CAN-DP/2

PROFIBUS-DP / CAN-Gateway

Software-Handbuch

zu Artikel: 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

Hinweis

Der Inhalt dieses Handbuches wurde mit größter Sorgfalt erarbeitet und geprüft. **esd** übernimmt jedoch keine Verantwortung für Schäden, die aus Fehlern in der Dokumentation resultieren könnten. Insbesondere Beschreibungen und technische Daten sind keine zugesicherten Eigenschaften im rechtlichen Sinne.

esd hat das Recht, Änderungen am beschriebenen Produkt oder an der Dokumentation ohne vorherige Ankündigung vorzunehmen, wenn sie aus Gründen der Zuverlässigkeit oder Qualitätssicherung vorgenommen werden oder dem technischen Fortschritt dienen.

esd electronic system design gmbh
Vahrenwalder Str. 207
30165 Hannover

Tel.: 0511/372 98-0
FAX : 0511/372 98-68
E-Mail: info@esd.eu
Internet: www.esd.eu

Handbuch-Datei:	I:\Texte\Doku\MANUALS\CAN\CAN-DP2\Deutsch\CAN-DP2_Software-Handbuch_de_11.wpd
Datum der Druckvorlage nerstellung:	2013-02-12

Beschriebene Software-Version:	Command-File: DP/CANopen:	CBXDP_00 1.0.0

Änderungen in den Kapiteln

Die hier aufgeführten Änderungen im Anwenderhandbuch betreffen sowohl Änderungen in der Firmware als auch reine Änderungen in der Beschreibung der Sachverhalte.

Handbuch-Rev.	Kapitel	Änderungen gegenüber Vorversion
1.0	-	Erste Version des deutschen CAN-DP/2-Software-Handbuchs
1.1	10	Kapitel neu: Lizenzhinweise zu Opensource FreeRTOS™ Betriebssystem

Weitere technische Änderungen vorbehalten.

Diese Seite ist bewusst unbedruckt.

Inhalt

Seite

1. Übersicht

1.1 Zu diesem Handbuch

Dieses Handbuch beschreibt die lokale Firmware des CAN-DP/2-Moduls. Die lokale Firmware steuert den Datenaustausch zwischen PROFIBUS-DP (im folgenden nur PROFIBUS genannt) und CAN.

Schicht 2-Implementierung
Das Handbuch enthält die Beschreibung der Schicht 2-Implementierung und der implementierten CANopen-Funktionalitäten.

Page-Mode
Des weiteren beschreibt das Handbuch den Page-Mode, der entwickelt wurde, um die Ansteuerung von mehr als 48 CAN-Identifiern mit einem Gateway zu ermöglichen. Zum allgemeinen Verständnis werden zunächst grundlegende Funktionen des Page-Modes beschrieben. Im Anschluss daran erfolgt die Beschreibung der Funktionsbausteine (FBs) und der Datenbausteine (DBs), die für die Realisierung des Page-Modes zur Verfügung gestellt werden.

11-Bit- und 29-Bit Identifier
Das Modul CAN-DP/2 unterstützt 11-Bit-CAN-Identifier und 29-Bit-CAN-Identifier (CAN2.0A/B).

1.2 Einführung in die Funktionsweise der Firmware

Das Gateway simuliert dem PROFIBUS ein Slave-Device mit einer definierten Anzahl von Ein- und Ausgangsbytes. Nach der Konfiguration des Gateways können CAN-Geräte wie PROFIBUS-Slaves angesprochen werden.

Die PROFIBUS-Ausgangsbytes werden auf den CAN-Bus gesendet. Ein bis acht Ausgangsbytes werden einem Tx-Identifier zugeordnet und können optional auch zyklisch gesendet werden. Den Eingangsbytes werden auf der CAN-Seite Rx-Identifier zugeordnet. Empfangene CAN-Daten werden vom PROFIBUS wie Eingangsdaten behandelt.

Die PROFIBUS-Stationsadresse wird über die Kodierschalter direkt am CAN-DP/2-Modul eingestellt.

1.3 Konfiguration über PROFIBUS-DP

Das CAN-DP/2-Modul wird über den PROFIBUS konfiguriert. Als Konfigurationstool kann z.B. der Siemens SIMATIC Manager für S7 verwendet werden. Dem Gateway werden dort logische Steckplätze (Module) zugeordnet, denen im Verlauf der Konfiguration weitere Parameter wie die SPS-Adresse, Datenrichtung, Datenlänge und CAN-Identifier zugeordnet werden.

1.4 Mehr adressierbare Identifier über den Page-Mode

Der Page-Mode bietet die Möglichkeit, mehr CAN-Identifier zu adressieren, als in einem PROFIBUS-Telegramm untergebracht werden können (also mehr als 48). Die Anzahl der möglichen Identifier wird lediglich durch den verfügbaren Speicherplatz auf der SPS und dem CAN-Gateway eingeschränkt.

Hinweis:
Der Page Mode kann nur genutzt werden, wenn als Konfigurations-Tool der Siemens SIMATIC Manager für S7 verwendet wird!

2. Funktionsweise der lokalen Firmware

Die folgende Abbildung soll die Funktionsweise der Firmware verdeutlichen.

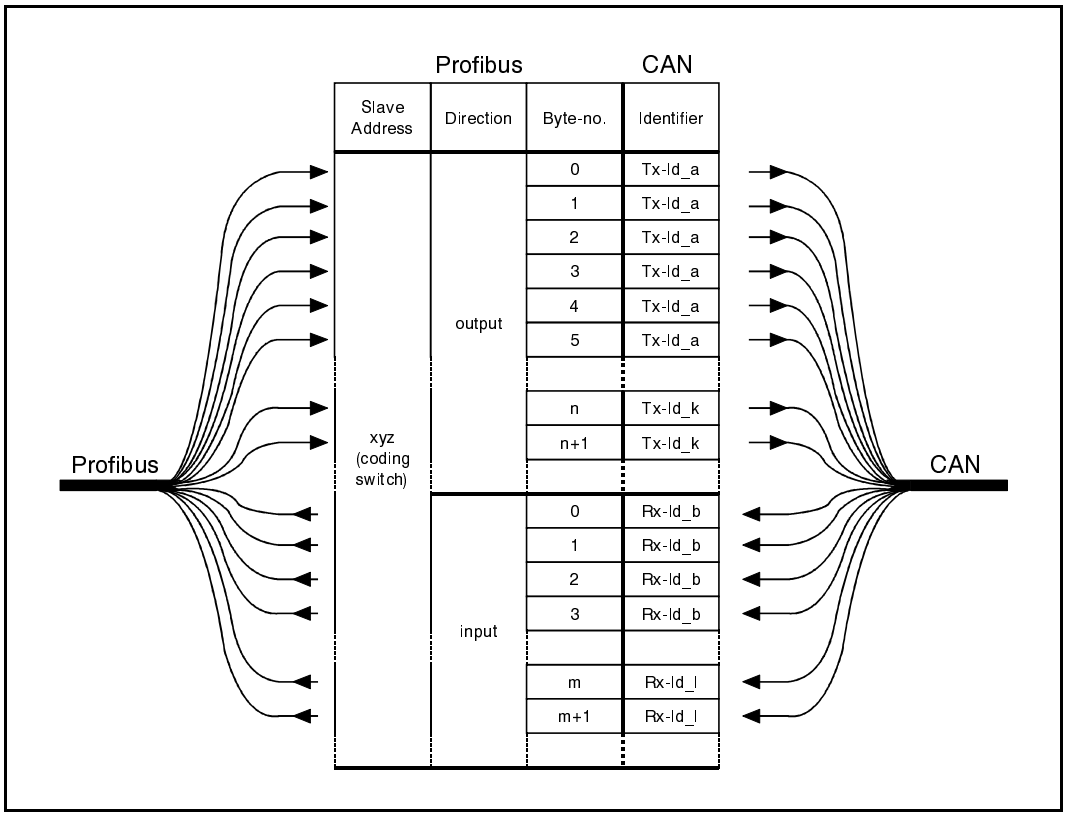


Abb. 2.1.1: Funktionsübersicht des CAN-DP/2-Moduls

2.1 PROFIBUS-Slave-Adresse

Das CAN-DP/2-Modul simuliert auf der PROFIBUS-Seite ein Slave-Modul. Die Slave-Adresse wird über die Kodierschalter am Modul eingestellt. Beim Einschalten wird die hexadezimal eingestellte PROFIBUS-Adresse abgefragt. Änderungen der Einstellungen müssen daher vor dem Einschalten durchgeführt werden, da Änderungen während des Betriebes keine Auswirkungen haben.

Der einstellbare Adressbereich beträgt *hexadezimal* 03 bis 7C bzw. *dezimal* 3 bis 124. Wird eine Adresse kleiner als 3 (dezimal) bzw. kleiner als 03_h eingestellt, so gilt die Adresse 3. Bei einer Adresseinstellung höher als $7C_h$ bzw. höher als 124 (dezimal) gilt die Adresse 124.

Der links angeordnete Kodierschalter (HIGH) dient zur Einstellung der höherwertigen Bits, der rechts angeordnete Kodierschalter (LOW) zur Einstellung der niederwertigen Bits.

Die PROFIBUS-Slave-Adresse kann *nur* über die Kodierschalter eingestellt werden. Eine Programmierung durch einen Klasse 2-Master über das Kommando ‘Set_Slave_Address’ ist *nicht* möglich.

2.2 Nutzdaten

Das CAN-DP/2-Modul simuliert in der aktuellen Software-Implementierung für die Eingangsrichtung und die Ausgangsrichtung zusammen bis zu 300 Bytes. Von diesen 300 Bytes können maximal 244 Bytes für eine Datenrichtung gewählt werden, ansonsten ist die Aufteilung als Eingangs-Bytes und Ausgangs-Bytes frei wählbar. (Beispiele: 150 Eingangs-Bytes und 150 Ausgangs-Bytes oder 244 Eingangsbytes und 56 Ausgangsbytes).

Jeweils ein bis acht Bytes (16 Bytes bei Verwendung des Communication-Windows, siehe ab Seite 38) werden einem Tx- oder einem Rx-Identifier zugeordnet. Ein und der selbe Identifier kann nicht als Tx-*und* als Rx-Identifier genutzt werden.
Optional können die einem Tx-Identifier zugeordneten Bytes auch zyklisch gesendet werden.

2.3 Watchdog (Ansprechüberwachung)

Die Firmware kann mit aktivierter oder deaktivierter Ansprechüberwachung betrieben werden. Es wird jedoch empfohlen, mit aktivierter Ansprechüberwachung zu arbeiten.

2.4 Diagnose

Zur Diagnose können der Leuchtzustand der LED-Anzeigen und die DP-Slave-Diagnose ausgewertet werden. Das Modul unterstützt fünf modulspezifische Diagnose-Bytes. Die Diagnose wird ab Seite 14 ausführlich beschrieben.

2.5 Parametrierungstelegramm (CAN-Bitrate)

Das CAN-DP/2-Modul unterstützt neben den sieben Standard-Bytes der Parametrierung acht weitere Modul-spezifische Byte. Hier kann der DP-Master z.B. die CAN-Bitrate verändern. Die Einstellung der Bitrate über das Parametrierungstelegramm ist auf Seite 27 beschrieben.

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

Die Global-Control-Dienste sind zur Zeit nicht implementiert.

2.7 PROFIBUS-DP Profile

Die PROFIBUS-DP Profile werden zur Zeit nicht unterstützt.

2.8 Mehr ansprechbare CAN-Identifier mit dem Page-Mode

Der Page-Mode bietet die Möglichkeit, mehr CAN-Identifier zu adressieren, als in einem PROFIBUS-Telegramm untergebracht werden können (also mehr als 48).

Das Handling des Page-Mode ist durch den zusätzlichen Protokollaufwand etwas komplizierter als beim Standardbetrieb des Gateways und der Datenaustausch zwischen PROFIBUS und CAN benötigt zwei statt einem SPS-Zyklus.

3. Inbetriebnahme und Diagnose

3.1 Voraussetzung für die Inbetriebnahme

In diesem Kapitel wird die Inbetriebnahme des CAN-DP/2-Moduls an einem PROFIBUS beschrieben, der von einer Siemens SIMATIC-S7-300 oder S7-400 gesteuert wird.
Um die Inbetriebnahme wie hier beschrieben durchführen zu können, benötigen Sie das Konfigurationsprogramm 'SIMATIC-Manager' mit dem Tool 'HW-Konfigurator'.

Hinweis:
Konfigurieren Sie das CAN-DP/2-Modul unbedingt zuerst mit der SPS über den SIMATIC-Manager wie in Kapitel: "5. Konfiguration mit dem SIMATIC Manager" beschrieben. Erst nach erfolgter Konfiguration kann das CAN-DP/2-Modul als CAN-Gerät erkannt werden!

3.2 Inbetriebnahme

3.2.1 Vorgehensweise

Bitte führen Sie zur Inbetriebnahme die folgenden Schritte aus:

1	Montieren und Verdrahten Sie das CAN-DP/2-Modul (Spannungsversorgung, CAN-Bus, siehe Hardware-Handbuch).
2	Stellen Sie die PROFIBUS-Adresse des Moduls am Kodierschalter ein.
3	Schließen Sie den PROFIBUS-Anschlussstecker an die PROFIBUS-Schnittstelle des CAN-DP/2-Moduls an.
4	Konfigurieren Sie die Einstellungen des CAN-DP/2-Moduls in der SPS mit dem SIMATIC-Manager.
5	Schalten Sie das Netzteil für das CAN-DP/2 ein. Das Modul muss jetzt anlaufen. Das CAN-DP/2-Modul wird nun automatisch über die SPS konfiguriert.

Hinweis:
Beachten Sie, dass insbesondere die CAN-Bitrate und die Module-ID (bei CANopen) über den PROFIBUS gesetzt sein müssen.

3.2.2 Anlauf

Nach dem Einschalten des Netzteils läuft das CAN-DP/2-Modul automatisch an. Es besitzt keinen eigenen Netzschalter.
Während der Anlaufphase blinken die LEDs “E” (PROFIBUS-DP Status) und die LED “D” (PROFIBUS-DP Data Exchange). Die an den Kodierschaltern eingestellte PROFIBUS-Adresse wird eingelesen.
Das Modul empfängt vom DP-Master die Projektierungsdaten und wertet die Angaben darin aus. Wenn die Projektierung mit dem Aufbau übereinstimmt, nimmt das CAN-DP/2-Modul den Datentransfer auf.

3.2.3 Datentransfer

Ist das Modul konfiguriert, so wird nach dem Anlauf automatisch der Datentransfer aufgenommen: Ändert der SPS-Master Sendedaten eines Identifiers, so werden die Daten vom CAN-DP/2-Modul auf dem CAN-Bus gesendet. Empfängt das CAN-DP/2-Modul Daten, so stellt es diese dem SPS-Master zur Verfügung.
Die Konfiguration ist in Kapitel 5 ‘Konfiguration mit dem SIMATIC-Manager’ ab Seite 25 beschrieben.

3.3 Diagnose über LED-Anzeige

Die Funktion der Leuchtdioden ist durch die Firmware festgelegt. Im normalen Betrieb sind die LEDs D, P und C nie aus, d.h. entweder sie blinken oder sie leuchten permanent.

Die Blinksequenzen, die in der folgenden Tabelle aufgeführt sind, wiederholen sich etwa alle sechs Sekunden.

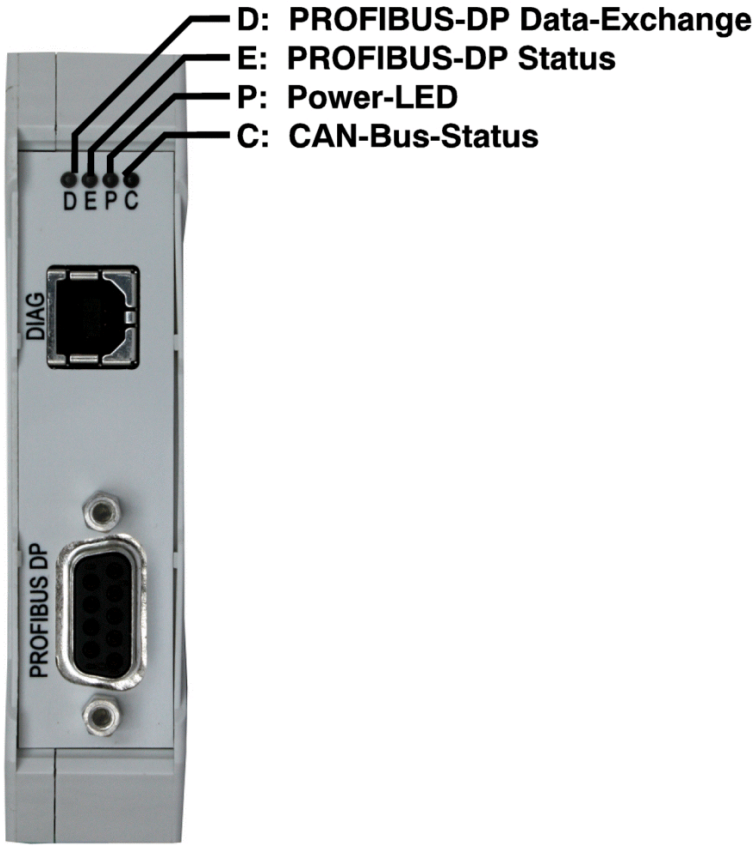


Abb. 3.3.1: Position der LEDs

LED	Funktion	Leuchtzustand	Bedeutung	Fehlerbehandlung
D (grün)	PROFIBUS-DP Data Exchange	aus	kein Datenaustausch	-
		an	Datenaustausch über PROFIBUS	-
E (rot)	PROFIBUS-DP Status	aus	PROFIBUS OK	-
		1x kurz blinkend	Bitrate wird gesucht	die Verbindung zum DP-Master ist ausgefallen, überprüfen Sie den PROFIBUS-Anschluss (Verdrahtungsfehler im PROFIBUS-Kabel, Kurzschluss, Abschlusswiderstand an falscher Stelle zugeschaltet ?)
		2x kurz blinkend	Bitrate wird überwacht	überprüfen Sie die eingestellte PROFIBUS-Adresse
		3x kurz blinkend	warten auf Parametriertelegramm	Parametriertelegramm ist fehlerhaft. Diagnose über SIMATIC-Manager oder System-Funktion SFC13 (DPNRM_DG) (siehe Kap. 3.4)
		4x kurz blinkend	warten auf Konfigurations-telegramm	Konfigurationstelegramm ist fehlerhaft. Diagnose über SIMATIC-Manager oder System-Funktion SFC13 (DPNRM_DG) (siehe Kap. 3.4)
		an	Störung	interner Fehler
P (grün)	Power LED	aus	Initialisierung noch nicht beendet	-
		an	Initialisierung erfolgreich abgeschlossen	-
C (grün)	CAN Bus Status	aus	keine Versorgungsspannung	überprüfen Sie die 24 V-Spannungs-versorgung
		1x kurz blinkend	CAN-Error (Morsezeichen ‘E’)	Störungen auf CAN Bus, Verdrahtung und Bitrate prüfen, siehe auch Hardware-Handbuch
		3x lang blinkend	CAN-Off (Morsezeichen ‘O’)	
		kurz-lang-lang	CAN-Warning (‘W’)	
		an	CAN-Bus OK	-

Tabelle 3.3.1: Leuchtzustände der LEDs

3.4 Slave-Diagnose

Das Modul unterstützt neben den in der Norm DIN EN 19245, Teil 3 vordefinierten sechs Diagnose-Bytes fünf weitere modulspezifische Diagnose-Bytes.

