



Benutzerhandbuch  
User Guide

# Programmierung mit Schnittstellenkarten

Programming with  
Interface Cards

	Seite
1. Allgemeines.....	4
1.1 Begriffserklärungen .....	4
1.2 Vorwort.....	4
1.3 Allgemeine Hinweise zur Kommunikation .....	4
1.4 Hinweise zum USB-Treiber.....	4
1.5 Aufbau der Kommunikation.....	4
1.6 Serielle Übertragungsparameter .....	4
1.7 Sollwerte und Istwerte umrechnen.....	5
1.8 Telegrammaufbau USB/RS232.....	5
1.9 Telegrammaufbau bei CAN .....	6
1.9.1 Bisheriges CAN-System mit Device node und RID .....	6
1.9.2 Das neue CAN-ID-System.....	6
1.9.3 Geteilte Telegramme .....	6
1.10 Telegrammaufbau bei IF-G1 .....	7
1.11 Telegrammaufbau IF-ExB (Ethernet) .....	7
1.12 Telegrammaufbau IF-ExB (USB).....	7
1.13 Timing von Telegrammen .....	7
2. Kommunikation in LabView .....	7
2.1 Übersicht Labview VIs.....	7
2.2 Installation.....	7
2.3 Kurzübersicht über die VIs .....	8
3. Kommunikation mit dem Gerät.....	8
3.1 Grundlegendes.....	8
3.1.1 Hinweis zur USB-Treiber-Bibliothek.....	8
3.2 Vorgehensweise.....	8
3.3 Telegramme bilden.....	9
3.3.1 Startdelimiter erzeugen .....	9
3.3.2 Die Maske .....	9
3.3.3 Beispiele.....	9
3.4 Das Zeitformat.....	10
3.4.1 Zeitformate für elektronische Lasten.....	10
3.4.2 Zeitformat für Netzgeräte .....	11
3.4.3 Zeitformat für Batterielader .....	11
3.5 Tipps.....	11
3.6 Hilfe bei Problemen.....	12
3.7 Fehlermeldungen .....	12
3.7.1 Erläuterungen zu einigen Fehlercodes .....	12
3.7.2 Fehlercodes Kommunikation .....	13
3.8 Objektlisten .....	13
3.8.1 Bedeutung der Spalten .....	13
3.8.2 Objektbeispiele- und erläuterungen .....	14
3.8.3 Über die Benutzerprofile .....	14
3.9 Gerätefehler, Alarmcodes und Alarmtypen.....	15
4. Profibus (IF-PB1).....	16
4.1 Allgemeines.....	16
4.2 Technische Daten.....	16
4.3 Unterstützte Geräteserien .....	16
4.4 Master-Slave-Kommunikation .....	16
4.4.1 Zyklisch .....	16
4.4.2 Azyklisch .....	16
4.5 Verkabelung / Terminierung .....	16
4.6 Übertragungsgeschwindigkeiten und Laufzeiten .....	17
4.6.1 Einfluß der Kabellänge.....	17
4.7 Betriebsmodi .....	17

4.8	Profibus-Slave-Adresse .....	18
4.9	Kommunikation mit dem Master.....	18
4.9.1	Zyklische Feldbusdaten .....	18
4.9.2	Azyklische Feldbusdaten .....	18
4.9.3	Anwendung der Datensätze.....	19
4.10	Projektierung am Beispiel einer Siemens Steuerung (SPS) .....	20
4.10.1	Einbindung des Netzgerätes an den Profibus.....	20
4.10.2	GSD-Datei-Installation .....	20
4.10.3	Plazierung der Module .....	21
4.10.4	Adressierung der zyklischen Module .....	21
4.10.5	Adressierung des azyklischen Moduls.....	21
4.11	Diagnose .....	22
4.11.1	Standarddiagnose.....	22
4.11.2	DP-V1 Alarmmanagement (erweiterte Diagnose).....	22

**⚠ Achtung!**

Für die textbasierende Kommunikation mit SCPI-Befehlen bei der GPIB-Karte IF-G1 oder den Netzwerkkarten IF-ExB Schnittstellenkarten gibt es gesonderte Handbücher.

# 1. Allgemeines

## 1.1 Begriffserklärungen

**Telegramm:** Kette von Bytes, mit unterschiedlicher Länge. Wird entweder zum Gerät gesendet oder vom Gerät empfangen.

**Singlecast:** Anfrage bzw. einfaches Senden an ein einzelnes Gerät. Bei in Reihe vernetzten Geräten, z.B. bei CAN, geht das Telegramm an alle Geräte, wird aber nur von dem adressierten Gerät akzeptiert. Betrifft den Startdelimiter (1. Byte einer Nachricht) bzw. die Verwendung von Geräteadressen.

**Broadcast:** Anfrage bzw. einfaches Senden an alle Geräte. Bei in Reihe vernetzten Geräten, z.B. bei CAN, geht das Telegramm an alle Geräte und wird auch von allen akzeptiert. Betrifft nur CAN. Kann benutzt werden, um ein Gerät das nicht per CAN verbunden ist, ohne Angabe des eigentlichen Device node anzusprechen. Der DN ist dann immer 0.

**Objekt:** beschreibt mit seinen Eigenschaften die Objektadresse und löst definierte Aktionen auf dem Zielgerät aus. Vergleichbar mit einem Befehl.

**Nachricht:** Datenpaket bei CAN, wie ein Telegramm, nur CAN-typisch ohne Startdelimiter und Checksumme.

**Botschaft:** wie Nachricht, bei Vector-Software verwendet

**Signal:** Teil einer Botschaft (Vector-Software)

## 1.2 Vorwort

Das hier erläuterte Kommunikationsprotokoll mit seiner objektorientierten Telegrammstruktur ist recht komplex. Es wird daher empfohlen, nach Möglichkeit die fertigen LabView-Bausteine zu benutzen, falls LabView als Programmieroberfläche genutzt wird. Die Anwendung des Protokolls in anderen Entwicklungsumgebungen, wie z.B. Visual Basic, C oder .NET, erfordert Programmierkenntnisse über die Einrichtung und Verwendung von Hardwareschnittstellen wie CAN oder USB und das Ansprechen der entsprechenden Treiber. Hier wird nur auf den Aufbau des Datenpakets (des Telegramms) eingegangen und nicht darauf, wie es richtig an das Gerät übertragen wird.

