0        B#16#0          subindex
        v0132     BYTE    B#16#2        B#16#2          ccs
        v0142     BYTE    B#16#0        B#16#0          length
        v0152     BYTE    B#16#0        B#16#0          data 0
        v0162     BYTE    B#16#0        B#16#0          data 1
        v0172     BYTE    B#16#0        B#16#0          data 2
        v0182     BYTE    B#16#0        B#16#0          data 3
        v0192     BYTE    B#16#0        B#16#0          reserve
        v0103     WORD    W#16#6060     W#16#6060       index
        v0123     BYTE    B#16#0        B#16#0          subindex
        v0133     BYTE    B#16#1        B#16#1          ccs
        v0143     BYTE    B#16#1        B#16#1          length
        v0153     BYTE    B#16#6        B#16#6          data 0
        v0163     BYTE    B#16#0        B#16#0          data 1
        v0173     BYTE    B#16#0        B#16#0          data 2
        v0183     BYTE    B#16#0        B#16#0          data 3
        v0193     BYTE    B#16#0        B#16#0          reserve
        v0104     WORD    W#16#6098     W#16#6098       index
        v0124     BYTE    B#16#0        B#16#0          subindex
        v0134     BYTE    B#16#1        B#16#1          ccs
        v0144     BYTE    B#16#1        B#16#1          length
        v0154     BYTE    B#16#FF       B#16#FF         data 0
        v0164     BYTE    B#16#0        B#16#0          data 1
        v0174     BYTE    B#16#0        B#16#0          data 2
        v0184     BYTE    B#16#0        B#16#0          data 3
        v0194     BYTE    B#16#0        B#16#0          reserve
        v0105     WORD    W#16#200B     W#16#200B       index
        v0125     BYTE    B#16#0        B#16#0          subindex
        v0135     BYTE    B#16#1        B#16#1          ccs
        v0145     BYTE    B#16#4        B#16#4          length
        v0155     BYTE    B#16#0        B#16#0          data 0
        v0165     BYTE    B#16#0        B#16#0          data 1
        v0175     BYTE    B#16#0        B#16#0          data 2
        v0185     BYTE    B#16#0        B#16#0          data 3
        v0195     BYTE    B#16#0        B#16#0          reserve
        v0106     WORD    W#16#6040     W#16#6040       index
        v0126     BYTE    B#16#0        B#16#0          subindex
        v0136     BYTE    B#16#1        B#16#1          ccs
        v0146     BYTE    B#16#2        B#16#2          length
        v0156     BYTE    B#16#0        B#16#0          data 0
        v0166     BYTE    B#16#1F       B#16#1F         data 1
        v0176     BYTE    B#16#0        B#16#0          data 2
        v0186     BYTE    B#16#0        B#16#0          data 3
        v0196     BYTE    B#16#0        B#16#0          reserve
        v0107     WORD    W#16#6060     W#16#6060       index
        v0127     BYTE    B#16#0        B#16#0          subindex
        v0137     BYTE    B#16#1        B#16#1          ccs
        v0147     BYTE    B#16#1        B#16#1          length
        v0157     BYTE    B#16#1        B#16#1          data 0
        v0167     BYTE    B#16#0        B#16#0          data 1
        v0177     BYTE    B#16#0        B#16#0          data 2
        v0187     BYTE    B#16#0        B#16#0          data 3
        v0197     BYTE    B#16#0        B#16#0          reserve
        v0108     WORD    W#16#6040     W#16#6040       index
        v0128     BYTE    B#16#0        B#16#0          subindex
        v0138     BYTE    B#16#1        B#16#1          ccs
        v0148     BYTE    B#16#2        B#16#2          length
        v0158     BYTE    B#16#0        B#16#0          data 0
        v0168     BYTE    B#16#F        B#16#F          data 1