Die Slave-Diagnose kann mit folgenden Funktionsbausteinen angefordert werden:

Automatisierungsgerätefamilie	Nummer	Name
SIMATIC mit IM 308-C SIMATIC S7/M7	FB 192 SFC 13	FB IM308C SFC DPNRM_DG

Tabelle 3.4.1: Funktionsbausteine zur Anforderung der Slave-Diagnose

3.4.1 Diagnose-Bytes 0...5

Die Belegung dieser Diagnose-Bytes ist in der Norm DIN EN 19245, Teil 3 vordefiniert. Im folgenden werden die Statusmeldungen unter Berücksichtigung des CAN-DP/2-Moduls beschrieben.

Folgende Bezeichnungen werden dabei verwendet:

Byte-Nummer	Status-Byte Bezeichnung
0	Stationsstatus 1
1	Stationsstatus 2
2	Stationsstatus 3
3	Master-PROFIBUS-Adresse
4	Herstellerkennung- High Byte
5	Herstellerkennung- Low Byte

Tabelle 3.4.2: Diagnose-Bytes 0...5

3.4.1.1 Stationsstatus 1

Der Stationsstatus 1 enthält Fehlermeldungen des DP-Slaves. Ist ein Bit ‘0’, so liegt kein Fehler vor. Ein auf ‘1’ gesetztes Bit signalisiert einen Fehler.

Bit	Fehlermeldung, wenn Bit = ‘1’	Fehlerbehandlung
0	DP-Slave kann nicht vom Master angesprochen werden	- richtige PROFIBUS-Adresse am CAN-DP/2 eingestellt ? - Busstecker korrekt angeschlossen ? - Betriebsspannung am CAN-DP/2 vorhanden ? - Netz aus/Netz ein am CAN-DP/2 durchgeführt, um DP-Adresse einzulesen ?
1	DP-Slave ist für den Datenaustausch noch nicht bereit	- warten, bis CAN-DP/2 die Hochlaufphase abgeschlossen hat
2	Die vom DP-Master an den DP-Slave gesendeten Konfigurations-daten stimmen nicht mit dem Aufbau des DP-Slaves überein.	- prüfen, ob der Stationstyp und der Aufbau des CAN-DP/2 mit dem Konfigurationstool korrekt eingeben worden sind
3	Der Slave hat externe Diagnosedaten.	- externe Diagnosedaten abfragen und auswerten
4	Die angeforderte Funktion wird vom DP-Slave nicht unterstützt.	- Projektierung prüfen
5	DP-Master kann die Antwort des DP-Slaves nicht interpretieren.	- Busaufbau prüfen
6	Falsche Parametrierung	- externe Diagnose-Bytes 9 und 10 auswerten
7	DP-Slave ist bereits von einem anderen Master parametriert worden.	- Dieses Bit ist immer ‘1’, wenn Sie z.B. gerade mit dem PG oder einem anderen DP-Master auf das CAN-DP/2 zugreifen. Die PROFIBUS-Adresse des Parametrier-masters befindet sich im Diagnose-Byte ‘Master-PROFIBUS-Adresse’.

Tabelle 3.4.3: Bits des Stationsstatus 1

3.4.1.2 Stationsstatus 2

Der Stationsstatus 2 enthält Statusmeldungen zum DP-Slave. Ist ein Bit ‘1’, so ist die entsprechende Meldung aktiv. Ein auf ‘0’ gesetztes Bit zeigt eine inaktive Meldung.

Bit	Fehlermeldung, wenn Bit = ‘1’
0	DP-Slave muss neu parametriert werden.
1	Es liegt eine Diagnosemeldung vor. Der DP-Slave kann nicht weiterlaufen, solange der Fehler nicht behoben ist (statische Diagnosemeldung).
2	Dieses Bit ist immer ‘1’.
3	Die Ansprechüberwachung ist für das CAN-DP/2 aktiviert.
4	DP-Slave hat das Freeze-Kommando empfangen.
5	DP-Slave hat das SYNC-Kommando empfangen.
6	Dieses Bit ist immer ‘0’.
7	DP-Slave ist deaktiviert.

Tabelle 3.4.4: Bits des Stationsstatus 2

3.4.1.3 Stationsstatus 3

Der Stationsstatus 3 ist reserviert und für die Diagnose des CAN-DP/2 ohne Bedeutung.

3.4.1.4 Diagnose-Byte 3: Master-PROFIBUS-Adresse

In diesem Byte wird die PROFIBUS-Adresse des Masters abgelegt, der den DP-Slave zuletzt parametriert hat und lesenden und schreibenden Zugriff auf den DP-Slave hat.

3.4.1.5 Diagnose-Byte 4 und 5: Herstellerkennung

Die Herstellerkennung ist in zwei Bytes kodiert. Für das CAN-DP/2-Modul wird die Kennung $\mathbf{04A4_h}$ zurückgegeben.

3.4.2 Externe (modulspezifische) Diagnose-Bytes

Das CAN-DP/2-Modul unterstützt die Diagnose-Bytes 6 bis 10 für modulspezifische Diagnosemeldungen.

Diagnose-Byte	Bedeutung
0...5	definiert in der PROFIBUS-Spezifikation (siehe vorangegangene Kapitel)
6	Längenangabe für modulspezifische Diagnoseinformationen (hier immer 5)
7	Header byte: Bit 0...5 enthalten die Blocklänge incl. Header (hier immer 4)
8	- DP-Dienst (SAP) der zum Fehler führte (Byte 8 = $3D_h$, $3E_h$), oder - Bus-Zustand, wenn in der Parametrierung das Flag *CAN-Diagnosis* = “yes”: Byte 8 kann dann folgende Werte einnehmen: 00_h OK 40_h WARN 80_h ERROR_PASSIVE $C0_h$ BUS_OFF
9	abhängig von Zustand des Bytes 8: Byte 8 = $3D_h$ Parametrierung (SAP61) fehlerhaft, Byte 9 enthält die Nummer des fehlerhaften Parametrier-Bytes Byte 8 = $3E_h$ Konfiguration (SAP62) fehlerhaft, Byte 9 enthält die Nummer des fehlerhaften PROFIBUS-Moduls (= Adresse des simulierten SPS-Steckplatzes) Byte 8 = 00_h, 40_h, 80_h oder C0h (Bus-Zustand) Byte 9 enthält den IRQ_LOST-Counter aus dem eingebauten CAN-Treiber des CAN-DP/2 Der IRQ_LOST-Counter ist der Zähler für verlorengegangene Nachrichten des CAN-Controllers. Dieser Zähler wird von einem Fehler-Ausgang des CAN-Controllers gesetzt. Er zeigt die Anzahl der verlorengegangenen CAN-Frames an (Empfangs- oder Sende-Nachrichten).

Diagnose-Byte	Bedeutung
10	abhängig von Zustand des Bytes 8: Byte 8 = $3D_h$ Parametrierung (SAP61) fehlerhaft, Byte 10 gibt dem PROFIBUS-Master den korrekten Wert an Byte 8 = $3E_h$ Konfiguration (SAP62) fehlerhaft 1 falscher E/A-Typ: "Aus- Eingang" oder "Leerplatz") richtig wäre "Eingang" oder "Ausgang" 2 falsche Einheit, z.B. "Worte" richtig wäre: Einheit = "Byte" 3 falsche Länge richtig wäre Länge = 1-8 oder 16 4 nur ein Byte für Identifier angegeben 5 Format-Angabe fehlt 6 falscher Identifier Byte 8 = 00_h, 40_h, 80_h oder C0h (Bus-Zustand) Byte 10 enthält den MSG_LOST-Counter aus dem eingebauten CAN-Treiber des CAN-DP/2. Der MSG_LOST_Counter ist der Zähler für verlorengegangene Nachrichten des FIFOs. Dieser Zähler wird erhöht, wenn Nachrichten aufgrund eines FIFO-Überlaufs verloren gehen (FIFO full).