## 1.3 Allgemeine Hinweise zur Kommunikation

Die Firmware der verschiedenen Geräte, die mit den Schnittstellenkarten gesteuert werden sollen, ist so programmiert, daß sie die Gegebenheiten und Probleme, die sich bei der Ansteuerung von mehreren Geräten ergeben, so weit wie möglich beachtet. Daher ist es nicht möglich, zu jeder Zeit und bei jedem Zustand des Gerätes alle Objekte zu verwenden. So sind zum Beispiel die Daten für den Funktionsmanager der Serien PSI 9000 und PSI 8000 (siehe Benutzerhandbuch) nur im Standby des Gerätes transferierbar, ansonsten kommt eine Fehlermeldung zurück.

Diese enthält einen Fehlercode, der unter Anderem darauf hinweist, daß sich das Gerät möglicherweise nicht im Standby befindet.

## 1.4 Hinweise zum USB-Treiber

Für **Windows XP, Windows 2003, Windows Vista bzw. Windows 7** sind werden durch den Treiber zwei Geräte installiert. Ein USB-Gerät als „USB Serial Converter“ und ein virtueller COM-Port (VCP) als „USB Serial Port“.

Die VCP-Funktion ist standardmäßig aktiviert und erzeugt einen neuen COM-Port für jeden am PC angemeldeten USB-Port der Schnittstellenkarte IF-Ux, IF-ExB oder IF-PB1.

Die momentan für LabView verfügbaren VIs (Stand: 06/2011) können Geräte mit USB-Schnittstelle nur über den virtuellen COM-Port ansprechen. Alternativ sind noch der GPIB-Port der IF-G1-Karte, sowie der Ethernetport der IF-ExB-Karten mit den VIs nutzbar, je nach am PC vorhandenen Gegebenheiten.

Aktuellere Versionen des Treiber als die auf der beiliegenden CD befindliche sind auf der Webseite des USB-Chip-Hersteller FTDI unter [www.ftdichip.com](http://www.ftdichip.com) zu finden.

## 1.5 Aufbau der Kommunikation

Die Kommunikation mit den zu steuernden Geräten basiert auf diesen zwei Telegrammformen:

a) **Sendung:** es wird ein Objekt gesendet, das einen Wert, z.B. Spannung, setzen soll. Sofern dies im momentanen Betriebszustand des Gerätes zulässig ist, wird das Objekt akzeptiert und ausgeführt. Das Gerät sendet dann **keine** Antwort. Falls die Ausführung nicht zulässig sein sollte, kommt jedoch eine Fehlermeldung.

b) **Anfrage:** es wird mittels eines Objekts eine Anfrage an das Gerät gesendet, worauf man eine Antwort erwartet. Ist die Anfrage für den momentanen Betriebszustand des Gerätes zulässig, wird sie ausgeführt und die Antwort wird unverzüglich gesendet, die als Inhalt die angefragten Daten enthält. Falls nicht, wird als Antwort eine Fehlermeldung gesendet.

## 1.6 Serielle Übertragungsparameter

Betrifft: RS232 und USB-Ports bei Verwendung des VCP (siehe Abschnitt 1.4).

Bei der seriellen Übertragung eines Bytes werden folgende Bits übertragen:

Startbit + 8 Datenbits + Paritätsbit + Stoppbit

Das Parität wird auf „ungerade“ (engl.=odd) geprüft.

Der USB-Port (IF-Ux, IF-ExB, IF-PB1) arbeitet intern im Gerät mit der Übertragungscharakteristik der RS232-Karte (IF-Rx). Für diese Kartentypen sind zur Konfiguration am jeweiligen Windowstreiber folgende Parameter mindestens zu setzen, sowie am Gerät bei Verwendung einer RS232-Karte:

Baudrate RS232-Karte:	9600Bd ... 57600Bd
Baudrate USB-Port:	57600Bd
Parität:	ungerade
Stoppbits:	1

### 1.7 Sollwerte und Istwerte umrechnen

Die Sollwerte und Istwerte (siehe externe [Objektlisten](#)) werden, mit wenigen Ausnahmen, als Prozentwerte übertragen, wobei 0x6400 = 100,00% entspricht. Demzufolge müssen reale Sollwerte vor der Übertragung und Istwerte nach dem Empfang umgerechnet werden.

Wenn also ein Gerät eine Nennausgangsspannung von 80V hat, dann würde der von Gerät übertragene Spannungsistwert 0x3200 = 50,00% der Spannung 40V entsprechen.

Das Highbyte ist die Prozentzahl (0x64 = dezimal 100) und das Lowbyte die Nachkommastellen der Prozentzahl. Man muß die eingehenden Istwerte sowie die ausgehenden Sollwerte daher umrechnen.

$$\text{Realer Wert} = \frac{\text{Nennwert Gerät} \cdot \text{Prozentwert}}{25600}$$

Beispiel: Nennwert des Gerätes ist 80V, der prozentuale Spannungsistwert kam als 0x2454 = 9300. Nach der Formel ergibt sich Istwert =  $(80 \cdot 9300) / 25600 = 29,06V$ .

$$\text{Zu sendender Wert} = \frac{25600 \cdot \text{Realer Wert}}{\text{Nennwert Gerät}}$$

Beispiel: die Leistung soll 500W sein, der Nennwert d. Gerätes ist 640W. Nach der Formel ergibt sich:  
 Prozent-Sollwert =  $(25600 \cdot 500) / 640 = 20000 = 0x4E20$ .

### 1.8 Telegrammaufbau USB/RS232

Die Schnittstellenkarten IF-Rx (RS232) und die USB-Ports der Karten IF-Ux, IF-ExB und IF-PB1 arbeiten mit einer leicht anderen Telegrammstruktur als die CAN-Karten IF-Cx. Lesen Sie im Abschnitt 1.9 weiter, wenn Sie eine CAN-Karte benutzen.

Das Telegramm hat den folgenden Aufbau

#### SD + DN + OBJ + DATEN + CS

und setzt sich aus diesen Bytes zusammen:

#### Byte 0: SD (start delimiter)

Der Startdelimiter zeigt den Beginn eines Telegramms an, die Länge der Daten, den Absender und den Telegrammtyp.

#### Bits 3-0: Datenlänge (Bytes 3-18)

Geben die Datenlänge - 1 der Daten im Telegramm an.

Bei einer Anfrage steht hier die Länge -1 der zurückerwarteten Daten. Beispiel: die [Objektliste](#) gibt 6 für die Länge des Objekts an, also wird das untere Nibble des SD mit 5 angegeben.