        v0178     BYTE    B#16#0        B#16#0          data 2
        v0188     BYTE    B#16#0        B#16#0          data 3
```

```
           v0198     BYTE    B#16#0        B#16#0          reserve
           v0109     WORD    W#16#6041     W#16#6041       index
           v0129     BYTE    B#16#0        B#16#0          subindex
           v0139     BYTE    B#16#2        B#16#2          ccs
           v0149     BYTE    B#16#0        B#16#0          length
           v0159     BYTE    B#16#0        B#16#0          data 0
           v0169     BYTE    B#16#0        B#16#0          data 1
           v0179     BYTE    B#16#0        B#16#0          data 2
           v0189     BYTE    B#16#0        B#16#0          data 3
           v0199     BYTE    B#16#0        B#16#0          reserve
           v01010    WORD    W#16#1801     W#16#1801       index
           v01210    BYTE    B#16#2        B#16#2          subindex
           v01310    BYTE    B#16#1        B#16#1          ccs
           v01410    BYTE    B#16#1        B#16#1          length
           v01510    BYTE    B#16#0        B#16#0          data 0
           v01610    BYTE    B#16#0        B#16#0          data 1
           v01710    BYTE    B#16#0        B#16#0          data 2
           v01810    BYTE    B#16#0        B#16#0          data 3
           v01910    BYTE    B#16#0        B#16#0          reserve
           v030      WORD    W#16#EEEE     W#16#EEEE       index
           v012101   BYTE    B#16#2        B#16#2          subindex
           v013101   BYTE    B#16#1        B#16#1          ccs
           v014101   BYTE    B#16#1        B#16#1          length
           v015101   BYTE    B#16#0        B#16#0          data 0
           v016101   BYTE    B#16#0        B#16#0          data 1
           v017101   BYTE    B#16#0        B#16#0          data 2
           v018101   BYTE    B#16#0        B#16#0          data 3
  129.0    v019101   BYTE    B#16#0        B#16#0          reserve
  130.0    v0301     WORD    W#16#1000     W#16#1000       index
  132.0    v012102   BYTE    B#16#0        B#16#0          subindex
  133.0    v013102   BYTE    B#16#2        B#16#2          ccs
           v014102   BYTE    B#16#0        B#16#0          length
           v015102   BYTE    B#16#0        B#16#0          data 0
           v016102   BYTE    B#16#0        B#16#0          data 1
           v017102   BYTE    B#16#0        B#16#0          data 2
           v018102   BYTE    B#16#0        B#16#0          data 3
           v019102   BYTE    B#16#0        B#16#0          reserve
  140.0    v0302     WORD    W#16#EEEE     W#16#EEEE       index


DB100: DATA_DB = Data block with input and output data of the maximum 127 motors
================================================================================
Address    Name              Type          Initial val.    Actual val.    Comment
-------    ----              ---           ------------    -----------    ---------
    0.0    force1            BYTE          B#16#0          B#16#0         Motor 1
    1.0    res1              BYTE          B#16#0          B#16#0
    2.0    steuerwort1       WORD          W#16#0          W#16#0
    4.0    sollposition1     DWORD         DW#16#0         DW#16#0
    8.0    empfangszaehler1  BYTE          B#16#0          B#16#0
    9.0    reserve1          BYTE          B#16#0          B#16#0
   10.0    statuswort1       WORD          W#16#0          W#16#0
   12.0    istposition1      DWORD         DW#16#0         DW#16#0
   16.0    force2            BYTE          B#16#0          B#16#0         Motor 2
           res2              BYTE          B#16#0          B#16#0