Tabelle 3.4.5: Modulspezifische Statusmeldungen

4. GSD-Datei

Im Folgenden ist die GSD-Datei (Geräte Stammdaten-Datei) des CAN-DP/2-Moduls abgedruckt. Die hier abgedruckten Angaben sollen zur Orientierung dienen. Ausschlaggebend sind die Daten in der mitgelieferten GSD-Datei **CDPS04A4.GSD**.

```
;==========================================================================================
; (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. Konfiguration mit dem SIMATIC Manager

5.1 Ablauf der Konfiguration

Das CAN-DP/2-Modul wird über den PROFIBUS konfiguriert.

Um das CAN-DP/2-Modul zu konfigurieren, ist wie folgt vorzugehen:

Hinweis:
Ohne korrekte Konfiguration mit dem SIMATIC Manager arbeiten das CAN-DP/2-Modul und die angeschlossenen CAN-Teilnehmer nicht zusammen und der Betrieb der angeschlossenen CAN-Teilnehmer kann gestört werden.
Insbesondere die im CAN-DP/2-Modul konfigurierte CAN-Bitrate und die Module-ID (bei CANopen) müssen zu den Einstellungen der angeschlossenen CAN-Teilnehmer passen!
Sollten Probleme auftreten, erhalten Sie weitere Informationen über die Diagnose wie in den Kapiteln "4.3 Diagnose über LED-Anzeige" und "4.4 Slave-Diagnose" beschrieben.

1. **CAN-DP/2 auswählen**
   Im Menü *Hardware Katalog* die Einstellung *Weitere Feldgeräte* und *Sonstige* wählen. Dort ist das *esd CAN-DP/2* auszuwählen.

2. **PROFIBUS-Adresse einstellen**
   Stellen Sie die PROFIBUS-Adresse wie in Kapitel 5.1.1 auf Seite 26 beschrieben ein.

3. **Parametrierungstelegramm (CAN-Bitrate, allgemeine Konfiguration und CANopen-Module-ID einstellen)**
   Führen Sie die Konfigurationseinstellungen mit Hilfe des Parametrierungstelegramms wie in Kapitel 5.1.2 auf Seite 27 beschrieben durch.

4. **Belegung der Steckplätze des DP-Slaves**
   Führen Sie die Belegung der Steckplätze wie in Kapitel 5.1.3 auf Seite 31 beschrieben durch.

5. **Konfiguration der Steckplätze (SPS-Adresse)**
   Konfigurieren Sie die Steckplätze wie in Kapitel 5.1.4 auf Seite 32 beschrieben.

6. **Einstellungen auf Festplatte sichern**
   Speichern Sie die Einstellungen wie in Kapitel 5.1.5 auf Seite 32 beschrieben.

5.1.1 Einstellen der PROFIBUS-Adresse

Es öffnet sich ein Eingabefenster, in dem die PROFIBUS-Stationsadresse eingestellt werden muss.

Achtung!
Hier muss die an den Kodierschaltern ***hexadezimal*** eingestellte Adresse in einen ***Dezimalwert umgerechnet*** und eingegeben werden!

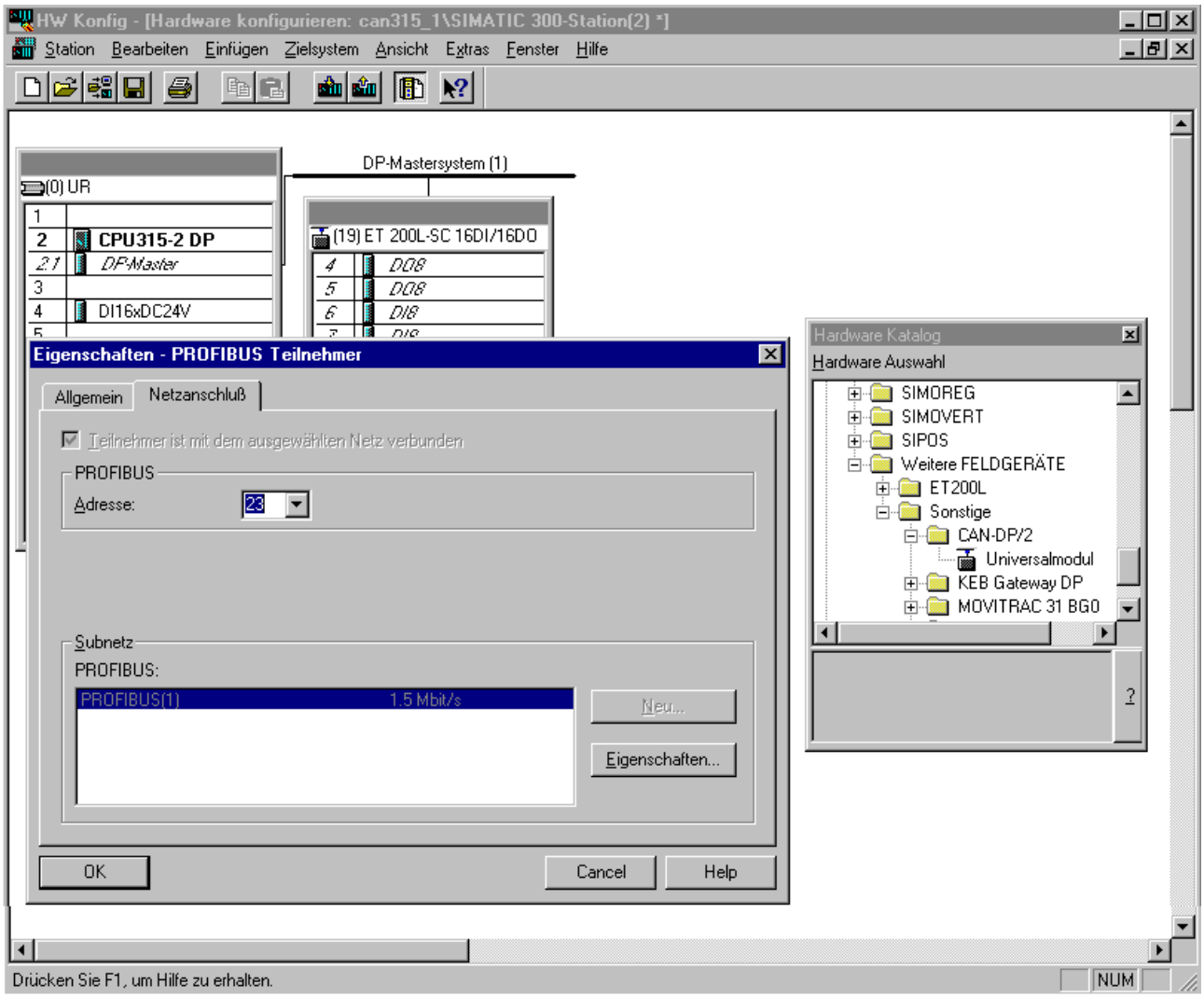


Abb. 5.1.1: Einstellung der PROFIBUS-Adresse des CAN-DP/2

5.1.2 Parametrierungstelegramm

Im Konfigurationsfenster wird jetzt automatisch das Modul ‘DP-Slave’ hinzugefügt. Mit Hilfe des Parametrierungstelegramms können jetzt verschiedene Konfigurationseinstellungen vorgenommen werden.

Die Modul-spezifischen Bytes des Parametrierungstelegramms können in dem Eigenschaftsfenster verändert werden, das sich öffnet, wenn ein Doppelklick auf die Kopfzeile des DP-Slave-Fensters ausgeführt wird (hier Zeile ‘`(17) CAN-DP/2`’).

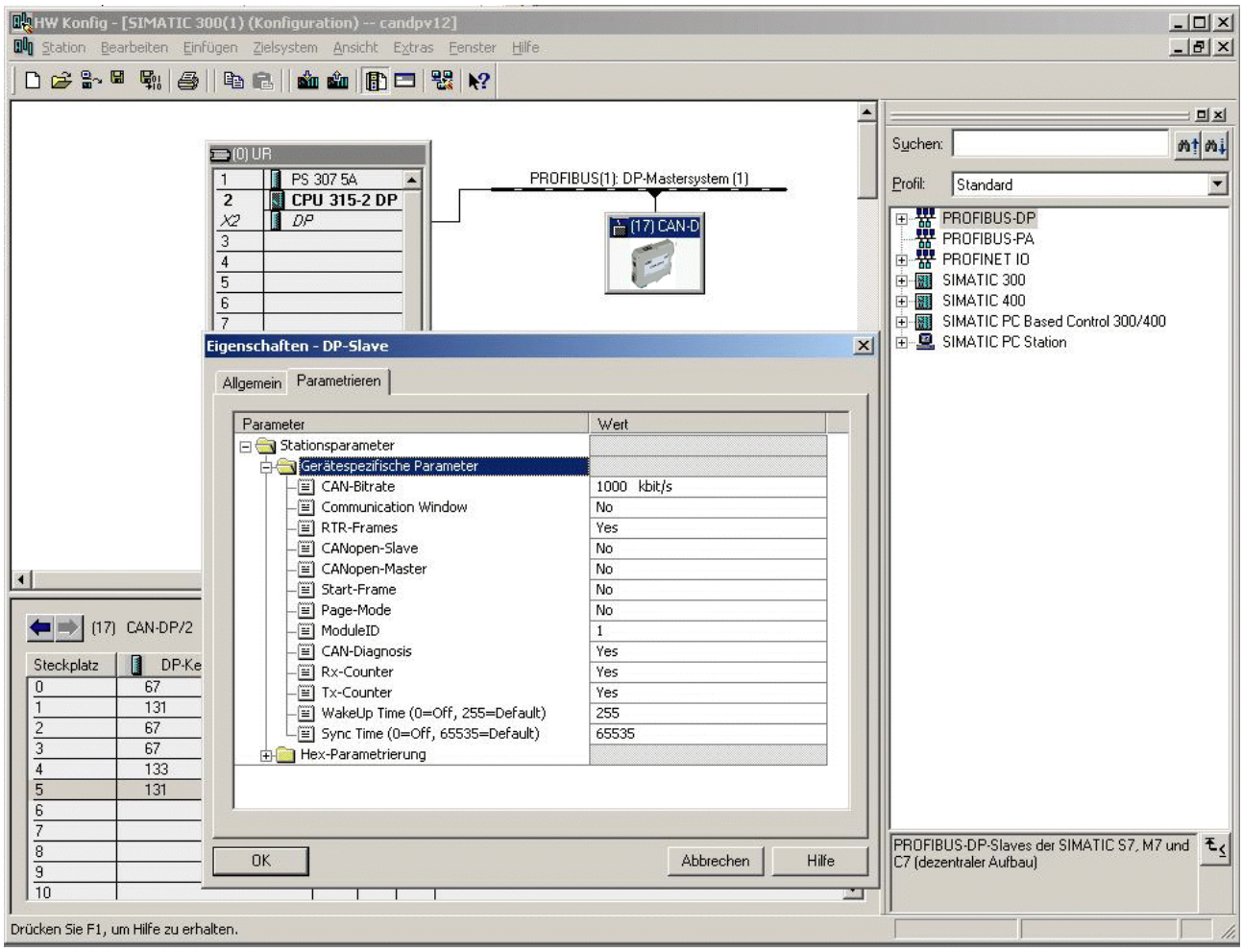


Abb. 5.1.2: Einstellung der Parameter im DP-Slave Eigenschaftsfenster

Hinweis:
Über den Auswahlpunkt *Hex-Parameter* können, wie in älteren Versionen der Software, die Parameter durch Eintrag von Hexadezimalwerten übergeben werden. Komfortabler ist selbstverständlich die Eingabe im oben gezeigten Format. Hier können die Parameter ‘direkt’ gesetzt werden. In den folgenden Beschreibungen wird daher auf die Einstellung mit Hexadezimalzahlen nicht weiter eingegangen.

Beschreibung der Parameter:

CAN-Bitrate Für die Bitrate bestehen die folgenden Auswahlmöglichkeiten:

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

Tabelle 5.1.1: Einstellung der Bitrate in 14 Stufen

Communication Window: (CW) Hier wird das Communication Window aktiviert. Es ist ab Seite 38 ausführlich beschrieben.

RTR-Frames: (NR) Senden von RTR-Frames für die über PROFIBUS konfigurierten Rx-Identifier freigeben.

CANopen-Slave: (CS) Gateway als CANopen-Slave konfigurieren.

CANopen-Master: (CM) Gateway als CANopen-Master konfigurieren.

Start-Frame: (AS) Nach Ablauf der Wake-Up-Time wird, wenn das Gateway ein Master ist, ein Start-Frame gesendet (Autostart).

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

Zulässige Kombinationen:

CW	NR	CS	CM	AS	PM	Bedeutung
x	no	yes	no	x	no	- nach Wake-Up-Time sendet das Modul automatisch *128 dez + Module-No.* und ist im Zustand 'Pre-Operational' - nach Erhalt eines Start-Frames: TxId ausgeben, RTR-Frames auf RxId senden
x	yes	yes	no	x	no	- nach Wake-Up-Time sendet das Modul automatisch *128 dez + Module-No.* und ist im Zustand 'Pre-Operational' - nach Erhalt eines Start-Frames: TxId ausgeben
x	no	no	yes	no	no	- nach Wake-Up-Time TxID ausgeben - RTR-Frames auf RxId senden
x	yes	no	yes	no	no	- nach Wake-Up-Time TxID ausgeben
x	no	no	yes	yes	no	- nach Wake-Up-Time Start-Frame, TxID ausgeben, RTR-Frames auf RxId senden
x	yes	no	yes	yes	no	- nach Wake-Up-Time Start-Frame, TxID ausgeben
x	yes/no	no	no	yes/no	no	- Die CANopen Einstellungen wie Wake-Up Time, RTR, Start-Frame und Sync-Time werden ignoriert.

x... Zustand des Parameters hier ohne Bedeutung

Tabelle 5.1.2: Beispiele für zulässige Parameterkombinationen

ModuleID: Module-ID des Gateways als CANopen-Slave.
Die *Module-ID*, unter der das Gateway angesprochen wird, wenn es als CANopen-Slave konfiguriert ist, wird über dieses Byte eingestellt.

Wertebereich: 1 ... 127 (dezimal)

CAN-Diagnosis: Wenn dieses Flag auf "yes" gesetzt wird, werden bei Fehlern auf dem CAN-Bus Diagnose-Telegramme (s. S. 20) gesendet

RX-Counter=yes: Im letzten Byte jedes Eingangsmoduls wird mit jedem empfangenen CAN-Telegramm auf diesem Identifier ein 8-bit-Zähler hochgezählt. Dazu muss die Länge für jedes konfigurierte CAN-Telegramm um eins erhöht werden. Soll beispielsweise ein Telegramm mit 8-byte-Datenlänge empfangen werden, so ist im Eingabefenster *Eigenschaften DP- Slave* für die Länge = 9 einzugeben.
Mit *RX-Counter=yes* kann der Empfang von Telegrammen, auch wenn sich der Dateninhalt nicht geändert hat, überwacht werden.

Tx-Counter=yes: Im letzten Byte jedes Ausgangsmoduls ist ein Byte reserviert, in dem z.B. ein Zähler hochgezählt werden kann. Mit jeder Änderung dieses Bytes wird das CAN-Telegramm versendet, auch wenn sich die eigentlichen Daten nicht geändert haben. Auch hier muss die Länge des konfigurierten CAN-Telegramms um eins erhöht werden. Soll beispielsweise ein Telegramm mit 8-byte-Datenlänge gesendet werden, so ist im Eingabefenster *Eigenschaften DP- Slave* für die Länge = 9 einzugeben. Das Zählerbyte wird nicht versandt.

Wakeup Time:

Über den Parameter *Wakeup Time* wird hier eine Wartezeit in Sekunden übergeben, die festlegt, wie lange das Modul nach einem RESET oder Power-On wartet, bevor es Daten auf den CAN-Bus sendet.

Die hier übergebene *Wakeup Time* überschreibt den bisher im CAN-DP/2-Gateway gespeicherten Wert der *Wakeup Time*, wenn ein Wert ungleich '255' eingetragen wird. Wird hier '255' eingetragen, so wird der im Gateway gespeicherte Wert verwendet.
Wird der Parameter *Wakeup Time* auf '0' gesetzt, so hält das Modul keine Wartezeit ein, sondern beginnt mit dem Senden von Daten, sobald diese zur Verfügung stehen.

Die ***Wakeup Time*** wird hier dezimal eingegeben.

Parameter	Wertebereich [dez] in [s]	Erläuterungen
Wakeup Time	0	Wakeup Time-Funktion aus
	1...254	Wakeup Time in Sekunden
	255	Bisherigen Wert aus Gateway verwenden (Default)

Tabelle 5.1.3: Funktion des Parameters *Wakeup Time*

SYNC Time:

Das CAN-DP/2-Modul kann für einfache CANopen-Anwendungen zyklisch das Kommandos SYNC senden.
Die Eingabe der Zykluszeit erfolgt in Millisekunden.
Die ***SYNC Time*** wird hier dezimal eingegeben.

Parameter	Wertebereich [dez] in [ms]	Erläuterungen
SYNC Time	0	kein SYNC senden möglich
	1...65534	SYNC Time in Millisekunden (1...65534 ms)
	65535	Bisherigen Wert aus Gateway verwenden (Default)

Tabelle 5.1.4: Funktion des Parameters *SYNC Time*

Achtung!
Die *SYNC Time* kann auf zwei verschiedene Arten gesetzt werden:
1. Wie hier beschrieben.
2. Über Byte 4 und 5 des Communication Windows (siehe Seite 42)

Alle Einträge sind gleichberechtigt, d.h. der letzte erfolgte Eintrag ist gültig!

5.1.3 Belegung der Steckplätze des DP-Slaves

Die gewünschte Anzahl an Steckplätzen die der DP-Slave für den Datenaustausch verwenden soll, wird eingestellt, indem bei aktiviertem DP-Slave-Fenster im Hardware-Katalog für jedes Byte ein Doppelklick auf das Gerät ‘Universalmodul’ ausgeführt wird. Im DP-Slave-Fenster werden die belegten Steckplätze durch eine ‘0’ angezeigt.

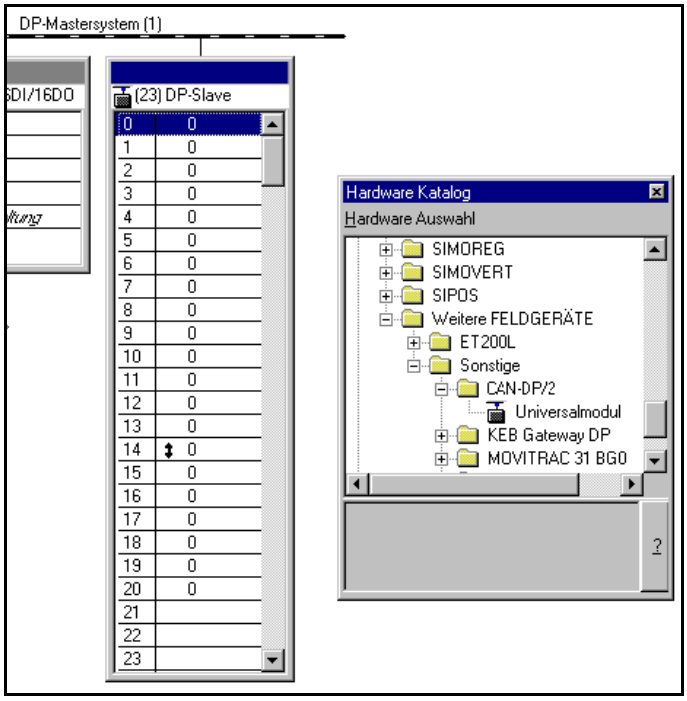


Abb. 5.1.3: DP-Slave Fenster und Hardeware Katalog

5.1.4 Konfiguration der Steckplätze

Zur Konfiguration ist ein Doppelklick auf den Steckplatzeintrag auszuführen. Es öffnet sich ein Eigenschaften-Fenster in dem die simulierten SPS-Steckplätze konfiguriert werden. Im folgenden sind zwei Beispiele mit 11-Bit-CAN-Identifiern dargestellt:

Datenrichtung:	**Eingang**	Datenrichtung:	**Ausgang**
SPS-Adresse:	172 dezimal	SPS-Adresse:	172 dezimal
Länge:	6	Länge:	6
Einheit:	Byte	Einheit:	Byte
Konsistent über:	gesamte Länge	Konsistent über:	gesamte Länge
Identifier:	0289 hexadezimal	Identifier:	0309 hexadezimal
form-Byte:	B8 hexadezimal	*form*-Byte:	B8 hexadezimal
		Zyklisches Senden:	nein

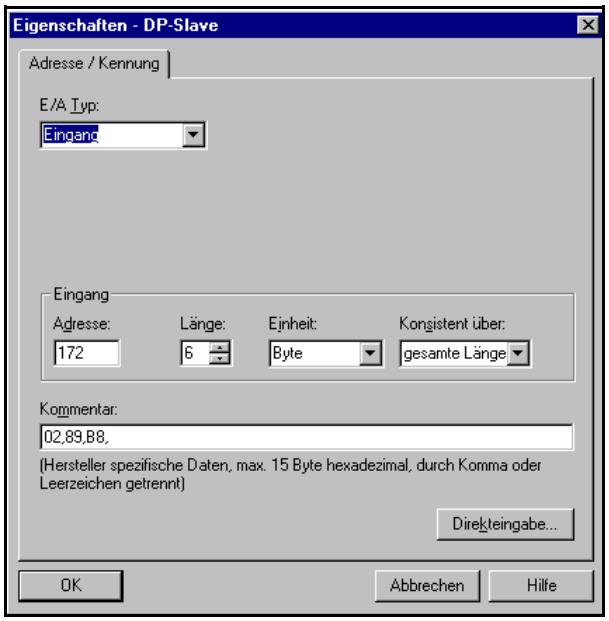


Abb. 5.1.4: Beispiel: Konfiguration von Eingangsdaten

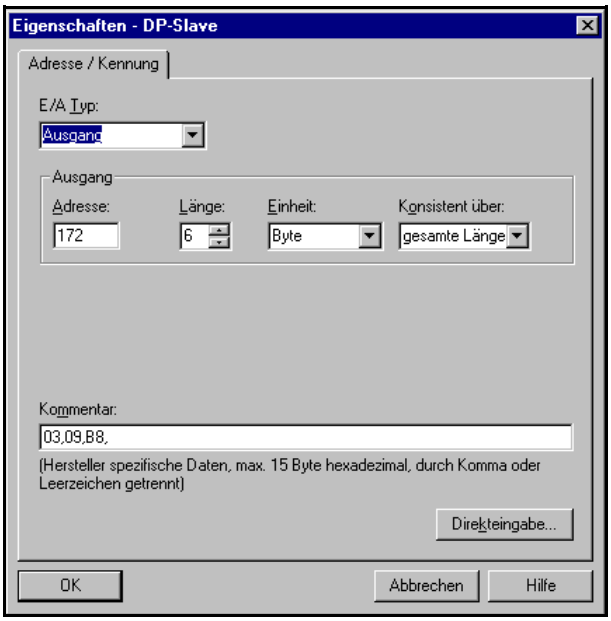


Abb. 5.1.5: Beispiel: Konfiguration von Ausgangsdaten

Achtung!
Um eine einwandfreie Funktion des Moduls zu gewährleisten, muss immer mindestens ein Ausgang (Einheit beliebig) definiert werden. Der Profibus-Controller VPC3 löst keinen Interrupt aus, wenn kein Ausgang definiert ist! Soll bei der Definition eines Ausgangs keine CAN-Zuordnung erfolgen, ist es zulässig, als Identifier den Wert $07F8_h$ einzutragen.

Die einzelnen Parameter des Eigenschaften-Fensters werden im Kapitel: “Beschreibung des Eigenschaftsfensters ‘*Eigenschaften DP-Slave*’” ausführlich beschrieben.

5.1.5 Speichern der Einstellungen auf Festplatte

Jetzt müssen die Einstellungen über die Menüpunkte *Station/Speichern* auf Festplatte gesichert werden. Anschließend werden die Einstellungen über die Menüpunkte *Zielsystem/Laden in Baugruppe* an die SPS übergeben.

5.2 Eingabefenster ‘*Eigenschaften - DP-Slave*’

E/A-Typ

In diesem Feld muss je nach gewünschter Datenrichtung ‘Eingang’ oder ‘Ausgang’ gewählt werden. Andere Einträge sind nicht zulässig.

Adresse

Hier wird die SPS-E/A-Adresse als **Dezimalzahl** eingetragen.

Länge, Einheit

Über diese Felder wird die Anzahl der Datenbytes vorgegeben.

Achtung!
Ist der ***RX-Counter=yes***, muss die konfigurierte Länge empfangener CAN-Telegramme jeweils um eins erhöht werden.
(Soll beispielsweise ein Telegramm mit 8-byte-Datenlänge empfangen werden, so ist im Eingabefenster *Eigenschaften DP-Slave* unter *Eingang* die Länge = 9 einzugeben.)

Ist der ***Tx-Counter=yes,*** muss die konfigurierte Länge gesendeter CAN-Telegramme jeweils um eins erhöht werden.
(Soll beispielsweise ein Telegramm mit 8-byte-Datenlänge gesendet werden, so ist im Eingabefenster *Eigenschaften DP- Slave* unter *Ausgang* die Länge = 9 einzugeben.).

Konsistent über

Der Eintrag in diesem Feld gibt an, ob die Daten während eines SPS-Zyklus als einzelne Einheiten (Bytes, Worte, etc.) oder als Gesamtes Paket (1-8 Bytes, bzw. 16 Bytes bei Communication-Window) übergeben werden sollen. Diese Funktion sollte nur bei Bedarf auf ‘gesamte Länge’ eingestellt werden, da die Übertragung als ‘Einheit’ schneller ist.

Hinweis:
Sollen die Daten konsistent über die gesamte Länge übertragen werden, so ist dies hier einzustellen *und* es müssen SFC14 und SFC15 verwendet werden (siehe Step7-SPS-Handbuch).

Kommentar

Im ***Kommentar***-Feld wird, jeweils durch Kommata getrennt, in den ersten beiden (vier) Bytes der **CAN-Identifier** und danach das Steuerbyte ***form*** übergeben. Optional können bei Ausgängen noch 2 Bytes für das zyklische Senden des in diesem Slot definierten CAN-Telegramms hinter dem Format-Byte folgen. Die beiden Bytes *cycle* geben die Zykluszeit in Millisekunden an. Das Datenformat ist hier für alle Einträge **hexadezimal** (!).
Die einzelnen Einträge im ***Kommentar***-Feld werden in den folgenden Kapiteln beschrieben.

	Kommentar-Bytes bei **11-Bit** Identifiern					
Byte-Nr.	1	2	3	4	5	6 ... 15
Inhalt:	CAN-Identifier		Format-Byte	Zykluszeit (optional bei Ausgängen)		nicht belegt
	ID_high	ID_low	form	Cycle_high	Cycle_low	

	Kommentar-Bytes bei **29-Bit** Identifiern							
Byte-Nr.	1	2	3	4	5	6	7	8 ... 15
Inhalt:	CAN-Identifier				Format-Byte	Zykluszeit (optional bei Ausgängen)		nicht belegt
	ID_UU	ID_UL	ID_LU	ID_LL				
	Bit31...Bit24	Bit23...Bit16	Bit15...Bit08	Bit07...Bit00	form	Cycle_high	Cycle_low	

5.2.1 Eingabe des CAN-Identifiers im *Kommentar*-Feld

Der **CAN-Identifier** wird in den ersten beiden (vier) Bytes des ***Kommentar***-Feldes übergeben.
Die Bytes sind dabei jeweils durch Kommata getrennt anzugeben.

Beispiel: In das Kommentarfeld soll der 11-Bit-Identifier 0309_h eingetragen werden.
Die zwei Bytes für den 11-Bit-Identifier müssen wie folgt in das Kommentar-Feld eingegeben werden: 03,09,

	Kommentar-Bytes bei **11-Bit** Identifiern					
Byte-Nr.	**1**	**2**	3	4	5	6 ... 15
Inhalt:	**CAN-Identifier**		Format-Byte	Zykluszeit (optional bei Ausgängen)		nicht belegt
	ID_high	ID_low	form	Cycle_high	Cycle_low	
Beispiel:	**03,**	**09,**	B8,	-	-	-
	(CAN-Identifier: 0309_h)		(Format-Byte: $B8_h$)	kein zyklisches Senden		-

Hinweis:
Die Eingabe eines 29-Bit-Identifiers erfordert *vier* Bytes und ein auf '1' gesetztes Bit 29 (Zählweise 0...31 Bit), damit das Modul eine Unterscheidung zwischen 11-Bit und 29-Bit-Identifier treffen kann.

Die mögliche Eingabe der vier Bytes für den 29-Bit-Identifier liegt damit zwischen 20,00,00,00 und 3F,FF,FF,FF.

Beispiel: In das Kommentarfeld soll der 29-Bit-Identifier 123456_h eingetragen werden.
Beachten Sie, das für 29-Bit Identifier das Bit 29 auf '1' gesetzt werden muss!
Die vier Bytes für den 29-Bit-Identifier müssen wie folgt in das Kommentar-Feld eingegeben werden: 20,12,34,56,

	Kommentar-Bytes bei **29-Bit** Identifiern							
Byte-Nr.	**1**	**2**	**3**	**4**	5	6	7	8 ... 15
Inhalt:	**CAN-Identifier**				Format-Byte	Zykluszeit optional bei Ausgängen		nicht belegt
	ID_UU	ID_UL	ID_LU	ID_LL				
	Bit31...Bit24	Bit23...Bit16	Bit15...Bit08	Bit07...Bit00	form	Cycle_high	Cycle_low	
Beispiel:	**20,**	**12,**	**34,**	**56,**	B8,	-	-	-
	(CAN-Identifier: 123456_h)				(Format-Byte: $B8_h$)	(kein zyklisches Senden)		-

Ist oben im Feld *E/A-Typ* ‘Eingang’ ausgewählt, so ist der hier eingetragene CAN-Identifier aus Sicht der SPS ein Rx-Identifier. Ist oben im Feld *E/A-Typ* ‘Ausgang’ ausgewählt, so ist der hier eingetragene CAN-Identifier ein Tx-Identifier.

Achtung!
Es darf kein Rx-Identifier mehrfach vorhanden sein !

Wenn unzulässigerweise zum Beispiel auf der SPS-Adresse 50 und auf der Adresse 51 der *gleiche* Rx-Identifier gewählt würde, so würden auf Adresse 50 keine neuen Rx-Daten eintreffen, sobald die Rx-Identifier-Zuweisung erfolgt ist. Die zuletzt empfangenen Daten würden unverändert erhalten bleiben.

Diese Rx-Identifier-Regel trifft auch für die im Communication-Window aktivierten Rx-Identifier zu.

5.2.2 Einstellung des Datenformats über das Steuerbyte *form*

Das Steuerbyte ***form*** wird im ***Kommentar***-Feld nach den ersten beiden (vier, bei 29-Bit-Identifier) Bytes für den **CAN-Identifier** übergeben. ***form*** dient zur Umwandlung der Nutzdaten vom Motorola-Format (high byte first) in das Intel-Format (low byte first).

Hintergrund: Nachrichten, die länger als 1 Byte sind, werden normalerweise auf einem CANopen-Netzwerk in Intel-Schreibweise übertragen, während die Siemens SPS im Motorola-Format arbeitet.

Beginnend bei Bit 7 des Format-Bytes kann entschieden werden, ob das nächstfolgende Byte mit umgewandelt, d.h. ‘geswapt’ werden soll oder nicht. Wird für ein Byte eine ‘1’ eingetragen, so erfolgt ein Vertauschen der folgenden Bytes bis einschließlich zur nächsten übergebenen ‘0’. Die Funktionsweise lässt sich am besten anhand eines Beispiels verdeutlichen.

Beispiel: Ein CAN-Telegramm enthält im ersten Wort ein Datum im Intel-Format, dann 2 Byte, die nicht gedreht werden sollen und in den letzten 4 Byte ein Langwort, wiederum im Intel-Format. Binär ergibt sich für das Format-Byte folgende Darstellung:

Bit-Nr.	7	6	5	4	3	2	1	0
Bit von *form*	**1**	**0**	**0**	**0**	**1**	**1**	**1**	**0**
hexa-dezimal	8				E			
Aktion	begin swap	end swap	unver-ändert	unver-ändert	begin swap	swap	swap	end swap

Daten-Bytes	1	2	3	4	5	6	7	8
CAN-Frame	2 Byte Intel-Format		Byte 3	Byte 4	4 Byte Intel-Format			
SPS-Daten	2 Byte Motorola-Format		Byte 3	Byte 4	4 Byte Motorola-Format			