*Ausnahme: Objekte vom Typ „string“, hier ist der Wert egal, solange er die in der Objektliste gegebene Länge nicht überschreitet, kann also 0...Objektlänge sein.*

#### Bit 4: Richtung

- 0 = Nachricht vom Gerät an den PC
- 1 = Nachricht vom PC an das Gerät

#### Bit 5:

- 0 = Singlecast, Nachricht an einen bestimmten Empfänger
- 1 = Broadcast, eine Rundnachricht an mehr als einen Teilnehmer (bei CAN), ansonsten wie Singlecast mit DN=0)

#### Bits 7+6: Sendungstyp

- 00= Reserviert
- 01= Anfrage von Daten
- 10= Antwort auf eine Anfrage
- 11= Daten senden

#### Byte 1: DN (device node)

Über den [device node](#) (=Adresse), wird das Gerät in Bussystemen wie CAN oder GPIB adressiert. Ein Geräteknoten darf innerhalb eines Bussystems nur einmalig vergeben werden. Wertebereich: 1...30, andere sind nicht gültig. Bei CAN berechnen sich aus dem Geräteknoten die CAN-IDs (je nach Geräteserie und Firmware), mehr dazu in Abschnitt 1.9. Für Punkt-zu-Punkt-Verbindungen wie RS232 oder USB ist diese Adresse nur wichtig, wenn man den Sendungstyp Singlecast (siehe Byte 0) benutzt. Beim Sendungstyp Broadcast kann der DN immer 0 sein.

#### Byte 2: OBJ (object)

Die Kommunikationsobjekte eines Gerätes werden über den hier angegebenen Wert adressiert. In den [Objektlisten](#) werden die weitere Funktion(en) oder Eigenschaften der Objekte beschrieben.

#### Byte 3 - 18: DATEN

Der Datenbereich kann bis zu 16 Bytes lang sein, die Länge des Telegramms variiert also. Bei einer Anfrage (PC → Gerät) werden keine Daten übermittelt. Der Datenbereich entfällt dann und ab Byte 3 folgt direkt die Checksumme (siehe unten). Nur bei einer Antwort (Netzgerät → PC), erwartet oder Fehler, werden Daten übermittelt.

#### Letzte 2 Bytes: CS (checksum)

Die Position der Prüfsumme (checksum) ist stets am Ende des Telegramms. Die Prüfsumme wird über die einfache Addition aller vorherigen Bytes des Telegramms gebildet und hinten angehängt. Sie ist zwei Bytes lang (16Bit-Wert). Das Highbyte wird vor dem Lowbyte gesendet.

#### Beispiel für ein Telegramm:

An ein Gerät mit Geräteadresse 1 soll das Objekt 71 gesendet werden (Istwerte anfragen). Das Telegramm könnte so aussehen (Hexwerte):

**55 01 47 00 9D**

Die Antwort wiederum könnte dann so aussehen:

**85 01 47 64 00 1E 00 50 00 01 9F**

(das ergibt 80V, 30A und 2400W bei einem Netzgerät mit 80V, 100A und 3000W Nennwerten, wie z.B. PSI 9080-100)

Siehe auch Abschnitt 1.7 für die Umrechnung der Werte. Weitere Beispiele siehe Abschnitte 3.4 und 3.8.2.

## 1.9 Telegrammaufbau bei CAN

### **Achtung!**

Die Geräteserien **PS 8000 (ab Firmware 6.01)**, **PSI 8000 (ab Firmware 3.14)** und **EL 3000/EL 9000 (ab Firmware 5.01)** haben ein umschaltbares CAN-ID-System. Andere Geräteserien unterstützen nur das System wie in 1.9.1 beschrieben.

Die CAN-Schnittstellenkarten IF-C1 und IF-C2 unterstützen den CAN-Standard 2.0a. Das erweiterte Adreßformat wird nicht verwendet.

Lesen Sie auch bitte in den Handbüchern zu Ihrem Gerät und den Schnittstellenkarten bezüglich der Kommunikationseinstellungen zu CAN nach.

### 1.9.1 Bisheriges CAN-System mit Device node und RID

Der CAN-Treiberbaustein benötigt für eine Nachricht den **Identifizier**, bis zu **8 Datenbytes** und die **Datenlänge**. Der Identifizier ist 11 Bit (CAN 2.0a) lang und wird durch den **device node**, das verschiebbare Adreßsegment **RID (Relocatable Identifier)** und den Typ der Nachricht gebildet. Für jedes Gerät sind zwei Identifizier vorgegeben:

[RID\*64 + device node \* 2] (1. Identifizier) und

[RID\*64 + device node \* 2 + 1] (2. Identifizier),

wobei der 1. Identifizier nur für Objekte benutzt wird, die Daten an das Gerät senden (Typ: Sendung) und der 2. für Objekte, die Daten anfragen (Typ: Anfrage) und auch für alle Antworten vom Gerät.

Pro CAN-Nachricht (Message) können maximal 8 Bytes übertragen werden. Das erste Byte wird belegt durch die Adresse des Kommunikationsobjekts. Danach können bis zu 7 Datenbytes folgen (siehe [Objektlisten](#)). Um ein Objekt mit einem 16 Byte großen Datenbereich zu schicken wären demnach drei Nachrichten nötig. Siehe auch Abschnitt 1.9.3.

Zwei Beispiele:

a) Das Gerät soll in den Fernsteuerbetrieb (remote) gesetzt werden. Dieser ist erforderlich, um das Gerät zu steuern und Sollwerte zu senden. Der **device node** wurde am Gerät auf 15 und der **RID** auf 3 gesetzt. Da nur gesendet wird, ist der Nachrichtentyp Sendung. Es ergibt sich ein Sende-Identifizier von  $3 * 64 + 15 * 2 = 222_{10}$  oder  $0xDE$ , laut obenstehender Formel. Nach der [Objektliste](#) wird das Objekt 54 (hex:  $0x36$ ) mit den Datenbytes  $0x10$  (Maske) und  $0x10$  (set remote) benötigt. Die sich ergebende Datenlänge ist 3. Somit sehen die Daten, die an den CAN-Controller übergeben werden, so aus (Hexwerte):

**ID DL DATEN**  
**DE 03 36 10 10**

b) Wollte man den Zustand des Gerätes nicht setzen, sondern abfragen, so würde nun der Identifizier  $0xDF$  (2. Identifizier) verwendet und zwecks einer Anfrage reicht die Objektzahl allein als Datum aus. Die sich ergebende CAN-Nachricht sieht dann so aus:

**DF 01 36**

und die Antwort sollte so aussehen:

**DF 01 36 10 10**

## 1.9.2 Das neue CAN-ID-System

### **Hinweis**

*Hinweis: das neue CAN-ID-System kann über ein Firmwareupdate für die Serien PS 8000, PSI 8000, EL 3000 und EL 9000 verfügbar gemacht werden.*

Das neue CAN-ID-System nutzt 3 normale CAN-IDs pro Gerät, plus 1 Broadcast-ID. Dadurch ist es zu Vector-Software wie z. B. CANalyzer oder CANoe kompatibel. Vector-Datenbasen vom Typ \*.dbc sind auf Anfrage erhältlich bzw. bereits auf der beiliegenden CD zu finden. Es gibt auch fertige Konfigurationen, die zu Demonstrations- und Testzwecken in diese Softwares geladen werden können.