           steuerwort2       WORD          W#16#0          W#16#0
           sollposition2     DWORD         DW#16#0         DW#16#0
           empfangszaehler2  BYTE          B#16#0          B#16#0
           reserve2          BYTE          B#16#0          B#16#0
           statuswort2       WORD          W#16#0          W#16#0
           istposition2      DWORD         DW#16#0         DW#16#0
           force3            BYTE          B#16#0          B#16#0         Motor 3
           res3              BYTE          B#16#0          B#16#0
           steuerwort3       WORD          W#16#0          W#16#0
           sollposition3     DWORD         DW#16#0         DW#16#0
           empfangszaehler3  BYTE          B#16#0          B#16#0
           reserve3          BYTE          B#16#0          B#16#0
           statuswort3       WORD          W#16#0          W#16#0
           istposition3      DWORD         DW#16#0         DW#16#0
           force4            BYTE          B#16#0          B#16#0         Motor 4
           res4              BYTE          B#16#0          B#16#0
           steuerwort4       WORD          W#16#0          W#16#0
           sollposition4     DWORD         DW#16#0         DW#16#0
           empfangszaehler4  BYTE          B#16#0          B#16#0
           reserve4          BYTE          B#16#0          B#16#0
           statuswort4       WORD          W#16#0          W#16#0
           istposition4      DWORD         DW#16#0         DW#16#0
           force5            BYTE          B#16#0          B#16#0         Motor 5
           res5              BYTE          B#16#0          B#16#0
           steuerwort5       WORD          W#16#0          W#16#0
```

	sollposition5	DWORD	DW#16#0	DW#16#0	
	empfangszaehler5	BYTE	B#16#0	B#16#0	
	reserve5	BYTE	B#16#0	B#16#0	
	statuswort5	WORD	W#16#0	W#16#0	
	istposition5	DWORD	DW#16#0	DW#16#0	
	force6	BYTE	B#16#0	B#16#0	Motor 6
	res6	BYTE	B#16#0	B#16#0	
	steuerwort6	WORD	W#16#0	W#16#0	
	sollposition6	DWORD	DW#16#0	DW#16#0	
	empfangszaehler6	BYTE	B#16#0	B#16#0	
	reserve6	BYTE	B#16#0	B#16#0	
	statuswort6	WORD	W#16#0	W#16#0	
	istposition6	DWORD	DW#16#0	DW#16#0	
	force7	BYTE	B#16#0	B#16#0	Motor 7
	res7	BYTE	B#16#0	B#16#0	
	steuerwort7	WORD	W#16#0	W#16#0	
	sollposition7	DWORD	DW#16#0	DW#16#0	
	empfangszaehler7	BYTE	B#16#0	B#16#0	
	reserve7	BYTE	B#16#0	B#16#0	
	statuswort7	WORD	W#16#0	W#16#0	
	istposition7	DWORD	DW#16#0	DW#16#0	
	force8	BYTE	B#16#0	B#16#0	Motor 8
	res8	BYTE	B#16#0	B#16#0	
	steuerwort8	WORD	W#16#0	W#16#0	
	sollposition8	DWORD	DW#16#0	DW#16#0	
	empfangszaehler8	BYTE	B#16#0	B#16#0	
	reserve8	BYTE	B#16#0	B#16#0	
	statuswort8	WORD	W#16#0	W#16#0	
	istposition8	DWORD	DW#16#0	DW#16#0	
128.0	force9	BYTE	B#16#0	B#16#0	Motor 9
	res9	BYTE	B#16#0	B#16#0	
	steuerwort9	WORD	W#16#0	W#16#0	
	sollposition9	DWORD	DW#16#0	DW#16#0	
	empfangszaehler9	BYTE	B#16#0	B#16#0	
	reserve9	BYTE	B#16#0	B#16#0	
	statuswort9	WORD	W#16#0	W#16#0	
	istposition9	DWORD	DW#16#0	DW#16#0	

9. Important CANopen Messages

The following table gives a short list of important general CANopen messages.

CAN-identifier [HEX]	Designation	Length	Data [HEX]	Explanations
0	NMT	2	01 xx	Starting all (preoperational -> operational)
0	NMT	2	80 xx	Operational -> preoperational
0	NMT	2	81 xx	Reset (e.g. CAN-I/O-module)
0	NMT	2	82 xx	Reset communication
80h	SYNC	0	-	Sync all
80h + *Node-ID*	EMCY	0...8 bytes	error code	Emergency message (e.g. by CANopen-I/O-module)

Node-ID. ... Node-ID of the accessed CANopen module

10. License

The CAN-DP/2 gateway uses the opensource FreeRTOS™ operating system.
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.