Daraus ergibt sich das Format-Byte zu $8E_h$. Sollten z.B. alle 8 Bytes vertauscht werden, so müßte für das Format-Byte der Wert FE_h übergeben werden.

Das unterste Bit hat generell keine Bedeutung, da das Telegramm und damit die Formatierung abgeschlossen ist. Es sollte stets zu 0 gesetzt werden.

Hinweis:
Der Parameter *form* muss immer angegeben werden, auch wenn keine Datenumformatierung vorgenommen werden soll. In diesem Fall ist für *form* der Wert ‘00’ einzutragen.

5.2.3 Einstellung zum Zyklischen Senden über *Cycle*

Bei Ausgängen können im ***Kommentar***-Feld hinter dem Format-Byte *form* 2 Bytes eingetragen werden, die das CAN-DP anweisen, das in diesem Slot definierte CAN-Telegramm zyklisch alle *cycle* Millisekunden zu senden. Das zyklische Senden erfolgt dann unabhängig davon, ob sich das Telegramm geändert hat oder nicht.

Der Eintrag der beiden Bytes ***cycle*** für die Zykluszeit erfolgt im Feld *Kommentar* hinter den beiden (vier, bei 29-Bit-Identifiern) Bytes für den CAN-Identifier und dem darauf folgenden Steuerbyte *form*. Wenn das zyklische Senden nicht benötigt wird, entfallen die beiden Bytes *cycle* im *Kommentar*.
Das erste der beiden Bytes für das zyklische Senden ist das höherwertige Byte.
Das Datenformat im Feld *Kommentar* ist für alle Einträge **hexadezimal** (!).

Beispiel: Ein CAN-Identifier soll zyklisch alle 1 s gesendet werden.
Die Zykluszeit *cycle* muss hexadezimal in Millisekunden angegeben werden und dafür zunächst umgerechnet werden:
1s = 1000ms=$03E8_h$

Für die Zykluszeit *cycle* sind also 03_h und $E8_h$ als 4. und 5. Byte (bzw. als 6. und 7. Byte bei 29-Bit-Identifiern) in das Feld *Kommentar* einzutragen.

	Kommentar-Bytes bei **11-Bit** Identifiern					
Byte-Nr.	1	2	3	**4**	**5**	6 ... 15
Inhalt:	CAN-Identifier		Format-Byte	**Zykluszeit** optional bei Ausgängen		nicht belegt
	ID_high	ID_low	form	Cycle_high	Cycle_low	
Beispiel:	03,	09,	B8,	**03,**	**E8,**	-
	(CAN-Identifier: 0309_h)		(Format-Byte: $B8_h$)	(Zykluszeit: $03E8_h$)		-

	Kommentar-Bytes bei **29-Bit** Identifiern							
Byte-Nr.	1	2	3	4	5	**6**	**7**	8 ... 15
Inhalt:	CAN-Identifier				Format-Byte	**Zykluszeit** optional bei Ausgängen		nicht belegt
	ID_UU	ID_UL	ID_LU	ID_LL				
	Bit31...Bit24	Bit23...Bit16	Bit15...Bit08	Bit07...Bit00	form	Cycle_high	Cycle_low	
Beispiel:	20,	12,	34,	56,	B8,	**03,**	**E8,**	-
	(CAN-Identifier: 123456_h)				(Format-Byte: $B8_h$)	(Zykluszeit: $03E8_h$)		-

5.3 Das Communication-Window

5.3.1 Einführung

Werden die angeschlossenen CANopen-Module wie in Kapitel ‘5.1 Ablauf der Konfiguration’ angesprochen, so wird für jeden CAN-Identifier eine eigene SPS-Adresse benötigt. Das Communication Window bietet den Vorteil, daß eine einzelne SPS-Adresse für verschiedene Tx-Identifier und verschiedene Rx-Identifier verwendet werden kann. Dies ist möglich, weil die Identifier der anzusprechenden CANopen-Module bei jedem Zugriff gemeinsam mit den Daten als Parameter übergeben werden.

Nachteil des Communication-Windows ist der geringere Datendurchsatz. Der Einsatz des Communication-Windows empfiehlt sich daher für zeitunkritische Zugriffe, wie z.B. das Schreiben von SDOs in CANopen Netzen nach dem Start der Anlage.

Die Datenlänge in der Konfiguration muss immer 16 Byte betragen!

Als Identifier ist immer ‘FFEF’ hexadezimal einzusetzen !

Das Communication-Window wird auf den folgenden Seiten ausführlich beschrieben.

5.3.2 Konfiguration des Communication-Windows

Das Communication-Window wird über den PROFIBUS konfiguriert. Es ist je ein Eintrag für das Senden und ein Eintrag für das Empfangen von Daten über das Communication-Window notwendig. Mehr als diese beiden Einträge werden von der Firmware nicht akzeptiert.

Die folgenden beiden Bilder zeigen die notwendigen Einträge**. Bis auf die SPS-Adresse und die Einträge für die SYNC Time in den Kommentar-Bytes 4 und 5 sind alle Parameter vorgegeben.** Auch der Identifier ist nicht frei wählbar! Für die Konsistenz ist immer die gesamte Länge anzugeben! Für Ein- und Ausgangsrichtung sind eine gemeinsame SPS-Adresse oder verschiedene SPS-Adressen zulässig.

Datenrichtung:	**Eingang**
SPS-Adresse:	beliebig (Bsp. hier: 30)
Länge:	16 (immer)
Einheit:	Byte
Konsistent über:	gesamte Länge !
Identifier:	FFEF hexadezimal (immer)
Form-Byte:	00 hexadezimal

Datenrichtung:	**Ausgang**
SPS-Adresse:	beliebig (Bsp. hier: 30)
Länge:	16 (immer)
Einheit:	Byte
Konsistent über:	gesamte Länge !
Identifier:	FFEF hexadezimal (immer)
Form-Byte:	00 hexadezimal

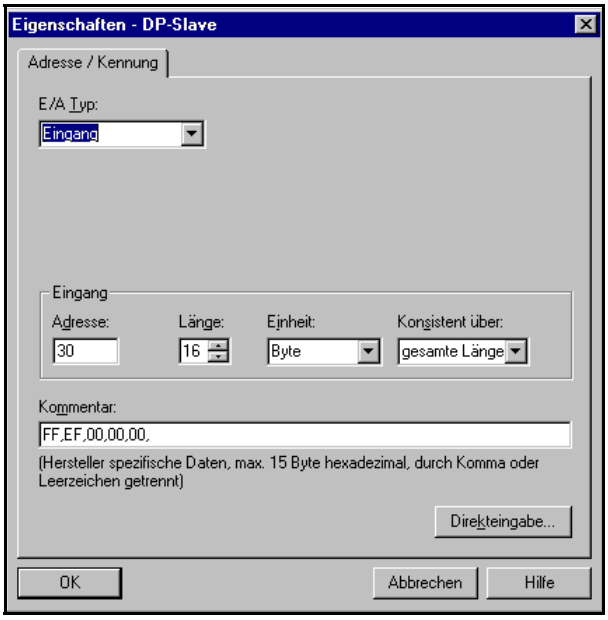


Abb. 5.3.1: Konfiguration des Eingangs-pfades des Communication-Windows

Abb. 5.3.2: Konfiguration des Ausgangs-pfades des Communication-Windows

5.3.3 Format des Communication-Windows

Die 16 Bytes des Communication-Windows sind je nach Datenrichtung unterschiedlich belegt.

5.3.3.1 Bytes des Communication-Windows beim Schreiben

(Kommandoaufruf und Senden von Daten SPS -> Gateway -> CAN)

Byte des Communication-Windows	Inhalt
0 1	High-Byte des CAN-Identifier (Identifier-Bit [15] 10...8) Low-Byte des CAN-Identifier (Identifier-Bit 7...0)
2 3	bei 11-Bit-Identifier Byte 2 und 3 immer ‘0’ bei 29-Bit CAN-Identifier Byte 2: Identifier-Bits 28...24 Byte 3: Identifier-Bits 23...16
4 5 6 7 8 9 10 11	Daten-Byte 0 Daten-Byte 1 Daten-Byte 2 Daten-Byte 3 Daten-Byte 4 Daten-Byte 5 Daten-Byte 6 Daten-Byte 7
12	Daten-Länge für Sendeaufträge (Tx)
13	SPS-Schleifenzähler (muss im Takt des OB1 inkrementiert werden, um dem Gateway den OB1-Zyklus mitzuteilen)
14	Sub-Command (immer auf ‘0’ setzen)
15	Command (Bedeutung siehe Seite 42)

Tabelle 5.3.1: Byte des Communication-Windows beim Schreiben

5.3.3.2 Bytes des Communication-Windows beim Lesen

(Kommando-Bestätigung und Empfang von Daten CAN -> Gateway -> SPS)

Byte des Communication-Windows	Inhalt
0 1	solange keine Empfangsdaten vorhanden ‘EEEE’$_h$, sonst High-Byte des CAN-Identifier (Identifier-Bit [15] 10...8) Low-Byte des CAN-Identifier (Identifier-Bit 7...0)
2 3	bei 11-Bit-Identifier Byte 2 und 3 immer ‘0’ bei 29-Bit CAN-Identifier Byte 2: Identifier-Bits 28...24 Byte 3: Identifier-Bits 23...16
4 5 6 7 8 9 10 11	Daten-Byte 0 Daten-Byte 1 Daten-Byte 2 Daten-Byte 3 Daten-Byte 4 Daten-Byte 5 Daten-Byte 6 Daten-Byte 7
12	Anzahl der empfangenen Daten-Bytes
13	Rückgabe des SPS-Schleifenzählers, der mit dem letzten PROFIBUS-Telegramm zum Gateway geschickt wurde
14	Rückgabe des Sub-Commands
15	Fehler-Code der Lesefunktion (wird zur Zeit nicht unterstützt)

Tabelle 5.3.2: Byte des Communication-Windows beim Lesen von CAN-Daten (Rx)

Die folgende Tabelle zeigt die Kommandos, die zur Zeit unterstützt werden. Das Sub-Command wird zur Zeit noch nicht ausgewertet und sollte beim Kommandoaufruf immer auf ‘0’ gesetzt werden.

Command	Funktion
1	Daten senden
3	Empfang auf freigeschalteten Rx-Identifiern
4	Freischalten von Rx-Identifiern für Empfang
5	Empfang (Command 4) deaktivieren
6	Senden eines RTR-Frames
7	führt Kommando 4 und Kommando 6 aus
(11)	(reserviert)
20	Wenn das Gateway als CANopen-Master konfiguriert ist: Zyklisches Senden des CANopen SYNC-Kommandos (ID 80_h, len = 0)

Tabelle 5.4.3 Kommandos des Communication-Windows

Achtung!
Ein Kommando ist erst dann vollständig abgearbeitet, wenn beim Lesen des Communication Windows in Byte 13 das CAN-DP/2-Modul den Wert des SPS-Schleifenzählers liefert, der beim Kommandoaufruf übergeben wurde.
Vor dem Aufruf des nächsten Kommandos sollte daher zunächst Byte 13 überprüft werden!

Erläuterungen zu den Kommandos:

Command 1: Daten senden

Zum Senden von Daten über das Communication-Window ist der CAN-Identifier in den Bytes 0 und 1 (bzw. 0...3 für 29-Bit-Identifier) einzutragen. Neben der Anzahl der zu sendenden Bytes ist außerdem ein SPS-Schleifenzähler einzutragen. Der Schleifenzähler ist vom Anwender zu realisieren. Er wird benötigt, um dem CAN-DP/2-Gateway den Status des OB1-Zyklus der SPS mitzuteilen.

Command 3: Empfang auf freigeschalteten Rx-Identifiern

Voraussetzung für den Empfang von Daten ist das Freischalten der CAN-Rx-Identifier, auf denen Daten empfangen werden sollen (siehe Command 4).
Nach dem Schreiben des Empfangskommandos 3 erhält man bei Lesezugriffen auf das Communication Window die auf Seite 41 gezeigte Datenstruktur. Die Rx-Daten werden asynchron zum SPS-Zyklus empfangen. Solange keine gültigen Daten eingetroffen sind, erhält man bei Lesezugriffen in den ersten Bytes den Wert ‘EEEE’$_h$ zurück. Erst wenn gültige Daten eingetroffen sind, wird der Rx-Identifier des gelesen Frames in den ersten Bytes lesbar. Über den zurückgegebenen SPS-Schleifenzähler in Byte 13 erfolgt außerdem die Zuordnung zum Lesekommando, das den Empfang der Daten angefordert hat.

Das Modul besitz einen FIFO-Speicher für 255 CAN-Frames zur Pufferung der empfangenen Rx-Daten. Sollten mehrere Rx-Frames auf einem Rx-Identifier eintreffen oder gehen Frames verschiedener zum Empfang freigeschalteter Rx-Identifier ein, so gehen die Daten nicht verloren, solange die SPS den FIFO-Speicher schneller ausliest als er gefüllt wird.

Command 4: Freischalten von Rx-Identifiern für den Empfang

Mit diesem Kommando muss der Rx-Identifier, dessen Daten empfangen werden sollen, freigeschaltet werden. Es dürfen auch mehrere Rx-Identifier gleichzeitig freigeschaltet sein. Hierzu ist das Kommando entsprechend oft aufzurufen.

Command 5: Empfang (Command 4) deaktivieren

Nach dem Aufruf dieses Kommandos werden auf den übergebenen Rx-Identifiern keine Daten mehr empfangen.

Command 6: Senden eines RTR-Frames

Hiermit wird ein Remote-Request-Frame gesendet. Vor dem Senden muss der Empfang auf dem Rx-Identifier mit Command 4 freigeschaltet werden.

Command 7: Führt Kommando 4 und Kommando 6 aus

Siehe dort.

Command 20: Zyklisches Senden des CANopen-Kommandos SYNC

Das CAN-DP/2-Modul kann für einfache CANopen-Anwendungen zyklisch das Kommando SYNC senden.
Das Kommando wird wie in der vorangegangenen Tabelle dargestellt gesendet.

Die Zykluszeit wird z.B. bei der Konfiguration des Communication-Windows im Eigenschaften-Fenster in den Bytes 4 und 5 übergeben (siehe Seite 39).
Die Eingabe erfolgt in Millisekunden.
Wertebereich: 0...$FFFE_h$ (0...65534 ms)

Achtung!
Damit sichergestellt ist, daß alle CANopen-Teilnehmer ihre neuen Daten erhalten haben wenn sie das SYNC-Kommando empfangen, kann der zyklische Sendeauftrag des SYNC-Kommandos das Senden des DP-Telegrams auf dem CAN-Bus nicht unterbrechen. Das heißt, trifft die Sendung eines DP-Telegramms zeitlich mit der Sendung des SYNC-Kommandos zusammen, wird das SYNC-Kommando verzögert, bis die Sendung des DP-Telegramms abgeschlossen ist.
Daher kann es zu geringen Zeitverschiebungen bei der zyklischen Sendung des SYNC-Kommandos kommen.

Achtung!
Die *SYNC Time* kann auf zwei verschiedene Arten gesetzt werden:
1. Im Parametrierungstelegramm des DP-Slave Eigenschaftenfensters (siehe Seite 27)
2. Über Byte 4 und 5 des Communication Windows (siehe Seite 39)

Die Einträge sind gleichberechtigt, d.h. der letzte erfolgte Eintrag ist gültig!

5.3.4 Beispiele zum Communication Window

5.3.4.1 Senden von Daten

1. Grundeinstellung des Communication Windows

Die Grundeinstellungen sind nur einmalig zur Einrichtung des Communication-Windows vorzunehmen.

1.1 Aktivierung des Communication Windows während der Konfiguration des DP-Gateways (siehe Seite 28)
Communication Window: yes

1.2 Definition der 16 Ein- und Ausgangs-Bytes des Communication Windows (siehe Seite 39) z.B.

Datenrichtung:	**Eingang**	Datenrichtung:	**Ausgang**
SPS-Adresse:	Bsp. hier: 30	SPS-Adresse:	Bsp. hier: 30
Länge:	16 (immer)	Länge:	16 (immer)
Einheit:	Byte	Einheit:	Byte
Konsistent über:	gesamte Länge !	Konsistent über:	gesamte Länge !
Identifier:	FFEF hexadezimal (immer)	Identifier:	FFEF hexadezimal (immer)
Form-Byte:	00 hexadezimal	Form-Byte:	00 hexadezimal

1.3 SPS-Schleifenzähler programmieren
8-Bit Schleifenzähler für Handshake-Funktion zwischen SPS und Gateway

	SPS-Zyklus (Pseudo Code):
...	...
1.	Lese Byte 13 (zurückgegebener Schleifenzähler) des ‘Comm..-Windows beim **Lesen**’ (siehe auch Seite 41)
2.	Vergleich des Byte 13 des ‘Comm.-Windows beim **Lesen**’ mit dem SPS-Schleifenzähler Byte 13 des ‘Comm.-Windows beim **Schreiben**’, wenn ungleich gehe zu 6., wenn gleich weiter bei 3.
3.	Erhöhe SPS-Schleifenzähler (Byte 13) aus ‘Bytes des Comm.-Windows beim **Schreiben**’ (siehe auch Seite 40)
4.	Auswertung der ‘Bytes des Comm.-Windows beim **Lesen**’ (siehe auch Seite 41), d.h. die Auswertung der Antwort auf den letzten Auftrag oder empfangenen CAN-Frame (Applikationsabhängig).
5.	Sende neues ‘Comm.-Window beim **Schreiben**’ (siehe auch Seite 40) mit erhöhtem Schleifenzählwert aus 3 und gegebenenfalls neuen Applikations-Daten.
6.	Setze SPS-Programm fort (neue Abfrage beim nächsten Programmdurchlauf)

2. Sendeauftrag starten durch Schreiben der 16 Bytes des Communication Windows

Byte des Communication-Windows	Inhalt	**Beispiel hier [hex]**
0 1	High-Byte des CAN-Identifier (Identifier-Bit [15] 10...8) Low-Byte des CAN-Identifier (Identifier-Bit 7...0)	**00** **12**
2 3	bei 11-Bit-Identifier Byte 2 und 3 immer ‘0’	**00** **00**
4 5 6 7 8 9 10 11	Daten-Byte 0 Daten-Byte 1 Daten-Byte 2 Daten-Byte 3 Daten-Byte 4 Daten-Byte 5 Daten-Byte 6 Daten-Byte 7	**00** **01** **02** **03** **04** **05** **06** **07**
12	Daten-Länge für Sendeaufträge (Tx)	**08**
13	SPS-Schleifenzähler	**8-Bit Counter**
14	Sub-Command (immer auf ‘0’ setzen)	**00**
15	Command ‘Daten senden’	**01**

Auf dem Tx-Identifier 0012_h werden die Daten-Bytes 00, 01, 02, 03, 04, 05, 06, 07 gesendet.

Zur Bestätigung der Kommandoausführung sollte anschließend ein Lesezugriff auf Byte 13 des Communication Windows erfolgen. Es muss den gleichen Wert des SPS-Schleifenzählers wie beim Kommandoaufruf enthalten.

5.3.4.2 Empfangen von Daten

1. Grundeinstellung des Communication Windows

Die Grundeinstellungen des Communication-Windows sind bereits im vorangegangenen Beispiel ‘Senden von Daten’ beschrieben worden.

2. Empfang von Daten

2.1 Freischalten des Rx-Identifiers für den Empfang

In diesem Beispiel sollen die Daten des Rx-Identifiers 0123_h empfangen werden.

Byte des Communication-Windows	Inhalt	**Beispiel hier [hex]**
0 1	High-Byte des CAN-Identifier (Identifier-Bit [15] 10...8) Low-Byte des CAN-Identifier (Identifier-Bit 7...0)	**01** **23**
2 3	bei 11-Bit-Identifier Byte 2 und 3 immer ‘0’	**00** **00**
4 5 6 7 8 9 10 11	Daten-Byte 0 Daten-Byte 1 Daten-Byte 2 Daten-Byte 3 Daten-Byte 4 Daten-Byte 5 Daten-Byte 6 Daten-Byte 7	**00** **00** **00** **00** **00** **00** **00** **00**
12	Daten-Länge für Sendeaufträge (Tx)	**00**
13	SPS-Schleifenzähler	**8-Bit Counter**
14	Sub-Command (immer auf ‘0’ setzen)	**00**
15	Command ‘Enable Rx-Identifier’	**04**

Zur Bestätigung der Kommandoausführung sollte anschließend an jeden Kommandoaufruf ein Lesezugriff auf Byte 13 des Communication Windows erfolgen. Es muss den gleichen Wert des SPS-Schleifenzählers wie beim Kommandoaufruf enthalten.

2.2 Empfang der Daten des freigeschalteten Rx-Identifiers einleiten

Byte des Communication-Windows	Inhalt	**Beispiel hier [hex]**
0 1	High-Byte des CAN-Identifier (Identifier-Bit [15] 10...8) Low-Byte des CAN-Identifier (Identifier-Bit 7...0)	**01** **23**
2 3	bei 11-Bit-Identifier Byte 2 und 3 immer ‘0’	**00** **00**
4 5 6 7 8 9 10 11	Daten-Byte 0 Daten-Byte 1 Daten-Byte 2 Daten-Byte 3 Daten-Byte 4 Daten-Byte 5 Daten-Byte 6 Daten-Byte 7	**00** **00** **00** **00** **00** **00** **00** **00**
12	Daten-Länge für Sendeaufträge (Tx)	**00**
13	SPS-Schleifenzähler	**8-Bit Counter + n**
14	Sub-Command (immer auf ‘0’ setzen)	**00**
15	Command ‘Read Rx-Identifier’	**03**

2.3 Lesen der Daten

Nach unbestimmter Zeit treffen die Rx-Daten ein und können mit einem Lesezugriff auf das Communication-Window ermittelt werden. Da das Eintreffen der Daten asynchron zu den SPS-Zyklen erfolgt, muss immer wieder das Communication Window gelesen werden, bis die Daten eingetroffen sind (polling). Ob es sich um die korrekten Daten handelt, die zu dem Lesekommando gehören, kann über einen Vergleich der Werte des SPS-Schleifenzählers ermittelt werden.

Ein Lesezugriff gibt folgende Bytes zurück:

Byte des Communication-Windows	Inhalt	**Beispiel hier [hex]**
0 1	High-Byte des CAN-Identifier (Identifier-Bit [15] 10...8) Low-Byte des CAN-Identifier (Identifier-Bit 7...0)	**01** **23**
2 3	bei 11-Bit-Identifier Byte 2 und 3 immer ‘0’	**00**
4 5 6 7 8 9 10 11	empfangenes Daten-Byte 0 empfangenes Daten-Byte 1 empfangenes Daten-Byte 2 empfangenes Daten-Byte 3 empfangenes Daten-Byte 4 empfangenes Daten-Byte 5 empfangenes Daten-Byte 6 empfangenes Daten-Byte 7	**AA** **BB** **CC** **DD** **EE** **FF** **00** **11**
12	Daten-Länge	**08**
13	SPS-Schleifenzähler	**8-Bit Counter + n**
14	zurückgegebenes Sub-Command (ohne Bedeutung)	**00**