Im Geräte-Setup wird, sofern eine CAN-Schnittstelle IF-C1 oder IF-C2 installiert ist, vom Anwender eine „Base ID“ eingestellt. Durch diese erhält ein Gerät drei CAN-IDs (Base ID, Base ID + 1 und Base ID + 2). Diese IDs müssen in einem Bus eindeutig sein und sind deshalb nur in 4er-Schritten einstellbar. Falls mehrere gleichartige Geräte an einem Bus angeschlossen sind, müssen alle eindeutige Base IDs haben.

Bezüglich der „Broadcast ID“ (auch: Broad ID) ist es andersherum. Diese sollte bzw. muß bei beteiligten Geräten gleich eingestellt sein, um eine Nachricht gleichzeitig an mehrere Einheiten zu schicken. Sinn dieser Funktion ist es, einen Sollwert oder Zustand bei den adressierten Geräten möglichst zeitgleich zu setzen. Die Broadcast-ID kann in 1er-Schritten eingestellt werden und muß sich von den anderen CAN-IDs des Gerätes unterscheiden.

Die drei „normalen“ CAN-IDs sind so zugeordnet:

Base ID = Wird vom PC nur zum Senden von Nachrichten benutzt, die etwas setzen (Sollwert, Status) und wo keine Antwort erwartet wird

Base ID + 1 = Wird vom PC zur Anfrage von Werten oder Status benutzt, wo eine Antwort vom Gerät erwartet wird. Es wird nur das angefragte Objekt gesendet (siehe Dokumentation über die verfügbaren Objekte (=Befehle)

Base ID + 2 = Wird vom Gerät verwendet um Antworten zu einer Anfrage oder Kommunikationsfehlermeldungen zu senden

Die Broadcast-ID ist so zugeordnet:

Broad ID = Wird vom PC nur zum Senden von Nachrichten an mehrere Geräte gleichzeitig benutzt, die etwas setzen

### 1.9.3 Geteilte Telegramme

Bei einem geteilten Telegramm, d.h. einem Telegramm, das sich aus mehreren Teilnachrichten zusammensetzt (nur zutreffend bei Objekten im „string“-Format), wird nach der Objektadresse eine weitere Kennung eingefügt. Die Kennung der ersten Nachricht ist  $0xFF$ , der zweiten Nachricht ist  $0xFE$  und die dritte Nachricht ist  $0xFD$ . Diese Kennung hilft dabei, diese Telegramme als aufgeteilt zu identifizieren und deren Dateninhalt nach Empfang wieder richtig zusammen zu setzen. Die Reihenfolge der Nachrichten ist nicht fest vorgegeben. Bei Verwendung der Gateway-Funktion (nur PSI9000) werden die geteilten Telegramme nicht vom Gateway zusammengesetzt. Dies muß in der übergeordneten Steuereinheit geschehen.

### 1.10 Telegrammaufbau bei IF-G1

Der Telegrammaufbau für die textbasierte Kommunikation über eine IEEE/GPIB-Karte ist in externen Handbüchern beschrieben:

Link: [SCPI-Befehlsliste für Netzgeräte](#)

Link: [SCPI-Befehlsliste für elektronische Lasten](#)

### 1.11 Telegrammaufbau IF-ExB (Ethernet)

Der Telegrammaufbau für die textbasierte Kommunikation über den Ethernet-Port einer IF-ExB-Karte ist in externen Handbüchern beschrieben:

Link: [SCPI-Befehlsliste für Netzgeräte](#)

Link: [SCPI-Befehlsliste für elektronische Lasten](#)

### 1.12 Telegrammaufbau IF-ExB (USB)

Siehe Abschnitte 1.6 und 1.8.

### 1.13 Timing von Telegrammen

Nach jeder Anfrage benötigt das Gerät typisch 5ms und maximal 50ms für eine Antwort. Grundsätzlich darf unmittelbar nach der Antwort wieder gesendet werden. Empfohlen wird eine Zeit von 100 ms zwischen zwei Befehlen, damit das Gerät nicht zu sehr durch die Kommunikation ausgebremst wird.

Befehle, die etwas setzen, wie Sollwerte, werden auch mit einer variablen Zeit (5-50ms) umgesetzt. Dazu kommt noch die Reaktionszeit des Leistungsteils und dessen Reglers.

#### Gateway (nur alte PSI 9000 Serie bis 2012)

Bei der Gateway-Funktion muß zudem die Übermittlung der Telegramme von einem Bussystem auf das andere Bussystem berücksichtigt werden. Hier kann sich die Antwort bis zu 200 ms verzögern.

Nach dem Empfangen einer Fehlermeldung sollte mindestens 100ms gewartet werden.

#### Broadcast bei CAN

Nach jeder Rundumanfrage können die Busteilnehmer nur nacheinander antworten. Abhängig vom Bussystem, der Baudrate und der Anzahl der angesprochenen Busteilnehmer, sowie dem zusätzlichen anderen Datenverkehr wird sich die Antwort mehr oder weniger verzögern. Da die Zeit nur individuell zu spezifizieren ist, kann sie in erster Annäherung mit *Busteilnehmeranzahl \* normale Antwortzeit* angenommen werden. In den meisten Fällen wird die Antwortzeit aber wesentlich kürzer sein.

## 2. Kommunikation in LabView

### 2.1 Übersicht Labview VIs

Zur Integration der Geräte in eigene Labview-Applikationen werden Labview VIs zur Verfügung gestellt.

Mit den virtuellen Instrumenten (VI) ist eine einfache Einbindung und Programmierung einer Anwendung möglich, ohne daß der Anwender sich in die unteren Ebenen der Kommunikation einarbeiten muß. Sie erleichtern das Einfügen in bestehende Anwendungen oder die Erstellung eines anwenderspezifischen Programms.

Um die Funktionen der VIs nutzen zu können, wird die Softwareentwicklungsumgebung Labview der Firma National Instruments benötigt. Die bereitgestellten Labview VIs benötigen LabView-Version 7.0 oder höher.

Folgende minimale Systemvoraussetzungen sollten erfüllt sein:

- Pentium 3 Prozessor mit 256 MB Hauptspeicher
- Windows Betriebssystem (Win98 oder WinXP)

Aktuelle Versionen können über unsere Webseite heruntergeladen werden.