15	Fehler-Code des Lesefunktion (ohne Bedeutung)	**00**

2.4 Empfang der Daten auf diesem Rx-Identifier deaktivieren

Sollen auf diesem Identifier keine weiteren Daten empfangen werden, sollte die Empfangsbereitschaft wieder aufgehoben werden.

Byte des Communication-Windows	Inhalt	**Beispiel hier [hex]**
0 1	High-Byte des CAN-Identifier (Identifier-Bit [15] 10...8) Low-Byte des CAN-Identifier (Identifier-Bit 7...0)	**01** **23**
2 3	bei 11-Bit-Identifier Byte 2 und 3 immer ‘0’	**00** **00**
4 5 6 7 8 9 10 11	Daten-Byte 0 Daten-Byte 1 Daten-Byte 2 Daten-Byte 3 Daten-Byte 4 Daten-Byte 5 Daten-Byte 6 Daten-Byte 7	**00** **00** **00** **00** **00** **00** **00** **00**
12	Daten-Länge für Sendeaufträge (Tx)	**00**
13	SPS-Schleifenzähler	**8-Bit Counter + m**
14	Sub-Command (immer auf ‘0’ setzen)	**00**
15	Command ‘Disable Rx-Identifier’	**05**

6. Page-Mode

Hinweis:
Der Page Mode kann nur genutzt werden, wenn als Konfigurations-Tool der Siemens SIMATIC Manager für S7 verwendet wird!

6.1 Eigenschaften

Der Page-Mode bietet die Möglichkeit, mehr CAN-Identifier zu adressieren, als in einem PROFIBUS-Telegramm untergebracht werden können (also mehr als 48). Die Anzahl der möglichen Identifier wird lediglich durch den verfügbaren Speicherplatz auf der SPS und dem CAN-Gateway eingeschränkt.
Auch mit dem Communication Window lassen sich mehr als 48 Identifier übertragen. Da beim Communication Window aber nur jeweils ein CAN-Frame pro SPS-Zyklus übertragen werden kann, ist es im wesentlichen für seltenere Zugriffe, wie z.B. einmalige Konfigurationen geeignet.

Das Handling des Page-Mode ist durch den zusätzlichen Protokollaufwand etwas komplizierter als beim Standardbetrieb des Gateways. Der Datenaustausch zwischen PROFIBUS und CAN benötigt aufgrund des notwendigen Handshakes zwei Zyklen statt einem SPS-Zyklus.

Um Ihnen die Bedienung des Page-Modes zu erleichtern, sind im Lieferumfang Funktionsbausteine und Datenbausteine enthalten, die den Page-Mode steuern.

6.2 Aktivierung

Vor der Aktivierung des Page-Modes müssen Sie die entsprechenden FB und DB in Ihr SPS-Programm integrieren. Bitte lesen Sie die folgenden Kapitel sorgfältig, um einen Einblick in die Funktionsweise zu bekommen und um die mitgelieferten FB und DB Ihren Anforderungen entsprechend einsetzen zu können.

Der Page-Mode wird mit dem SIMATIC-Manager (bei SIEMENS SPS, S7) aktiviert.

6.3 Communication-Window im Page-Mode

Über den Eintrag *Communication Window* kann während der Konfiguration des Gateways im DP-Slave-Eigenschaftenfenster das Communication Window aktiviert werden (siehe Seite 28).

Das Communication-Window wird dabei wie im normalen Betrieb eingerichtet und bedient (siehe Seite 38). Das Communication-Window muss jedoch zwingend im letzten Segment definiert werden.

Hinweis:
Die Nutzung des Communication Windows (*CW*) ist nur zur Konfiguration der angeschlossenen CAN-Geräte sinnvoll. Sind die angeschlossenen CAN-Teilnehmer konfiguriert, ist der normale Page-Mode (*PM*) vorzuziehen.

6.4 Funktionsweise

6.4.1 Übersicht

Um mehr CAN-Identifier versorgen zu können, als in einen PROFIBUS-Telegram untergebracht werden können, ist ein protokollgesteuerter Datenaustausch zwischen SPS und Gateway notwendig.

Für die Kommunikation werden sogenannte *Pages* definiert, in denen die Parameter und Daten übergeben werden. Auf der SPS-Seite wird ein Eingangsbereich und ein Ausgangsbereich für die Übertragung der Pages reserviert.
Nach dem Systemstart wird zunächst eine Page mit Initialisierungsdaten zwischen SPS und Gateway ausgetauscht. In den anschließenden Pages überträgt die SPS die Konfiguration der Tx- und Rx-Identifier. Diese Pages enthalten die für den CAN-Bus verwendeten Identifier-Nummern, die Anzahl der Bytes und Informationen zum Datenformat.

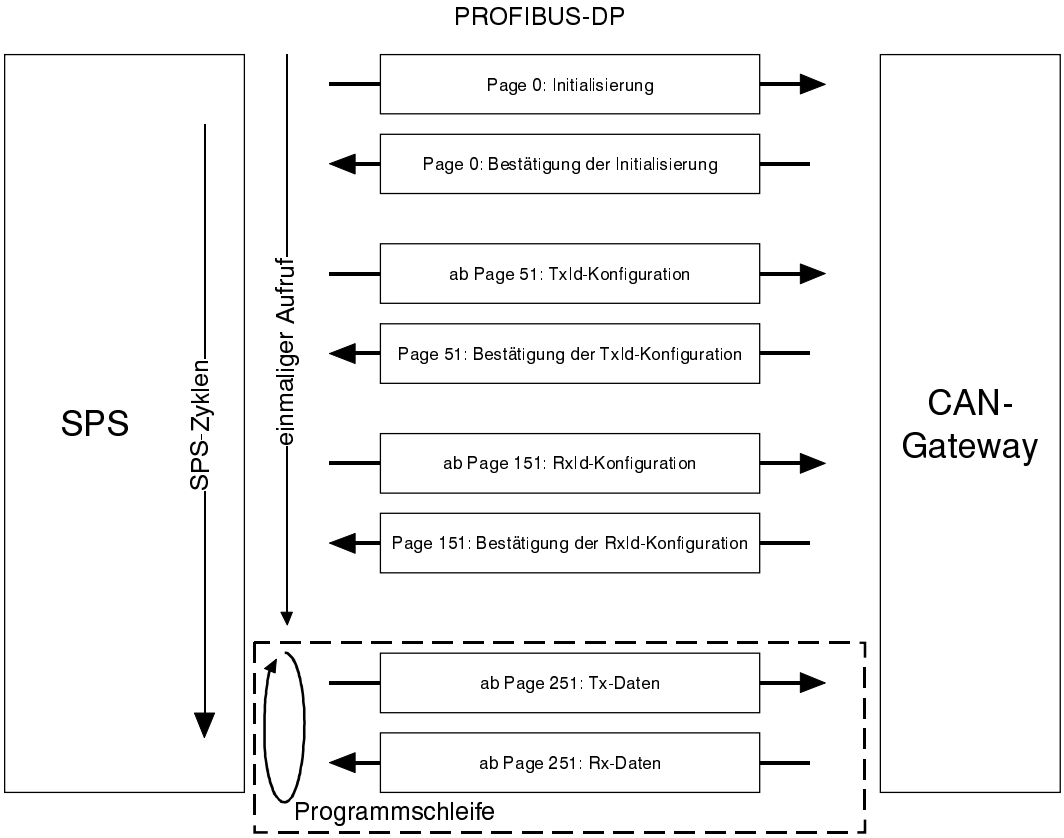


Abb. 6.4.1: Austausch von Parametern und Daten im Page-Mode (Übersicht)

Ist die Initialisierung abgeschlossen, kann der Datenaustausch erfolgen. Mit jedem SPS-Zyklus wird eine Eingangs- und eine Ausgangs-Page übergeben. Müssen mehr Identifier versorgt werden, als in einer Page untergebracht werden können, so werden in den nächsten SPS-Zyklen die nächsten Identifier bearbeitet. Mit steigender Anzahl von Identifiern und abhängig von der Länge der pro Identifier zu übertragenden Daten werden also mehr SPS-Zyklen benötigt, um alle Daten zu übertragen. Um die Anzahl der SPS-Zyklen klein zu halten, sollten die Eingangs- und Ausgangs-Page möglichst groß gewählt werden.

Unten ist ein Beispiel angefügt, in dem mit einer SIEMENS SIMATIC S7-SPS 127 Motoren angesteuert werden, die zusammen 127 Tx- und 127 Rx-Identifier belegen. In diesem Beispiel werden 20 SPS-Zyklen zur Versorgung aller Identifier benötigt (für 10 Pages werden 20 Zyklen benötigt).

Hinweis:
Im Lieferumfang sind Funktionsbausteine (FB) und Datenbausteine (DB) enthalten, mit denen die Übertragung der Pages gesteuert werden kann. Für den Anwender ist es daher nicht notwendig, die Steuerung der Pages selbst zu programmieren !

Die FB und DB werden ab Seite 61 beschrieben.

6.4.2 Definition der SPS-Adressen

Der Page-Mode benötigt Eingangs- und Ausgangsadressen. Die Anzahl der belegten Adressen ist nach oben nur durch die SPS eingeschränkt. Für die Eingänge muss mindestens eine Page-Größe von 32 Bytes vorgesehen werden, damit die Initialisierung über Page 0 und 1 durchgeführt werden kann.

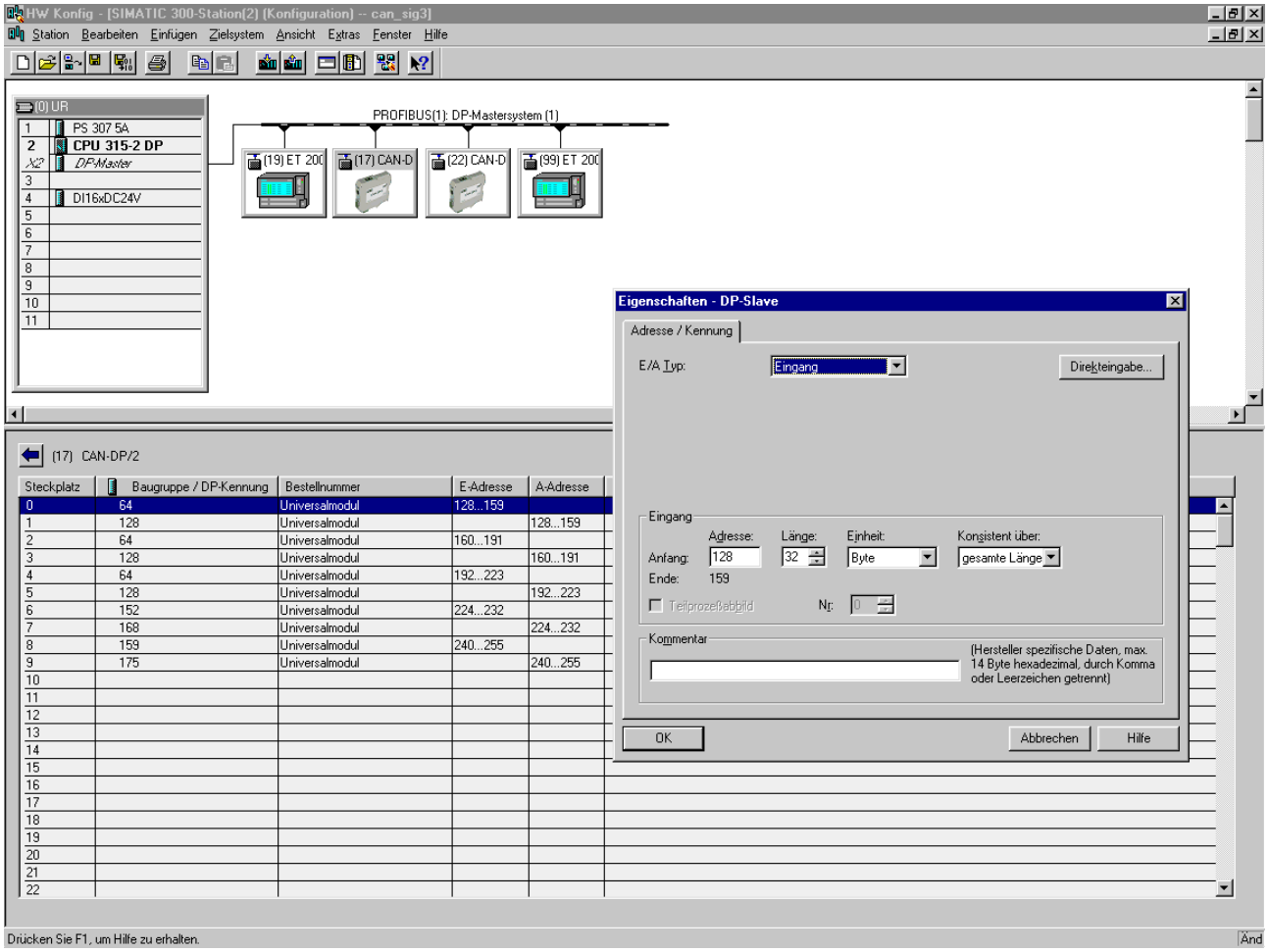


Abb. 6.4.2: Beispiel 1: Konfiguration der SPS-Adressen im Page-Mode

Beispiel 1:
In der vorangegangenen Abbildung ist die Belegung einer SIMATIC-S7-300 SPS für den Page-Mode dargestellt. Es sind 105 Bytes für den Page-Mode und 16 Byte für das Communication Window eingetragen. Damit ist die S7-300 voll belegt, da sie maximal 122 Bytes zur Verfügung stellt.

Im folgenden wird der Eintrag eines SPS-Steckplatzes als Page-*Segment* bezeichnet.

In Beispiel 1 ist für jedes Segment eine Datenlänge von 32 Bytes und die Konsistenz über die gesamten 32 Bytes eingestellt worden. Die Datenlänge wurde nicht größer gewählt, weil die S7-300 nicht mehr als 32 Bytes konsistent übertragen kann. Dies ist jedoch für den Page-Mode zwingend erforderlich.

Generell sollte ein Segment 32 Bytes groß gewählt werden. Vorausgesetzt, es sind bereits mindestens 32 Bytes für die Eingangsdaten eingetragen worden, so ist es für das letzte Segment auch zulässig, eine beliebige Länge zwischen 0 und 32 Bytes einzusetzen. Die Länge der Eingangsdaten darf sich von der Länge der Ausgangsdaten unterscheiden. Es ist jedoch zwingend erforderlich, daß die Eingangsadressen aufeinanderfolgender Steckplätze fortlaufend sind und daß die Ausgangsadressen aufeinanderfolgender Steckplätze fortlaufend sind.

Beispiel 2:
Für die Ausgangs-Page sind auf dem Steckplatz 0 ab Adresse 128 32 Bytes belegt worden. Steckplatz 1 ist ebenfalls mit 32 Bytes belegt und reicht somit von Adresse 160...191. Steckplatz 2 ist nur mit 18 Bytes belegt und reicht von Adresse 192...209. Es ergibt sich eine maximale Größe für die Ausgangs-Page von 82 Bytes.

In der folgenden Abbildung ist die Page im Adressraum der SPS dargestellt. Als Anwendungsbeispiel ist die Belegung mit der Tx-Konfigurationspage (Page 51) eingetragen. Bei einer Größe von 82 Bytes ließen sich in einer Page 11 Tx-Identifier konfigurieren. In den letzten vier Bytes wird die Endekennung eingetragen. Werden mehr Tx-Identifier benötigt, so werden anschließend die Tx-Pages 52, 53 usw. übertragen.

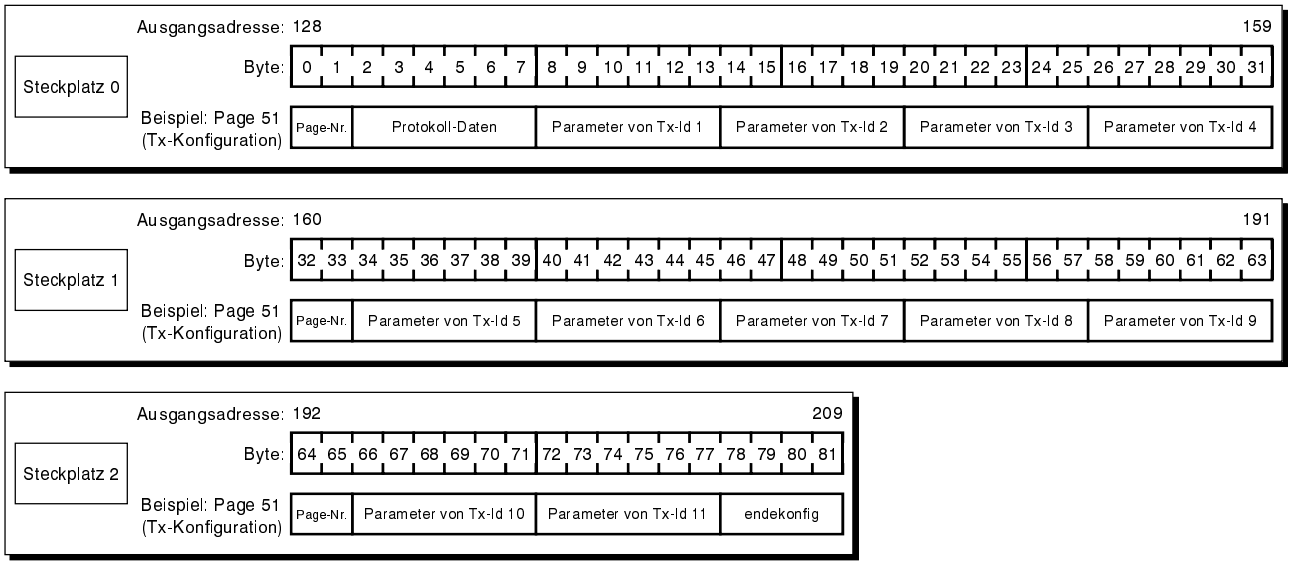


Abb. 6.4.4: Beispiel 2: Ausgangs-Page mit einer Länge von 82 Bytes

Wird der Page-Mode mit Communication-Window eingesetzt, so muss das Communication-Window zwingend in dem Segment definiert werden, das dem **letzten** SPS-Steckplatz zugeordnet ist.

Die folgende Tabelle zeigt zusammenfassend die Regeln für die Adressbelegung der SPS im Page-Mode:

Regeln für die Belegung der Adressen im Page-Mode	
1	Mindestens 32 Eingangsbytes definieren !
2	Segmentgröße immer = 32 Bytes ! Ausnahme: letztes Segment ≤ 32 Bytes !
3	Konsistenz immer über gesamte Länge !
4	Fortlaufende Adressierung der Segmente der Eingänge und der Ausgänge !
5	Communication-Window in die letzten beiden Segmente (falls gewünscht) !

Tabelle 6.4.1: Regeln für die Belegung der Adressen im Page-Mode

6.4.3 Struktur der Pages

Die maximale Länge der Page hängt von der Adreßkonfiguration ab, die der Anwender trifft (siehe Seite 52).

Die ersten acht Bytes enthalten bei allen Pages Informationen, die für den protokollgesteuerten Austausch der Pages zwischen SPS und Gateway benötigt werden. anschließend folgen die ‘Nutzdaten’ der Page, die während der Konfiguration z.B. die Definition der Identifiern oder im Betrieb die Daten der Identifier enthalten.

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	...
Länge [Bytes]	2		6						abhängig von Page-Nr.								...	
Inhalt	Page-Nr.		Protokoll-Daten						Nutzdaten								...	
Beispiel	51		...						z. B. Tx-Identifier-Definition								...	

Tabelle 6.4.2: Struktur der Pages

In den ersten beiden Bytes wird in jedem Segment einer Page die Page-Nummer eingetragen. Die Page-Nummer kennzeichnet die zu übertragende Page und den Typ der Page. In der folgenden Tabelle sind die Page-Nummern, Page-Typen und die zur Verfügung stehenden Funktions- und Datenbausteine dargestellt.

Page-Nummer	Page-Typ	Funktions-baustein	Datenbaustein	Formaloperand FREIGABE
0	Initialisierungspage	FB2	-	0
51...150	Tx-Konfiguration		DB94	1
151...250	Rx-Konfiguration		DB95	
≥ 251	Datenaustausch		Output: DB96 Input: DB97	

Tabelle 6.4.3: Übersicht der Pages

Auf den Inhalt der Bytes 3 bis 7, die ‘Protokolldaten’, wird hier nicht weiter eingegangen. Bitte verwenden Sie den im Lieferumfang enthaltenen Funktionsbaustein (FB2), um die Übertragung der Pages zu steuern. Sie enthalten die nötigen Kommandos für die Protokollsteuerung.

In den folgenden Kapiteln werden die Page-Typen beschrieben.

6.4.4 Initialisierung über Page 0 und 1

Nach einem Systemstart ist es zunächst erforderlich, daß das Gateway die Größe der zuvor konfigurierten Page an die SPS sendet. Dies erfolgt über die sogenannte *Page 1*.

Die SPS muss hierzu zunächst die *Page 0* an das Gateway senden. Das Gateway sendet daraufhin die Initialisierungsdaten in Page 1 zurück.

Im Lieferumfang ist ein Funktionsbaustein enthalten, der das Senden und empfangen der Pages 0 und 1 durchführt (FB2). Wir empfehlen, diesen Funktionsbaustein zu verwenden. Wird der Funktionsbaustein FB2 verwendet, so sind vom Anwender keine weiteren Parameter zu setzen.

Die Initialisierung benötigt etwas Zeit. Daher sollte mit dem Senden der nächsten Page für ca. 5 sec gewartet werden. Es kann z.B. ein SPS-Timer programmiert werden, der die Wartezeit einplant.

6.4.5 Tx-Konfiguration über Page 51...150

Die Konfiguration der Tx-Identifier erfolgt über Page 51 bis 150 (dezimal). Die Struktur der Page ist wie folgt:

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...
Länge [Bytes]	2		6						4				1	1	4				1	1	...	
Inhalt	Page-Nr.		Protokoll-Daten						*TxId_wert*				*form*	*länge*	*TxId_wert*				*form*	*länge*	...	
									Parameter von Tx-Identifier 1						Parameter von Tx-Identifier 2							
Beispiel	≥ 51		...						301				B8	6	303				B8	8	...	

Tabelle 6.4.4: Struktur der Pages 51...150

Die Bytes 0 bis 7 enthalten die bereits besprochenen Protokoll-Informationen (siehe auch Seite 55).

Ab Byte 8 im ersten Segment (ab Byte 3 in den folgenden Segmenten) wird die Definition der gewünschten Tx-Identifier an das CAN-Gateway übergeben. Für jeden Tx-Identifier werden hierzu 6 Bytes benötigt:

TxId_wert In diesen vier Bytes wird der Zahlenwert des Tx-Identifiers eingetragen.

form Mit diesem Byte kann ausgewählt werden, ob die Ausgangsdaten vom Motorola-Datenformat der SPS in das Intel-Datenformat des CAN-Netzes gewandelt werden sollen oder nicht. Das Byte *form* ist bereits auf Seite 36 ausführlich beschrieben worden.

länge Hier wird die Anzahl der Datenbytes des Tx-Identifiers übergeben. Zulässig sind Einträge von 1 bis 8.

6.4.6 Rx-Konfiguration über Page 151...250

Die Konfiguration der Rx-Identifier erfolgt über Page 151 bis 250 (dezimal). Die Struktur der Page ist wie folgt:

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...
Länge [Bytes]	2		6						4				1	1	4				1	1	...	
Inhalt	Page-Nr.		Protokoll-Daten						*RxId_wert*				*form*	*länge*	*RxId_wert*				*form*	*länge*	...	
									Parameter von Rx-Identifier 1						Parameter von Rx-Identifier 2							
Beispiel	≥ 151		...						301				B8	6	303				B8	8	...	

Tabelle 6.4.5: Struktur der Pages 151...250

Die Bytes 0 bis 7 enthalten die bereits besprochenen Protokoll-Informationen (siehe auch Seite 55).

Ab Byte 8 im ersten Segment (ab Byte 3 in den folgenden Segmenten) wird die Definition der gewünschten Rx-Identifier an das CAN-Gateway übergeben. Für jeden Rx-Identifier werden hierzu 6 Bytes benötigt:

RxId_wert In diesen vier Bytes wird der Zahlenwert des Rx-Identifiers eingetragen.

form Mit diesem Byte kann ausgewählt werden, ob die Eingangsdaten vom Intel-Datenformat des CAN-Netzes in das Motorola-Datenformat der SPS gewandelt werden sollen oder nicht. Das Byte *form* ist bereits auf Seite 36 ausführlich beschrieben worden.

länge Hier wird die maximale Anzahl der zu empfangenden Datenbytes des Rx-Identifiers übergeben. Zulässig sind Einträge von 1 bis 8.

6.4.7 Datenaustausch über Page 251...n

Das Lesen und Schreiben der Nutzdaten erfolgt über die Page 251 (dezimal) und folgende. Die maximale Anzahl der Daten-Pages beträgt 65285.

Die Struktur der Page für die Ausgangsdaten kann von der Struktur der Page der Eingangsdaten abweichen, weil die Anzahl der Rx-Daten von der Anzahl der Tx-Daten abweichen kann.

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...
Länge [Bytes]	2		6						1	1...8 (hier: 4)				1	1...8 (hier: 6)						...	
Inhalt	Page-Nr.		Protokoll-Daten						*force*	*Tx_Nutzdaten*				*force*	*Tx_Nutzdaten*						...	
									Daten von Tx-Identifier 1					Daten von Tx-Identifier 2								
Beispiel	≥ 251		...						2	12 34 56 78_h				1	BA 98 76 54 32 10_h						...	

Tabelle 6.4.6: Beispiel einer Daten-Page für Ausgangsdaten

Die Bytes 0 bis 7 enthalten die bereits besprochenen Protokoll-Informationen (siehe auch Seite 55).

Im ersten Segment werden die Daten des ersten Identifiers ab Byte 8 an das Gateway übergeben. Die Daten des nächsten Identifiers folgen direkt im Anschluss daran, d.h. es werden jeweils nur so viele Daten-Bytes pro Identifier übertragen, wie in *länge* definiert worden sind!
Im zweiten Segment werden die Daten der Identifier bereits ab Byte 2 übergeben, weil die Bytes 2 bis 7 hier nicht mit Protokoll-Informationen belegt sind.

force In diesem Byte kann eingetragen werden, wann die Tx-Daten gesendet werden sollen:

force	Sendung
0	Daten werden nicht als CAN-Frame ausgegeben
1	Daten werden immer (nach jedem PROFIBUS-Telegram) als CAN-Frame ausgegeben
2	Daten werden nur bei Veränderung der Daten als CAN-Frame ausgegeben
3	Daten werden einmal als CAN-Frame ausgegeben *)
4	Daten werden einmal als CAN-Frame ausgegeben *)

*) Wechsel zwischen 3 und 4 bewirkt eine gezielte Ausgabe der Daten

Tabelle 6.4.7: Festlegung der Ursache für das Senden von Tx-Daten

Tx_Nutzdaten Hier werden die zu sendenden Nutzdaten dieses Tx-Identifiers eingetragen.

Byte	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	...
Länge [Bytes]	2		6						1	1...8 (hier: 6)						1	1...8 (hier: 4)				...	
Inhalt	Page-Nr.		Protokoll-Daten						*count_in 1*	*Rx_Nutzdaten*						*count_in 2*	*Rx_Nutzdaten*				...	
									Daten von Rx-Identifier 1							Daten von Rx-Identifier 2						
Beispiel	≥ 251		...						14	11 22 33 44 55 66_h						15	99 88 77 66_h				...	

Tabelle 6.4.8: Beispiel einer Daten-Page für Eingangsdaten

count_in x In diesem Byte wird vom Gateway ein Eingangszähler eingetragen. Der Eingangszähler wird mit jedem eintreffenden Rx-Frame inkrementiert. Er kann vom Anwender z.B. für die Programmierung eines Guarding-Protokolls verwendet werden.

Rx_Nutzdaten Hier werden die empfangenen Nutzdaten dieses Rx-Identifiers eingetragen.

6.5 Einsatz des Page-Modes mit FBs und DBs

Im vorangegangenen Kapitel ist die prinzipielle Funktion des Page-Modes und die Belegung der Pages beschrieben worden, um die Funktionalität zu verdeutlichen. Im Lieferumfang des Gateways sind Funktionsbausteine und Datenbausteine als Quell-Code enthalten, die Sie in Ihr SPS-Programm einbinden sollten, wenn Sie den Page-Mode verwenden wollen.

6.5.1 Funktionsbaustein FB 2: Konfiguration und Datenaustausch

Mit dem Funktionsbaustein FB 2 können alle Konfigurationen und Datentransfers des Page-Modes ausgeführt werden. Die Typen der Datenbausteine, die FB 2 dabei verwendet, sind im folgenden Beispiel-Aufruf aufgeführt.

Aufruf von FB2 (Beispiel):