### 2.2 Installation

Um die Labview VIs in Ihre Umgebung einzubinden, lesen Sie bitte die Installationshinweise in der Datei „installation\_deutsch.pdf“ auf der der Schnittstellenkarte beiliegenden CD.

Nach der Installation finden Sie die VIs in LabView normalerweise im Kontextmenü unter „Instrumenten-I/O -> Instrumententreiber -> IF-XX“.

Es gibt VIs, die nur für Geräte einer bestimmten Serie gedacht sind und auch nur mit diesen funktionieren. Diese befinden sich in Unterordnern des VI-Satzes, die nach der Serie benannt sind. Andere VIs sind gemeinsam nutzbar. Funktion und Benutzung sind im Handbuch zu den VIs beschrieben. Dieses rufen Sie wie gewohnt über die LabView-Hilfe (Strg+H) auf oder öffnen die CHM-Datei direkt. Je nach Windowsversion kann es aufgrund von Sicherheitseinstellungen des Internetexplorers nötig sein, die Datei vor dem Öffnen auf die Festplatte zu kopieren, damit Sie den Inhalt sehen können.



#### Hinweis

Lesen Sie die LabView VIs Hilfedatei auf der beiliegenden CD, um einen Überblick über Funktion und Handhabung der VIs zu bekommen.

### 2.3 Kurzübersicht über die VIs

Stand: 03/2015

Nach dem Entpacken der VIs sind mehrere Unterordner verfügbar.

Die allgemeinen VIs im Ordner „\Common“:

- **Device\_close.vi** - schließt die Kommunikation mit dem gewählten Gerät. Vor Ende des Programms muß bzw. bei nicht weiterer Nutzung sollte der Kommunikationsport des Gerätes geschlossen werden.
- **Device\_scan.vi** - Sucht vorhandene Hardwareschnittstellen (RS232, GPIB, Ethernet, USB(VCP)) nach kompatiblen Geräten ab und gibt die Anzahl sowie Geräteinformationen heraus, die weiterverwendet werden können.
- **Device\_select.vi** - dient zur Auswahl und zum Öffnen eines Gerätes aus der Liste der Geräte, die mit Device\_scan.vi ermittelt wurde. Es können parallel mehrere Geräte ausgewählt und geöffnet werden.

Die serienspezifischen VIs in den Ordnern:

- „\BCI8 Series“ - für Geräte der Serie BCI 800 R
- „\PS2 Series“ - für Geräte der Serie PS 2000 B
- „\PS5 Series“ - für Geräte der Serie PS 5000 A
- „\PS8 Series“ - für Geräte der Serien PS 8000
- „\PSI5 Series“ - für Geräte der Serie PSI 5000 A
- „\PSI8 Series“ - für Geräte der Serien PSI 8000 und PSI 800 R
- „\PSI9 Series“ - für Geräte der Serien PSI 9000 (alt und neu)
- „\EL Series“ - für Geräte der Serien EL 3000 und EL 9000
- „\ELR Series“ - für Geräte der Serie ELR 9000

Die Beispiele im Ordner „\Examples“ sollen den Standardgebrauch der allgemeinen und spezifischen VIs verdeutlichen. Dazu können die serienspezifischen VIs für das jeweils zu testende Gerät ersetzt werden.

 **Achtung!**

**Wann immer ein serienspezifisches VIs, z. B. „el9000.vi“ durch ein anderes serienspezifisches ersetzt wird, muß die Enumerationskonstante am Eingang „command list“ gelöscht und neu erzeugt werden!**

### 3. Kommunikation mit dem Gerät

#### 3.1 Grundlegendes

Im Folgenden wird auf den Aufbau der Telegramme, die Abhängigkeiten der Kommunikation vom Zustand des zu steuernden Gerätes und Probleme mit der Kommunikation eingegangen, ohne detailliert zu erläutern, wie z. B. bei USB der USB-Treiber auf „low level“ anzusprechen ist bzw. wie eine CAN-Nachricht richtig verschickt wird. Dies ist vom Anwender und abhängig von der Einsatzsituation unserer Geräte selbst in Erfahrung zu bringen.

##### 3.1.1 Hinweis zur USB-Treiber-Bibliothek

Bei Verwendung einer USB-Karte IF-Ux oder des USB-Ports der Ethernetkarte IF-E1B bzw. der Profibuskarte IF-PB1 ist ein Treiber zu installieren. Die Kommunikation kann danach wahlweise über den USB-Treiber (low-level access) oder über einen virtuellen COM-Port (VCP) erfolgen, wobei letzter wesentlich einfacher zu handhaben ist

Auf der beiliegenden CD befindet sich im Ordner \manuals\other\ftdi ein PDF, daß die Funktionen zum Ansprechen des USB-Treibers beschreibt. Generell gilt, daß ein Gerät (in dem Fall die USB-Hardware) zuerst zu öffnen ist (FT\_Open o.ä.), dann zu konfigurieren (FT\_SetBaudRate, FT\_SetDataCharacteristics und eventuell noch andere) ist und danach geschrieben (FT\_Write) oder gelesen werden (FT\_GetQueueStatus, FT\_Read) kann. Wird das Gerät nicht mehr benutzt, sollte es geschlossen werden (FT\_Close), wobei davon abgeraten wird, Öffnen und Schließen für jeden Schreib-Lese-Zyklus zu machen. Die Funktionen FT\_Write und FT\_Read dienen zum Transport der Telegrammbytes der eigentlichen Kommunikation, die in den nächsten Abschnitten beschrieben wird.

#### 3.2 Vorgehensweise

Das Programmieren der unterschiedlichen Geräte, in denen die Schnittstellenkarten verwendet werden, erfolgt stets nach dem gleichen Schema, wenngleich sich Anzahl und Funktion der Kommunikationsobjekte, die durch eine bestimmte Geräteserie unterstützt werden, unterscheiden.

Generell gilt:

- Überwachung (Monitoring), also nur Abfrage von Istwerten und Status, ist immer möglich. Die Geräte benötigen dazu keinen Fernsteuerbetrieb.
- Setzen von Zuständen und Sollwerten erfordert die Aktivierung des Fernsteuerbetriebes (remote = ferngesteuert durch eine digitale Schnittstelle)
- Der Fernsteuerbetrieb kann durch bestimmte Umstände blockiert sein, z. B. expliziter Lokalbetrieb oder wenn sich ein Gerät in einer Betriebsart befindet, die keinen Fernsteuerbetrieb zuläßt und dann eine derartige Anfrage ignoriert. Näheres dazu entnehmen Sie bitte der Bedienungsanleitung Ihres Gerätes.