```
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
```

Erläuterung der Datenbausteine und Parameter:

Datenbaustein/Parameter	Funktion	Ausführliche Beschreibung siehe Seite
FREIGABE	Freigabe nach Initialisierung über Page 0 und 1.	62
WRITE_ADDRESS	Startadresse des ersten Ausgangssegmentes.	62
WRITE_CONFIG_DB	Datenbaustein zur Definition der Tx-Identifier.	62
WRITE_DB	Datenbaustein zum Schreiben der Ausgangsdaten	66
READ_ADDRESS	Startadresse des ersten Eingangssegmentes.	63
READ_CONFIG_DB	Datenbaustein zur Definition der Rx-Identifier.	64
READ_DB	Datenbaustein zum Lesen der Eingangsdaten.	68
RET_VALUE	Meldung über die Bearbeitung der aktuellen Page	69

Tabelle 6.5.1: Funktion der von FB2 verwendeten Datenbausteine

FREIGABE

Freigabe nach Initialisierung über Page 0 und 1.
Die Initialisierung erfolgt mit dem Funktionsbaustein FB2 mit dem Bit *FREIGABE* = 0. Für alle anderen Operationen ist es auf '1' zu setzen. FB2 benötigt zur Initialisierung lediglich einen Instanz-DB.

WRITE_ADDRESS

Startadresse des ersten Ausgangssegmentes.
Mit diesem Parameter wird der SPS die SPS-Startadresse des ersten Segmentes der Ausgangs-Page übergeben.

WRITE_CONFIG_DB (DB94)

Datenbaustein zur Definition der Tx-Identifier.
Im mitgelieferten SPS-Quell-Code ist der *WRITE-CONFIG-DB* als Datenbaustein DB94 umgesetzt worden.
Im *WRITE-CONFIG-DB* werden für jeden zu beschreibenden Tx-Identifier 6 Bytes benötigt:

Adresse	Byte 0...3	Byte 4	Byte 5	Bemerkung
0	*Tx-Identifier 1*	*form 1*	*länge 1*	Definition von Tx-Id 1
6	*Tx-Identifier 2*	*form 2*	*länge 2*	Definition von Tx-Id 2
12	*Tx-Identifier 3*	*form 3*	*länge 3*	Definition von Tx-Id 3
:	:	:	:	:
n · 6	*endekonfig*	-	-	Kennzeichnung des Endes des DBs oder der Tx-Konfiguration

Tabelle 6.5.2: Aufbau des *WRITE-CONFIG-DB*

Tx-Identifier x

Hier muss der Wert des Tx-Identifiers eingetragen werden.
11-Bit CAN ID: 0 bis 2047
29-Bit CAN-ID: 0 bis 536870911

form x

Im Parameter *form* wird Ausgewählt, ob die Ausgangsdaten vom Motorola-Datenformat der SPS in das Intel-Datenformat des CAN-Netzes gewandelt werden sollen oder nicht. Das Byte *form* ist bereits auf Seite 36 ausführlich beschrieben worden.

länge x

In dieses Bytes wird die Anzahl der Daten-Bytes, die auf dem hier definierten Tx-Identifier gesendet werden sollen, eingetragen.

endekonfig Der SPS muss mitgeteilt werden, ob für die Definition der Tx-Identifier ein weiterer Datenbaustein notwendig ist und wann die Tx-Definition abgeschlossen ist.

- Wird ein weiterer Datenbaustein benötigt, so muss als letzte Tx-Identifier-Definition der hexadezimale Wert ‘DDDDDDDD’ eingetragen werden. FB2 wird daraufhin mit der Bearbeitung des nächsten DBs fortfahren.
- Ist der letzte Tx-Identifier definiert worden, so wird dies dem FB2 über den Eintrag des hexadezimalen Wertes ‘EEEEEEEE’ als letzte Tx-Identifier-Definition mitgeteilt. FB2 wird danach mit der Konfiguration der Rx-Identifier fortfahren.

Die Länge der Datenbausteine ist variabel. Die benötigte Länge errechnet sich aus der Anzahl der benötigten Tx-Identifier zuzüglich der 4 Bytes für die Endekennung.

Beispiel:
Es sind 16 Tx-Identifier, beginnend mit DB11 zu definieren.

DB11 definiert die Tx-Ids 1 ... 10,
benötigt also eine Länge von $(10 \cdot 6 + 4) = 64$ Bytes
Endekennung = $DDDDDDDD_h$

DB12 definiert die Tx-Ids 11 ... 16,
benötigt also eine Länge von $(6 \cdot 6 + 4) = 40$ Bytes
Endekennung = $EEEEEEEE_h$

> **Hinweis:**
> Im FB2 muss zur Ausführung der Tx-Konfiguration das Bit `FREIGABE` = 1 gesetzt sein!

READ_ADDRESS **Startadresse des ersten Eingangssegmentes.**
Mit diesem Parameter wird der SPS die SPS-Startadresse des ersten Segmentes der Eingangs-Page übergeben.

READ_CONFIG_DB
(DB95)

Datenbaustein zur Definition der Rx-Identifier.
Im mitgelieferten SPS-Quell-Code ist der *READ-CONFIG-DB* als Datenbaustein DB95 umgesetzt worden.
Im *READ-CONFIG-DB* werden für jeden zu beschreibenden Rx-Identifier 6 Bytes benötigt:

Adresse	Byte 0...3	Byte 4	Byte 5	Bemerkung
0	*Rx-Identifier 1*	*form 1*	*länge 1*	Definition von Rx-Id 1
6	*Rx-Identifier 2*	*form 2*	*länge 2*	Definition von Rx-Id 2
12	*Rx-Identifier 3*	*form 3*	*länge 3*	Definition von Rx-Id 3
:	:	:	:	:
n · 6	*endekonfig*	-	-	Kennzeichnung des Endes des DBs oder der Rx-Konfiguration

Tabelle 6.5.3: Aufbau des *READ-CONFIG-DB*

Rx-Identifier x Hier muss der Wert des Rx-Identifiers eingetragen werden.
11-Bit CAN ID: 0 bis 2047
29-Bit CAN-ID: 0 bis 536870911

form x Im Parameter *form* wird Ausgewählt, ob die Eingangsdaten vom Intel-Datenformat des CAN-Netzes in das Motorola-Datenformat der SPS gewandelt werden sollen oder nicht. Das Byte *form* ist bereits auf Seite 36 ausführlich beschrieben worden.

länge x In dieses Bytes wird die Anzahl zu empfangenden Daten-Bytes dieses Rx-Identifiers eingetragen.

endekonfig Der SPS muss mitgeteilt werden, ob für die Definition der Rx-Identifier ein weiterer Datenbaustein notwendig ist und wann die Rx-Definition abgeschlossen ist.
- Wird ein weiterer Datenbaustein benötigt, so muss als letzte Rx-Identifier-Definition der hexadezimale Wert ‘DDDDDDDD’ eingetragen werden. FB2 wird daraufhin mit der Bearbeitung des nächsten DBs fortfahren.
- Ist der letzte Rx-Identifier definiert worden, so wird dies dem FB2 über den Eintrag des hexadezimalen Wertes ‘EEEEEEEE’ mitgeteilt. FB2 wird danach mit der Übertragung der Nutzdaten fortfahren.

Die Länge der Datenbausteine ist variabel. Die benötigte Länge errechnet sich aus der Anzahl der benötigten Rx-Identifier zuzüglich der 4 Bytes für die Endekennung.

Beispiel:
Es sind 19 Rx-Identifier, beginnend mit DB26 zu definieren.

DB26 definiert die Rx-Ids 1 ... 10,
benötigt also eine Länge von (10 · 6 + 4) = 64 Bytes
Endekennung = $DDDDDDDD_h$

DB27 definiert die Tx-Ids 11 ... 19,
benötigt also eine Länge von (9 · 6 + 4) = 58 Bytes
Endekennung = $EEEEEEEE_h$

Hinweis:
Im FB2 muss zur Ausführung der Rx-Konfiguration das Bit *FREIGABE* = 1 gesetzt sein!

WRITE_DB
(DB96)

Datenbaustein zum Schreiben der Ausgangsdaten

Die Ausgangsdaten werden im Datenbaustein geordnet nach der Identifier-Nummer (TxId1, TxId2, etc.) abgelegt. Für jeden Tx-Identifier werden in einem Byte die Länge (Anzahl der Daten-Bytes + *force*-Byte), in einem Byte der Parameter *force* und anschließend die Nutzdaten abgelegt. Die Anzahl der Nutzdaten kann von 1 bis 8 Byte variieren. Die Daten des folgenden Tx-Identifiers schließen immer direkt an die des vorhergehenden an. Die Adresse, ab der die Daten eines Tx-Identifiers abgelegt sind, muss also anhand der Daten der vorangegangenen Tx-Identifier errechnet werden.

Adresse	1 Byte	1 Byte	*n* Byte	Bemerkung
0	*länge 1*	*force 1*	*nutzdaten 1*	Nutzdaten von Tx-Id 1

Adresse	1 Byte	1 Byte	*m* Byte	Bemerkung
n + 2	*länge 2*	*force 2*	*nutzdaten 2*	Nutzdaten von Tx-Id 2

Adresse	1 Byte	1 Byte	*l* Byte	Bemerkung
n + 2 + *m* + 2	*länge 3*	*force 3*	*nutzdaten 3*	Nutzdaten von Tx-Id 3

:

Adresse	1 Byte	-	Bemerkung
xxx	*endedata*	-	Kennzeichnung des Endes des DBs oder der Ausgangsdaten

Tabelle 6.5.4: Aufbau des *WRITE_DB*

länge x In dieses Byte wird die Anzahl der Daten-Bytes , die auf dem hier definierten Tx-Identifier gesendet werden sollen, eingetragen (+1 für das *force*-Byte):

länge = (Anzahl der Daten-Bytes) + 1

force x

In diesem Byte kann vorgegeben werden, wann die Daten des Tx-Identifiers auf dem CAN-Bus gesendet werden sollen.

force	Funktion
0	Daten werden nicht als CAN-Frame ausgegeben.
1	Daten werden immer (nach jedem PROFIBUS-Telegramm) als CAN-Frame ausgegeben.
2	Daten werden nur bei Veränderung der Daten als CAN-Frame ausgegeben.
3	Daten werden nur einmal als CAN-Frame ausgegeben.
4	Daten werden nur einmal als CAN-Frame ausgegeben.

Tabelle 6.5.5: Bedeutung des Parameters *force*

Um den CAN-Frame mit den Nutzdaten einmalig zu senden, muss der Parameter *force* auf den Wert '3' gesetzt werden. Wird der Parameter im nächsten Zyklus wieder auf '3' gesetzt, erfolgt keine Sendung. Um wiederholt zu senden, muss *force* im Folgezyklus auf den Wert '4' gesetzt werden. Jedes weitere Umschalten zwischen den Werten führt zu einer Sendung des Frames.

nutzdaten x

Die Nutzdaten von 1 bis 8 Bytes werden im Anschluss an den Parameter *force* eingetragen.

endedata

Mit diesem Parameter wird der SPS mitgeteilt, ob ein weiterer Datenbaustein mit Nutzdaten folgt oder ob dies die letzen Nutzdaten waren, die gesendet werden sollen.

- Wird ein weiterer Datenbaustein benötigt, so muss hinter der Definition der letzten Nutzdaten dieses DBs für *länge* der hexadezimale Wert 'DD' eingetragen werden. FB2 wird daraufhin mit der Bearbeitung des nächsten DBs fortfahren.
- Sind die letzten Nutzdaten der Anwendung eingetragen, so wird dies dem FB2 über den Eintrag des hexadezimalen Wertes 'EE' in die Zelle *länge* mitgeteilt. FB2 wird danach mit der Übertragung der Nutzdaten des ersten `WRITE_DBs` fortfahren.

Hinweis:
Im FB2 muss zum Schreiben der Ausgangsdaten das Bit `FREIGABE` = 1 gesetzt sein!

READ_DB
(DB97)

Datenbaustein zum Lesen der Eingangsdaten.
Die Eingangsdaten werden im Datenbaustein geordnet nach der Identifier-Nummer (RxId1, RxId2, etc.) abgelegt. Für jeden Rx-Identifier werden in einem Byte die Länge (Anzahl der Daten-Bytes + *count_in*-Byte), in einem Byte der Eingangszähler *count_in* und anschließend die Nutzdaten abgelegt. Die Anzahl der Nutzdaten kann von 1 bis 8 Byte variieren. Die Daten des folgenden Rx-Identifiers schließen immer direkt an die des vorhergehenden an. Die Adresse, ab der die Daten eines Rx-Identifiers abgelegt sind, muss also anhand der Daten der vorangegangenen Rx-Identifier errechnet werden.

Adresse	1 Byte	1 Byte	*n* Byte	Bemerkung
0	*länge 1*	*count_in 1*	*nutzdaten 1*	Nutzdaten von Rx-Id 1

Adresse	1 Byte	1 Byte	*m* Byte	Bemerkung
n + 2	*länge 2*	*count_in 2*	*nutzdaten 2*	Nutzdaten von Rx-Id 2

Adresse	1 Byte	1 Byte	*l* Byte	Bemerkung
n + 2 + m + 2	*länge 3*	*count_in 3*	*nutzdaten 3*	Nutzdaten von Rx-Id 3

:

Adresse	1 Byte	-	Bemerkung
xxx	*endedata*	-	Kennzeichnung des Endes des DBs oder der Eingangsdaten

Tabelle 6.5.6: Aufbau des `WRITE_DB`

länge x — In dieses Byte wird die Anzahl der zu empfangenden Daten-Bytes dieses Rx-Identifiers (+1 für das *count_in*-Byte) eingetragen:

länge = (Anzahl der Daten-Bytes) + 1

count_in x — In diesem Byte wird vom Gateway ein Eingangszähler eingetragen. Der Eingangszähler wird mit jedem eintreffenden Rx-Frame inkrementiert. Er kann vom Anwender z.B. für die Programmierung eines Guarding-Protokolls verwendet werden.

nutzdaten x — Die Nutzdaten von 1 bis 8 Bytes werden im Anschluss an den Parameter *count_in* eingetragen.

endedata Mit diesem Parameter wird der SPS mitgeteilt, ob ein weiterer Datenbaustein mit Nutzdaten folgt oder ob dies die letzen Nutzdaten waren, die eingelesen werden sollen.

- Wird ein weiterer Datenbaustein benötigt, so muss hinter der letzten Nutzdaten dieses DBs in die Zelle *länge* der hexadezimale Wert ‘DD’ eingetragen werden. FB2 wird daraufhin mit der Bearbeitung des nächsten DBs fortfahren.
- Sind die letzten Nutzdaten der Anwendung eingetragen, so wird dies dem FB2 über den Eintrag des hexadezimalen Wertes ‘EE’ in die Zelle *länge* mitgeteilt. FB2 wird danach mit der Übertragung der Nutzdaten des ersten `READ_DBs` fortfahren.

Hinweis:
Im FB2 muss zum Lesen der Eingangsdaten das Bit `FREIGABE` = 1 gesetzt sein!

RET_VALUE **Meldung über die Bearbeitung der aktuellen Page.**
Dieser Parameter ist ‘0’, wenn das Bit *FREIGABE* = ‘0’ ist. Ist *FREIGABE* = ‘1’, so enthält `RET_VALUE` eine Nummer, die aussagt, welcher Page-Typ zur Zeit bearbeitet wird:

`RET_VALUE` (bei *FREIGABE* = 1)	Momentan übertragener Page-Typ
0	keine Page-Übertragung
1	reserviert
2	Tx-Konfiguration über Page 51...150
3	Rx-Konfiguration über Page 151...250
4	Daten-Page 251...n

Tabelle 6.5.7: Rückgabeparameter `RET_VALUE`

6.6 Vorgehensweise

Die folgende Auflistung gibt eine Step-by-step-Anleitung zur Konfiguration und zum Betrieb des Page-Modes.

1. Hardware-Konfiguration

1.1 SPS-Adressraum für den Page-Mode (Eingänge/Ausgänge) festlegen, d.h.
- Segmente konfigurieren: n · 32 Byte + x Byte (x ≤ 32)
- Adressen aufeinanderfolgend!

1.2 bei Bedarf: Communication Window (am Ende)

2. SPS-Programm

2.1 FB2 einbinden:

WRITE_ADDRESS:	Startadresse des 1. Ausgangssegmentes
READ_ADDRESS:	Startadresse des 1. Eingangssegments

2.2 Datenbausteine:

WRITE_CONFIG_DB:	erzeugen und vorbesetzen (Länge des DB ermitteln!)
READ_CONFIG_DB:	erzeugen und vorbesetzen (Länge des DB ermitteln!)
WRITE_DB:	erzeugen und während des Programms mit Daten versorgen
READ_DB:	erzeugen und während des Programms die Daten lesen

3. Weitere mitgelieferte FBs:

3.1 FB4: Datenaustausch über Communication-Window
3.2 FB1: Initialisierung CANopen-Modulen anhand einer Liste (*INIT_LIST_DB*, *INIT_DB*)
3.3 FB3: Steuerung von 127 einheitlichen CANopen-Devices

7. Editieren der GSD-Datei mit einem Texteditor

Das Modul sollte möglichst mit einer PROFIBUS-Konfigurations-Software, wie z.B. dem SIMATIC Manager konfiguriert werden. Jedoch unterstützt nicht jede PROFIBUS-Konfigurations-Software “Universalmodule” (siehe Kapitel “5. Konfiguration mit dem SIMATIC Manager”).
Werden keine Universalmodule unterstützt, kann die GSD-Datei über einen Texteditor angepasst werden.

Die Konfiguration eines Moduls erfolgt mit Hilfe eines Konfigurations-Frames, dessen Inhalt in der GSD-Datei eingetragen wird.

Der Konfigurationsframe wird in drei Oktetts aufgeteilt (siehe auch PROFIBUS-Spezifikation, Normative Part 8, Seite 738, Abb. 16):

Oktett 1: *Anzahl_der_herstellerspezifischen_Daten*
Oktett 2: *Anzahl _der_Ausgangs_oder_Eingangsbytes*
Oktett 3: *Herstellerspezifisches_Konfigurationsbyte*
:
:

Die Oktetts haben folgende Bedeutung:

Oktett 1: *Anzahl_der_herstellerspezifischen_Daten*

Da das CAN-DP/2 immer ein spezielles ID-Format nutzt, um ein angeschlossenes CAN-Modul darzustellen, hat das Identifier-Byte die folgende Struktur (siehe auch PROFIBUS-Specification-Normative-Part-8, Seite 737)

MSB LSB

Bit-Nr.:	7	6	5	4	3	2	1	0
Inhalt:	00: Leermodul 01: Eingang 10: Ausgang		immer 0	immer 0	Länge der herstellerspezifischen Daten: 0011 11-Bit Identifier 0101 29-Bit Identifier 0101 Communication Window			

Beispiel Oktett 1:

Bit-Nr.:	7	6	5	4	3	2	1	0
Inhalt:	1	0	0	0	0	0	1	1

= 0x83 Ausgang, 3 Bytes hersteller-spezifische Daten (11-Bit Identifier)

Oktett 2: ***Anzahl _der_Ein/Ausgangsbytes***

Oktett 2 gibt die Konsistenz, die Struktur (Byte/Word) und die Anzahl der Ein- bzw. Ausgangsbytes an.
Längenbytes des Output vom PROFIBUS Master aus betrachtet (Siehe auch PROFIBUS-Specification-Normative-Part-8, Seite 738)

MSB LSB

Bit-Nr.:	7	6	5	4	3	2	1	0
Inhalt:	Konsistent über 0: Byte oder Word 1: gesamte Länge	Längen-Format 0: Byte-Struktur 1: Word-Struktur	Anzahl der Inputs/Outputs					

Bit 5 4 3 2 1 0	Bedeutung
0 0 0 0 0 0	1 Byte, bzw. 1 Word
:	:
1 1 1 1 1 1	64 Bytes, bzw. 64 Words

Beispiel Modul 1:

Bit-Nr.:	7	6	5	4	3	2	1	0
Inhalt:	0	0	0	0	0	1	0	1

= 0x05 6 Bytes Daten

Oktett 3, 4, 5: ***Herstellerspezifisches_Konfigurationsbyte***

Oktett 3 und Oktett 4 = CAN-Identifier Beispiel: Identifier 0x0203
Oktett 5 = Form Byte

Beispiel Modul 1

Der Konfigurations-Frame für das Modul 1 hat die folgende Struktur und muss in die GSD-Datei eingefügt werden.

Beispiel für manuell vorgenommene GSD-Datei-Einträge:

```
...

Module="Name des Moduls" 0x83, 0x05, 0x02, 0x03, 0x00
EndModule

...
```

Bedeutung der Einträge unter "Name des Moduls":

"Name des Moduls" ...	Kommentar zur Benennung des Moduls		
0x83...	Modul ist ein Ausgang (0x80) und es folgen drei herstellerspezifische Konfigurationsbytes (0x03).		
0x05...	Konsistent über Byte, Längenangabe in Bytes (0x05) und es werden 6 Bytes Daten übertragen (0x05 = 6...1).		
0x02...	Herstellerspezifische Daten:	0x02	
0x03...	Herstellerspezifische Daten:	0x03	Identifier = 0x0203 (z.B.: CANopen Rx PDO für Modul-ID 3)
0x00...	Herstellerspezifische Daten:	0x00	Kein Byte-Swapping, d.h. die Reihenfolge der Daten wird nicht geändert

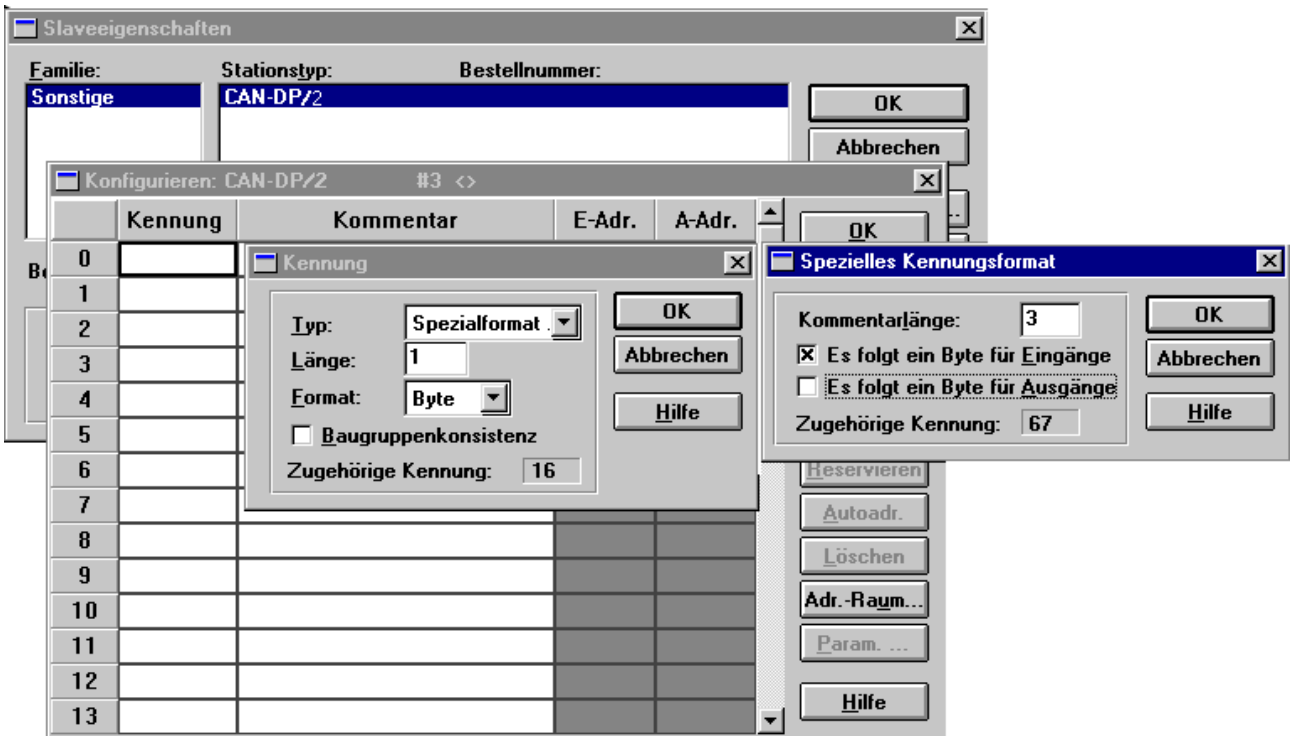
Achtung!
Bitte beachten Sie, dass Sie die GSD-Datei umbenennen müssen. Der Dateiname darf maximal 8 Zeichen lang sein. Einige Konfigurations-Software für den Profibus Master arbeitet nicht mit längeren Dateinamen.

8. Beispiele

8.1 Beispiel-SPS SIMATIC S5-95: Besonderheit bei ‘COM-PROFIBUS’

Wird das CAN-Gateway an einer SIMATIC S5-95 oder einer anderen S5-Einheit betrieben, so ist eine besondere Vorgehensweise notwendig, da das Konfigurationsprogramm ‘COM-PROFIBUS’ die Kommentare nicht mit in den PROFIBUS-String übernimmt. Um trotzdem die notwendigen Informationen (ID und Format) an das Gateway übergeben zu können, ist wie folgt vorzugehen:

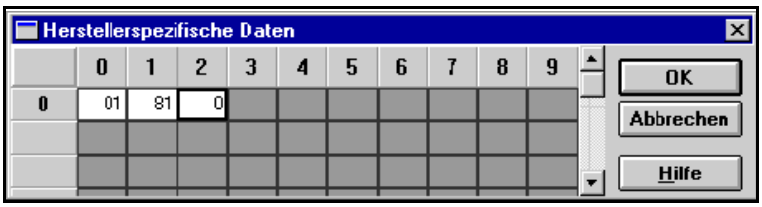
1. Nach Aufruf des CAN-DP/2-Moduls das Fenster *Konfigurieren* öffnen



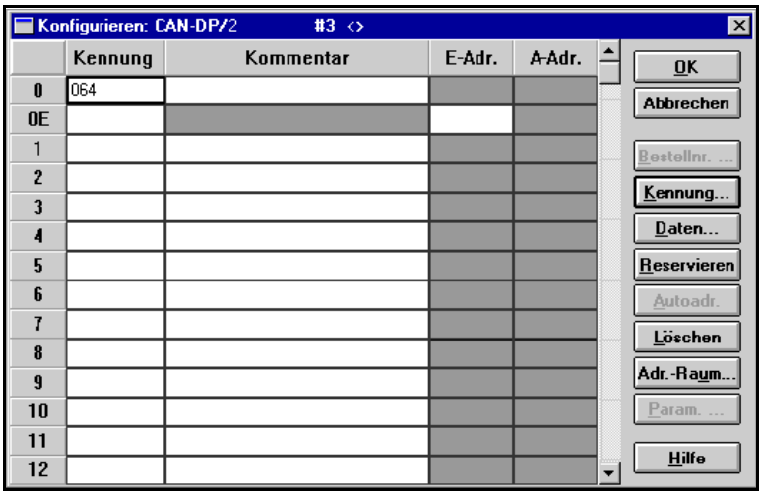
2. Dort den Button *Kennung* drücken und in dem sich öffnenden Fenster als *Typ* ‘**Spezialformat**’ eintragen. **Es muss Spezialformat gewählt werden, weil sonst der Kommentar nur als Text ausgewertet wird !** Die Einträge für die *Länge* und das *Format* können beliebig gewählt werden, da sie im folgenden nicht weiter ausgewertet werden.

3. Nach dem Drücken von *OK* öffnet sich das Fenster *Spezielles Kennungsformat*. Dort muss immer für die *Kommentarlänge* ‘3’ für 11-Bit-Identifier, bzw. ‘5’ für 29-bit Identifier oder Communication Window, eingetragen werden. Außerdem muss die Port-Richtung festgelegt werden.

4. Nach dem Drücken von *OK* öffnet sich das rechts abgebildete Fenster, in dessen Bytes 0 und 1 der CAN-Identifier und in Byte 3 der Parameter *form* (siehe Seite 36) eingetragen werden.

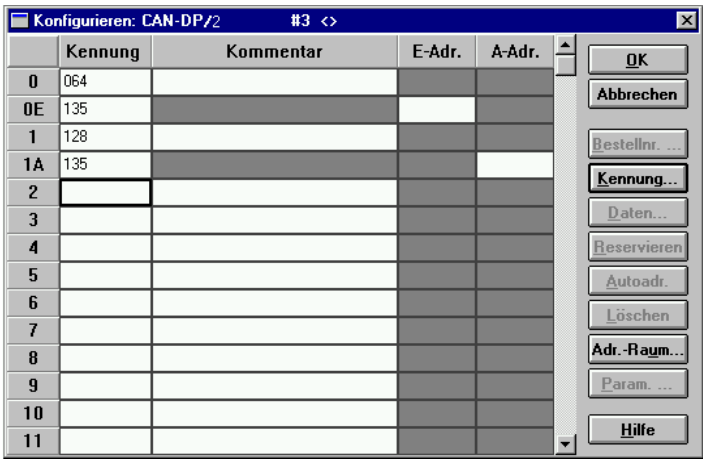


Danach erfolgt der Eintrag der Kennung (hier 67) in das Fenster *Konfigurieren*.



5. Jetzt muss der Cursor in den folgenden Eintrag (hier 0E) gesetzt werden und wiederum der Button *Kennung* gedrückt werden. In dem sich öffnenden Fenster (siehe rechts) wird für den vorher definierten Eingang z.B. die *Länge* '8' und das *Format* 'Byte' eingetragen. Falls gewünscht, kann die *Baugruppenkonsistenz* aktiviert werden.

6. Das nächste CAN-Modul, für das ein Eintrag erfolgen soll, könnte beispielsweise einen Ausgang erfordern. In diesem Fall müßten sich, 8 Byte Datenlänge und *Baugruppenkonsistenz* angenommen, die folgenden Einträge ergeben:



8.2 Beispiel-Applikation mit Page-Mode

Das folgende Beispiel zeigt die FBs und DBs zur Steuerung von 127 gleichartigen CANopen-Devices (hier Motoren).