Um ein Gerät zu **steuern**, sprich z. B. einen Sollwert zu senden und zu setzen, müssen Sie mindestens

**1. den Fernsteuerbetrieb aktivieren (Objekt 54)**

**2. den Sollwert senden**

und, wenn nicht bereits geschehen,

**3. den Eingang/Ausgang einschalten.**

Der Fernsteuerbetrieb sollte verlassen werden, wenn er nicht mehr benötigt wird. Solange er aber aktiviert ist, kann das Gerät nicht oder nur bedingt manuell bedient werden. Der Modus wird stets auf der Front des Gerätes angezeigt.

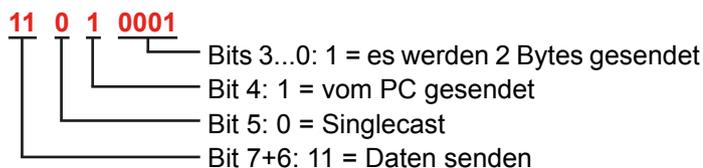
**3.3 Telegramme bilden**

**3.3.1 Startdelimiter erzeugen**

**! Hinweis**

*Der Startdelimiter wird bei CAN nicht benötigt*

Nach der Vorgabe des Telegrammformats (siehe Abschnitt 1.8) ist das erste Byte der Startdelimiter (SD), der von der Richtung des Telegramms und dem Anfragetyp abhängig ist. Nimmt man zum Beispiel einen SD von 0xD1, dann sieht der in Bits zerlegt so aus:



Alternativ zum bitweisen Zusammensetzen kann man sich das vereinfachen, indem man Hexwerte addiert. Ausgehend von Bit 7+6 ergibt sich folgendes:

**SD = Sendungstyp + Castform + Richtung + Datenlänge**

wobei Sendungstyp entweder

- 0xC0** Daten senden oder
- 0x40** Anfrage

und Castform entweder

- 0x00** Singlecast (Device node wird verglichen) oder
- 0x20** Broadcast (Device node wird ignoriert)

und Richtung entweder

- 0x10** vom PC ans Gerät oder
- 0x00** vom Gerät an den PC

und die Datenlänge - 1 von

- 0x00...0x0F** bis zu 16 Bytes am Stück

**! Hinweis**

*Die Datenlänge entspricht der Anzahl der gesendeten Bytes -1!! Dies ist stets beachten!!*

**3.3.2 Die Maske**

Einige Objekte erfordern eine Maske, die immer vor dem eigentlichen Steuerbyte gesendet wird. Diese Maske ist in Spalte 7 der [Objektlisten](#) angegeben. Im Steuerbyte haben die einzelnen Bits unterschiedliche Funktionen, daher muß bestimmt werden, welches Bit geändert werden soll. Dies übernimmt die Maske mit einer 1 für das jeweilige Bit. Beispiel: man möchte Bit 4 auf 1 verändern, dann wäre das Steuerbyte 0x10 und die Maske 0x10, entsprechend der Wertigkeit der Bits. Ist in der Objektliste nur eine einzelne Maske vorgegeben, wie z. B. 0x0F, so werden immer alle betroffenen Bits geändert, weil das Objekt das so erfordert (z. B. Objekt 56, Funktionsmanager bei PSI 8000 und PSI 9000). Hier darf im unteren Nibble des Bytes immer nur ein Bit gesetzt sein. Da wäre die Maske dann immer 0x0F und das Steuerbyte könnte z. B. 0x02 sein (Befehl: STEP).

**! Achtung!**

**Um Kollisionen der Bitfunktionen zu vermeiden, sollte stets nur ein Bit pro Telegramm verändert werden, auch wenn mehrere gleichzeitig möglich wären.**

**3.3.3 Beispiele**

**Beispiel 1: Gerät in Fernsteuerbetrieb setzen**

Die Geräteadresse des anzusprechenden Gerätes ist beispielsweise 5, das zu benutzende Objekt 54 (als Hexwert 0x36), die Maske für den Fernsteuerbetrieb (siehe auch [Objektlisten](#)) ist 0x10 und das Steuerbyte für Fernsteuerung 0x10 (remote = ein). Somit ergibt sich dieses Hexbytes-Telegramm (Präfixe weggelassen für bessere Übersicht):

**D1 05 36 10 10 01 2C**

Zum Umkehren des Ganzen, also der Deaktivierung, ist dann

**D1 05 36 10 00 01 1C**

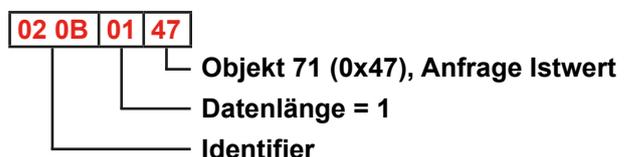
zu senden. Die Maske bleibt gleich, nur das Steuerbyte ändert sich.

**Beispiel 2: Istwerte abfragen über CAN**

Bei CAN entfallen der Startdelimiter und die vom Anwender berechnete extra Checksumme. Somit benötigen wir nur das Objekt (laut [Objektliste](#) 71 (hex: 0x47) für die Istwerte), den Identifier und die Länge der Nachricht. Bei der CAN-Nachricht zählt die Objektzahl zur Datenlänge, daher ergibt sich hier eine Datenlänge von 1, weil hier nur die Objektzahl zur Anfrage der Istwerte gesendet wird.

Nach dem alten CAN-ID-System (siehe Abschnitte 1.9.1 und 1.9.2) sei die Geräteadresse (device node) auch hier 5, der RID sei mal auf 8 gesetzt. Gemäß der Formel aus Abschnitt 1.9.1 ergibt sich ein Identifier von  $8 * 64 + 5 * 2 + 1 = 523$  (hex = 0x20B). Die +1 deshalb, weil es eine Anfrage ist. Wir schicken also an CAN ID 0x20B ein Byte.

Die Daten für die CAN-Übertragung könnten dann so aussehen (je nach Darstellung in der jeweiligen Softwareoberfläche):



**⚠ Achtung!**

Das Beispiel oben zeigt nicht den Dateninhalt der Nachricht, der tatsächlich über den CAN-Bus übertragen wird. Der Identifier und die Datenlänge sind nicht Teil des Dateninhalts.

Eine mögliche Antwort auf diese Anfrage könnte so aussehen:

**02 0B 06 64 00 0A 00 42 AA**

Gleicher Identifier, Datenlänge ist 6, weil immer drei 16-Bit-Istwerte übertragen werden. Die Istwerte werden als Prozentzahlen übertragen und müssen entsprechend des Typs des Gerätes zurückgerechnet werden. Siehe dazu auch Abschnitt „1.7 Sollwerte und Istwerte umrechnen“. Für eine EL 9080-200 ergäben sich hier 100% für Spannung (=80V), 10% für Strom (=20A) und 66,7% für die Leistung (=1600W).

**! Hinweis**

Gerätenennwerte wie Nennstrom, Nennleistung und Nennspannung, können mit entsprechenden Objekten aus dem Gerät gelesen und zur Umrechnung benutzt werden.

**3.4 Das Zeitformat**

Das für alle unsere Geräteserien verwendete Zeitformat kann mit einem 16bit-Wert Zeitangaben zwischen 1µs und 100h übertragen. Zeitwerte werden vom Gerät, an das sie übertragen werden, auf Plausibilität geprüft. Zu hohe bzw. zu niedrige Werte werden nicht akzeptiert und es wird mit einer Fehlermeldung geantwortet. Die oberen 4 Bits des 16-Bit-Wertes werden als Maske für den Zeitbereich genutzt, die restlichen Bits für den Zeitwert selbst. Das Zeitformat wird für das Lesen und Setzen von Werten gleichermaßen benutzt.

Das Zeitformat ist gültig für alle Geräte, bei denen irgendeine Funktion Zeitwerte benutzt. Die Auflösung der Zeitbereiche in der Tabelle unten deckt sich nicht immer mit der Auflösung der Zeitwerte am Gerät. Für diesen Fall werden an das Gerät gesendete Zeitwerte gerundet. Ein Beispiel: es wird der Zeitwert 0x23E7 an eine elektronische Last geschickt. Das sind laut Tabelle 999 x 1µs = 999µs. Die manuell am Gerät einstellbare Zeit in diesem Zeitbereich ist aber 0,95ms oder 1ms. Die 999µs werden auf 950µs abgerundet.

Deswegen wird auch, wenn der Zeitwert zurückgelesen wird, nicht 0x23E7 sondern 0x23B6 (=950) geantwortet.

**! Hinweis**

Es werden nicht von jeder Geräteserie alle Zeitschlüssel verwendet.

**! Hinweis**

Nicht alle Zeitwerte, die auslesbar sind, sind nach dem Zeitformat aufgebaut. Siehe [Objektlisten](#).

**3.4.1 Zeitformate für elektronische Lasten**

Für die elektronischen Lasten und die **Anstiegszeit (Objekt 92)** gilt, gemäß der Haupttabelle:

Zeitbereich		Schrittweite Gerät	Zeit- schlüssel	Wertebereich*	
von	bis			von	bis
30µs	99µs	1µs	0x2000	0x201E	0x2063
0.10ms	0.99ms	50µs	0x2000	0x2064	0x23DE
1.0ms	9.9ms	100µs	0x3000	0x3064	0x33DE
10ms	99ms	1ms	0x6000	0x6064	0x63DE
100ms	200ms	1ms	0x7000	0x7064	0x70C8

\* Werte, die von der Schrittweite abweichen, werden bei Empfang vom Gerät abgerundet

Für die elektronischen Lasten und die **Pulsbreite (Objekte 90 und 91)** gilt, gemäß der Tabelle oben:

Zeitbereich		Schrittweite Gerät	Zeit- schlüssel	Wertebereich*	
von	bis			von	bis
0,05ms	0.95ms	50µs	0x2000	0x2032	0x23B6
1.00ms	9.95ms	50µs	0x3000	0x3064	0x33E3
10ms	99.9ms	100µs	0x6000	0x6064	0x63E7
100ms	999ms	1ms	0x7000	0x7064	0x73E7
1.00s	9.99s	10ms	0x4000	0x4064	0x43E7
10.0s	100s	100ms	0x9000	0x9064	0x93E8

\* Werte, die von der Schrittweite abweichen, werden bei Empfang vom Gerät abgerundet

Für die elektronischen Lasten und die **Batterietestlaufzeit (Objekt 64)** gilt, gemäß der Tabelle oben:

Zeitbereich		Schrittweite Gerät	Zeit- schlüssel	Wertebereich*	
von	bis			von	bis
1s	3599s	1s	0x8000	0x8001	0x8E0F
1h	99h:59m	1m	0xC000	0xC03C	0xD76F

\* Werte, die von der Schrittweite abweichen, werden bei Empfang vom Gerät abgerundet

Zeitschlüssel *	Zeitwert (Bits 11..0)				Auflösung	resultierende Zeitbereich
	min.(dez)	min.(hex)	max.(dez)	max.(hex)		
0x2000 <sup>1</sup>	0	0x00	999	0x3E7	1µs	0 ... 0,999ms
0x3000 <sup>2</sup>	100	0x64	999	0x3E7	10µs	1ms ... 9,99ms
0x6000 <sup>1</sup>	100	0x64	999	0x3E7	100µs	10ms ... 99,9ms
0x7000 <sup>2</sup>	100	0x64	999	0x3E7	1ms	100ms ... 999ms
0x0000 <sup>1</sup>	0	0x00	4999	0x1387	2ms	0 ... 9,998s
0x4000 <sup>1</sup>	100	0x64	5999	0x176F	10ms	1,00s ... 59,99s
0x8000 <sup>1</sup>	1	0x01	3599	0xE0F	1s	1s ... 59min:59s
0x9000 <sup>2</sup>	100	0x64	1000	0x3E8	100ms	10,0s ... 100,0s
0xC000 <sup>1</sup>	0	0x00	5999	0x176F	1m	00:00h ... 99:59h

Tabelle: Alle Zeitformat für alle Geräteserien

\* Wenn der Schlüssel ausmaskiert werden soll, um empfangene Zeitwerte in reale Zeitwerte umzurechnen, sind die Bits 15..13 bzw. Bits 15..12 relevant, je nach Zeitbereich

**Beispiel 1:** Sie möchten bei einer elektronischen Last die Anstiegszeit auf 75ms setzen. Bei 75ms ist die Auflösung an der Last 1ms. Es ist also der Zeitbereich mit Schlüssel 0x6000 zu verwenden. Dessen Auflösung ist 0,1ms, daher ergibt sich ein Wert von 750 (75ms : 0,1ms), das entspricht 0x2EE. Es müßte dann also 0x62EE als Zeitwert für die Anstiegszeit (Objekt 92) gesendet werden.

 <b>Hinweis</b>
<i>LabView-Nutzer müssen die Zeit anders vorgeben, siehe VI-Beschreibung</i>

**Beispiel 2:** Der Zeitwert des Batterietests wurde ausgelesen und soll umgerechnet werden. Beim Batterietest ist die Auflösung 1s. Da die Zeitbereiche die Auflösung von 1s nur bis zu 1 Stunde zulassen und darüber hinaus in Minuten aufgelöst wird, würde der Zeitwert 0x8743 einer Zeit von 1859s oder 30m59s entsprechen, der Zeitwert 0xC532 dann 1330m oder 22h10m. Die Sekunden werden im Zeitbereich 0xC000 nicht angegeben, man würde also eine Minute lang immer den gleichen Zeitwert geliefert bekommen.