```
Beispiel fuer den Aufruf des FB 1
---------------------------------
( Initialisierung von CANopen-Modulen anhand einer Liste (s.u.) )

Netzwerk 7: Motoren initialisieren
-----------
      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                      // Motornummer
      L     W#16#600
      +I
      T     #TX_ID
      L     #MOTOR                      // Motornummer
      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 initialization
      ==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 initialization
      T     DBW [#t000]                 // save in init-list-DB
      L     W#16#FFFF                   // means: configuration in FB1 allways runs
      L     #t016                       // status of initialization
      ==I
      SPB   M014
      L     W#16#FFFE                   // means: configuration in FB1 is ready
      L     #t016                       // status of initialization
      ==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

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

Aufruf des FB 2: Datenaustausch ueber Page-Modus
------------------------------------------------
(
  1. : Page  0     -> Einlesen der Laengen (unbedingt notwendig !!!)
  2. : Page  51 ff -> Tx-Konfiguration ( einmal )
  3. : Page 151 ff -> Rx-Konfiguration ( einmal )
  4. : Page 251 ff -> Datenaustausch: output und input ( zyklisch )
         ( Page 251, 252, 253, ... xyz, 251, 252, ... xyz (je nachdem wieviele Pages notwendig sind)
)
)

Netzwerk 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

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

Aufruf des FB 4: Befehle ueber das Kommunikationsfenster absetzen
-----------------------------------------------------------------
(nach Abschluss der Initialisierung mit FB 1, da FB 1 auch ueber das Kommunikationsfenster arbeitet)

Netzwerk 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 von Motor 1
      T     MW     0
      L     0
      T     MW     4
      T     MB    12
      T     MB    14                   // subcommand
      L     4
      T     MB    15                   // command = 4: Empfang aktivieren
      SPA   M499
M404: U     M     95.4
      SPB   M405
      L     W#16#18A                   // 1.PDO von Motor 10
      T     MW     0
      L     0
      T     MW     4
      T     MB    12
      T     MB    14                   // subcommand
      L     4
      T     MB    15                   // command = 4: Empfang aktivieren
      SPA   M499
M405: U     M     95.5
      SPB   M406
      L     W#16#183                   // 1.PDO von Motor 3
      T     MW     0
      L     0
      T     MW     4
      T     MB    12
      T     MB    14                   // subcommand
      L     4
      T     MB    15                   // command = 4: Empfang aktivieren
      SPA   M499
M406: U     M     95.6
      SPB   M407
      L     W#16#184                   // 1.PDO von Motor 4
      T     MW     0
      L     0
      T     MW     4
      T     MB    12
      T     MB    14                   // subcommand
      L     4
      T     MB    15                   // command = 4: Empfang aktivieren
      SPA   M499
M407: U     M     95.7
      SPB   M408
      L     W#16#187                   // 1.PDO von Motor 7
      T     MW     0
      L     0
      T     MW     4
      T     MB    12
      T     MB    14                   // subcommand
      L     4
      T     MB    15                   // command = 4: Empfang aktivieren
      SPA   M499
M408: U     M     96.0
      SPB   M499
      L     W#16#188                   // 1.PDO von Motor 8
      T     MW     0
      L     0
      T     MW     4
      T     MB    12
      T     MB    14                   // subcommand
      L     4
      T     MB    15                   // command = 4: Empfang aktivieren
      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


Netzwerk 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



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

Aufruf des FB3
=======================
( im FB 3 werden der FB 1 ( Motorinitialisierung ) und FB 2 ( Datenaustausch ) aufgerufen )

      CALL  FB     3 , DB103
       FREIGABE :=E125.7
       KONFIG_DB:=W#16#5D               // DB93: Motor 1,3,4,7,8 ; 1,4,8 initialisieren
       DATEN_DB :=W#16#64
       RET_VALUE:=MW16
      L     MW    16
      L     2
      >=I
      =     M     99.7
                  // Initialisierung der Motoren fertig => Kommunikationsfenster kann benutzt werden
```

```
DB93: KONFIG_DB = Konfigurations-Datenbaustein
==============================================
Adresse  Name  Typ   Anfangswert  Aktualwert  Kommentar
-------  ----  ---   -----------  ----------  ---------
    0.0  v000  WORD  W#16#5E      W#16#5E     Nummer des 1.DB mit Tx-Konfiguration
    2.0  v002  WORD  W#16#2FE     W#16#2FE    Laenge der DBs mit Tx-Konfiguration
    4.0  v004  WORD  W#16#0       W#16#0      Anzahl der DBs mit Tx-Konfiguration
    6.0  v006  WORD  W#16#5F      W#16#5F     Nummer des 1.DB mit Rx-Konfiguration
    8.0  v008  WORD  W#16#2FE     W#16#2FE    Laenge der DBs mit Rx-Konfiguration
   10.0  v010  WORD  W#16#0       W#16#0      Anzahl der DBs mit Rx-Konfiguration
   12.0  v012  WORD  W#16#60      W#16#60     Nummer des 1.DB mit Ausgangs-Daten
   14.0  v014  WORD  W#16#3F9     W#16#3F9    Laenge der DBs mit Ausgangs-Daten
   16.0  v016  WORD  W#16#0       W#16#0      Anzahl der DBs mit Ausgangs-Daten
   18.0  v018  WORD  W#16#61      W#16#61     Nummer des 1.DB mit Eingangsdaten
   20.0  v020  WORD  W#16#3F9     W#16#3F9    Laenge der DBs mit Eingangs-Daten
   22.0  v022  WORD  W#16#0       W#16#0      Anzahl der DBs mit Eingangs-Daten
   24.0  v024  WORD  W#16#80      W#16#80     SPS-Startadresse der Ausgabe-Page
   26.0  v026  WORD  W#16#80      W#16#80     SPS-Startadresse der Eingabe-Page
   28.0  v028  WORD  W#16#F0      W#16#F0     SPS-Startadresse des Ausgabe-Kommunikations-Fensters
   30.0  v030  WORD  W#16#F0      W#16#F0     SPS-Startadresse des Eingabe-Kommunikations-Fensters
   32.0  v032  WORD  W#16#62      W#16#62     Nummer des DB mit Init-Liste


Motor 1, 3, 4, 7 und 8 einrichten ==> DB 94, DB 95, DB 96 und DB 97

DB94: TX_KONFIG = Datenbaustein mit Konfiguration der Tx-Identifier
===================================================================
Adresse     Name      Typ     Anfangswert   Aktualwert      Kommentar
-------     ----      ---     -----------   ----------      ---------
    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 (Ende-Kennung)

            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_KONFIG = Datenbaustein mit Konfiguration der Rx-Identifier
===================================================================
Adresse     Name      Typ     Anfangswert   Aktualwert      Kommentar
-------     ----      ---     -----------   ----------      ---------
    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 = Datenbaustein fuer die Ausgangsdaten
```

```
=====================================================
Adresse     Name       Typ      Anfangswert   Aktualwert      Kommentar
-------     ----       ---      -----------   ----------      ---------
    0.0     laenge1    BYTE     B#16#0        B#16#7          Laenge
    1.0     force1     BYTE     B#16#0        B#16#2          Force-Byte
    2.0     data01     BYTE     B#16#0        B#16#0          Datenbyte 0
    3.0     data11     BYTE     B#16#0        B#16#0          Datenbyte 1
    4.0     data21     BYTE     B#16#0        B#16#0          Datenbyte 2
    5.0     data31     BYTE     B#16#0        B#16#0          Datenbyte 3
    6.0     data41     BYTE     B#16#0        B#16#0          Datenbyte 4
    7.0     data51     BYTE     B#16#0        B#16#0          Datenbyte5
    8.0     laenge2    BYTE     B#16#0        B#16#7          Laenge
    9.0     force2     BYTE     B#16#0        B#16#1          Force-Byte
   10.0     data02     BYTE     B#16#0        B#16#0          Datenbyte 0
   11.0     data12     BYTE     B#16#0        B#16#0          Datenbyte 1
            data22     BYTE     B#16#0        B#16#0          Datenbyte 2
            data32     BYTE     B#16#0        B#16#0          Datenbyte 3
            data42     BYTE     B#16#0        B#16#0          Datenbyte 4
            data52     BYTE     B#16#0        B#16#0          Datenbyte5
            laenge3    BYTE     B#16#0        B#16#7          Laenge
            force3     BYTE     B#16#0        B#16#4          Force-Byte
            data03     BYTE     B#16#0        B#16#0          Datenbyte 0
            data13     BYTE     B#16#0        B#16#0          Datenbyte 1
            data23     BYTE     B#16#0        B#16#0          Datenbyte 2
            data33     BYTE     B#16#0        B#16#0          Datenbyte 3
            data43     BYTE     B#16#0        B#16#0          Datenbyte 4
            data53     BYTE     B#16#0        B#16#0          Datenbyte5
            laenge4    BYTE     B#16#0        B#16#7          Laenge
            force4     BYTE     B#16#0        B#16#0          Force-Byte
            data04     BYTE     B#16#0        B#16#0          Datenbyte 0
            data14     BYTE     B#16#0        B#16#0          Datenbyte 1
            data24     BYTE     B#16#0        B#16#0          Datenbyte 2
            data34     BYTE     B#16#0        B#16#0          Datenbyte 3
            data44     BYTE     B#16#0        B#16#0          Datenbyte 4
            data54     BYTE     B#16#0        B#16#0          Datenbyte5
            laenge5    BYTE     B#16#0        B#16#7          Laenge
            force5     BYTE     B#16#0        B#16#3          Force-Byte
            data05     BYTE     B#16#0        B#16#0          Datenbyte 0
            data15     BYTE     B#16#0        B#16#0          Datenbyte 1
            data25     BYTE     B#16#0        B#16#0          Datenbyte 2
            data35     BYTE     B#16#0        B#16#0          Datenbyte 3
            data45     BYTE     B#16#0        B#16#0          Datenbyte 4
            data55     BYTE     B#16#0        B#16#0          Datenbyte5
   40.0     laenge6    BYTE     B#16#0        B#16#EE         Laenge
            force6     BYTE     B#16#0        B#16#0          Force-Byte


DB97: INPUT_DB = Datenbaustein fuer die Eingangsdaten
=====================================================
Adresse     Name               Typ     Anfangswert    Aktualwert     Kommentar
-------     ----               ---     -----------    ----------     ---------
    0.0     laenge1            BYTE    B#16#0         B#16#7         Laenge
    1.0     zaehler1           BYTE    B#16#0         B#16#0         Zaehler
    2.0     data01             BYTE    B#16#0         B#16#0         Datenbyte 0
    3.0     data11             BYTE    B#16#0         B#16#0         Datenbyte 1
    4.0     data21             BYTE    B#16#0         B#16#0         Datenbyte 2
    5.0     data31             BYTE    B#16#0         B#16#0         Datenbyte 3
    6.0     data41             BYTE    B#16#0         B#16#0         Datenbyte 4
    7.0     data51             BYTE    B#16#0         B#16#0         Datenbyte 5
    8.0     laenge2            BYTE    B#16#0         B#16#7         Laenge
            zaehler2           BYTE    B#16#0         B#16#0         Zaehler
            data02             BYTE    B#16#0         B#16#0         Datenbyte 0
            data12             BYTE    B#16#0         B#16#0         Datenbyte 1
            data22             BYTE    B#16#0         B#16#0         Datenbyte 2
            data32             BYTE    B#16#0         B#16#0         Datenbyte 3
            data42             BYTE    B#16#0         B#16#0         Datenbyte 4
            data52             BYTE    B#16#0         B#16#0         Datenbyte 5
            laenge3            BYTE    B#16#0         B#16#7         Laenge
            zaehler3           BYTE    B#16#0         B#16#0         Zaehler
            data03             BYTE    B#16#0         B#16#0         Datenbyte 0
            data13             BYTE    B#16#0         B#16#0         Datenbyte 1
            data23             BYTE    B#16#0         B#16#0         Datenbyte 2
            data33             BYTE    B#16#0         B#16#0         Datenbyte 3
            data43             BYTE    B#16#0         B#16#0         Datenbyte 4
            data53             BYTE    B#16#0         B#16#0         Datenbyte 5
            laenge4            BYTE    B#16#0         B#16#7         Laenge
            zaehler4           BYTE    B#16#0         B#16#0         Zaehler
            data04             BYTE    B#16#0         B#16#0         Datenbyte 0
            data14             BYTE    B#16#0         B#16#0         Datenbyte 1
```

```
          data24            BYTE    B#16#0         B#16#0         Datenbyte 2
          data34            BYTE    B#16#0         B#16#0         Datenbyte 3
          data44            BYTE    B#16#0         B#16#0         Datenbyte 4
          data54            BYTE    B#16#0         B#16#0         Datenbyte 5
          laenge5           BYTE    B#16#0         B#16#7         Laenge
          zaehler5          BYTE    B#16#0         B#16#0         Zaehler
          data05            BYTE    B#16#0         B#16#0         Datenbyte 0
          data15            BYTE    B#16#0         B#16#0         Datenbyte 1
          data25            BYTE    B#16#0         B#16#0         Datenbyte 2
          data35            BYTE    B#16#0         B#16#0         Datenbyte 3
          data45            BYTE    B#16#0         B#16#0         Datenbyte 4
          data55            BYTE    B#16#0         B#16#0         Datenbyte 5
   40.0   laenge6           BYTE    B#16#0         B#16#EE        Laenge
          zaehler6          BYTE    B#16#0         B#16#0         Zaehler


DB98: INIT_LIST_DB = Datenbaustein mit der Liste der Motoren, die  vorhanden sind / initialisiert werden
==========================================================================================
Adresse  Name          Typ   Anfangswert  Aktualwert  Kommentar
-------  ----          ---   -----------  ----------  ---------
    0.0  init_db1      WORD  W#16#63      W#16#63     Motor 1 vorhanden und initialisieren
    2.0  init_offset1  WORD  W#16#0       W#16#0      --> Liste ab DB 99 Datenwort 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 nicht vorhanden
   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 vorhanden
   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 vorhanden und initialisieren
   26.0  init_offset4  WORD  W#16#82      W#16#82     --> Liste ab DB 99 Datenwort 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 nicht vorhanden
   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 nicht vorhanden
   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 vorhanden
   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 vorhanden und initialisieren
   58.0  init_offset8  WORD  W#16#0       W#16#0      --> Liste ab DB 99 Datenwort 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 nicht vorhanden
   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
     .      .            .      .            .
     .      .            .      .            .
     .      .            .      .            .
                              (immer 0 => Motor xxx nicht vorhanden)


DB99: INIT_DB = Datenbaustein mit der Initialisierungsliste
==============================================================
Adresse    Name      Typ     Anfangswert   Aktualwert      Kommentar
-------    ----      ---     -----------   ----------       ---------
    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 => lesen (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 => schreiben (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: DATEN_DB = Datenbaustein mit den Ein- und Ausgangsdaten der maximal 127 Motoren

Adresse	Name	Typ	Anfangswert	Aktualwert	Kommentar
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.Wichtige CANopen-Messages

Die folgende Tabelle zeigt eine kurze Auflistung wichtiger allgemeiner CANopen-Nachrichten.

CAN-Identifier [HEX]	Bezeichnung	Länge	Daten [HEX]	Bemerkungen
0	NMT	2	01 xx	Start an alle (Preoperational -> Operational)
0	NMT	2	80 xx	Operational -> Preoperational
0	NMT	2	81 xx	Reset (z.B. CAN-I/O-Modul)
0	NMT	2	82 xx	Reset Communication
80h	SYNC	0	-	Sync an alle
80h + *Node-ID*	EMCY	0...8 Bytes	Fehlercode	Emergency Message (z.B. von CANopen-I/O-Modul)

Node-ID. ... Node-ID des angesprochenen CANopen-Moduls

10. Lizenzen

Dieses Produkt verwendet das Opensource FreeRTOS™ Betriebssystem.
Der vollständige Wortlaut der Lizenz ist im esd's "3rd Party Licensor Notice" Dokument als Bestandteil der Produktdokumentation auf der mitgelieferten CD enthalten.