**Beispiel 3:** Setzen der Pulsbreite für A (Objekt 90) auf 5s. Laut der 2ten obigen Tabelle ergibt sich der Zeitschlüssel 0x4000. Mit der für diesen Zeitbereich geltenden Auflösung von 10ms ergibt sich der Wert 500 (5s : 0,01s), in hexadezimal 0x1F4. Der resultierende Zeitwert wäre daher 0x41F4.

**3.4.2 Zeitformat für Netzgeräte**

Gilt für: PSI 8000 / PSI 9000

Für Zeitangaben in den Sequenzen des Funktionsmanagers sind Zeitformate mit 2ms-Raster oder einem Vielfachen davon zu verwenden. Für andere Zeitwerte (z. B. Objekte 39-43 und 47) werden die gleichen Zeitschlüssel verwendet:

Zeitbereich		Schrittweite Gerät	Zeit- schlüssel	Wertebereich	
von	bis			von	bis
0.002s	9.998s	2ms	0x0000	0x0001	0x1387
10.00s	59.99s	10ms	0x4000	0x43E8	0x576F
1:00m	59:59m	1s	0x8000	0x803C	0x8E0F
1:00h	99:59h	1m	0xC000	0xC03C	0xD733

**3.4.3 Zeitformat für Batterielader**

Gilt für: BCI 800 R

Batterieladegeräte der Serie BCI 800 R nutzen für die abgelaufene Ladezeit keins der Zeitformate, sondern zählen in Sekunden, Minuten, Stunden und Tagen in einzelnen Bytes. Siehe BCI 800 R Objektliste.

Für andere Parameter, wie die max. Zeit einer Ladephase, wird der Zeitschlüssel 0xC000 verwendet:

Zeitbereich		Schrittweite Gerät	Zeit- schlüssel	Wertebereich	
von	bis			von	bis
0:00h	99:59h	1m	0xC000	0xC000	0xD76F

**3.5 Tipps**

I. Geräteadresse eines Gerätes ermitteln (nicht bei GPIB)

Wenn Sie z. B. ein Gerät über USB steuern möchten und dessen Geräteadresse (Device node) nicht kennen, können Sie zuerst eine Anfrage mit der Broadcastadresse 0 stellen und z. B. die Geräteklasse abfragen. Das Gerät wird in der Antwort dann die am Gerät eingestellte Adresse (device node) mitsenden, die dann für zukünftige Befehle verwendet werden kann. Alternativ kann bei USB- oder RS232-Ports das Ansprechen generell über den Nachrichtentyp Broadcast (siehe Abschnitt „3.3.1 Startdelimiter erzeugen“) erfolgen.

II. Remote und Standby

Mit dem Objekt 54 wird der Fernsteuerbetrieb (Remote) aktiviert oder der Eingang/Ausgang des Gerätes ein- bzw. ausgeschaltet. Die Verwendung des Objektes und der Maske lassen es zwar zu, daß im Steuerbyte beides gleichzeitig gesetzt/zurückgesetzt werden kann, dies ist aber nicht zu empfehlen. Setzen des Eingangs/Ausgangs erfordert, daß bereits Remote aktiv ist und sollte daher nach der Aktivierung von Remote durch erneutes Senden des Objektes 54 geschehen, mit dem entsprechenden Bit. Beim Beenden des Fernsteuerbetriebes umgekehrt genauso.

Wenn man beide Bits gleichzeitig setzt, kann es vorkommen, daß das Gerät zuerst den Ausgang/Eingang setzen will, bevor Remote aktiv ist und das wird mit einer Fehlermeldung quittiert. Daher ist es sinnvoll nach dem Setzen des Ausgangs/Eingangs dessen Status durch das Objekt 70 zurückzulesen.

### 3.6 Hilfe bei Problemen

**Problem:** Das Gerät läßt sich gar nicht ansprechen bzw. antwortet nicht.

#### Mögliche Ursachen bei USB

- Für die USB-Karte wird ein Treiber benötigt, prüfen Sie ob dieser korrekt installiert ist und ob Sie das Gerät im Windows Gerätemanager gelistet sehen.
- Es wird die falsche Geräteadresse (device node) verwendet, um das Gerät anzusprechen.

#### Mögliche Ursachen bei RS232

- Für die RS232-Karte wird kein 1:1 Kabel verwendet.
- Es wird die falsche Geräteadresse (device node) verwendet, um das Gerät anzusprechen.
- Für Gerät und PC sind unterschiedliche Baudraten oder andere serielle Parameter ungleich eingestellt.
- Das verwendete Kabel ist zu lang für die eingestellte Baudrate (siehe Handbuch zu den Schnittstellenkarten, Abschnitt „2. Technische Daten“).

#### Mögliche Ursachen bei GPIB

- Wenn sich mehrere Geräte am IEEE-Bus befinden sind möglicherweise eine oder mehrere Adressen doppelt belegt.
- Es wird die falsche Syntax verwendet, z. B. reagiert eine elektronische Last auf den Befehl OUTP nicht, da sie einen Eingang hat, oder es werden Befehle verwendet, die für das angesprochene Gerät nicht gültig sind.

#### Mögliche Ursachen bei CAN:

- Es wird die falsche CAN-ID verwendet.
- Falsche Baudrate eingestellt
- Falschen Sample point gewählt (nur bei PSI 9000, siehe Gerätehandbuch).
- Das Gerät ist am Ende des Busses und nicht terminiert.

**Problem:** Es wurden mehrere Anfragen gestellt, aber nicht alle wurden beantwortet

**Ursache:** Die Anfragen wurden zu schnell nacheinander gestellt. In Abhängigkeit von der verwendeten Übertragungsart und -geschwindigkeit und der Verarbeitungszeit des Befehls im Gerät ist zwischen zwei Befehlen eine gewisse Zeit zu warten.

Faustformel: Wartezeit = Übertragungszeit + Ausführungszeit  
Die Ausführungszeit liegt bei typ. 5-20ms, jenachdem ob nur angefragt wurde oder etwas gesetzt werden sollte.

**Problem:** Sollwerte oder Status werden nicht gesetzt

#### Mögliche Ursachen

- Das angesprochene Gerät befindet sich nicht im Fernsteuerbetrieb oder kann nicht in diesen gesetzt werden, weil dies für den momentanen Zustand nicht zulässig ist oder eine andere Bedingung für das Setzen eines Status ist nicht erfüllt.
- Die gesendeten Werte sind falsch (zu groß, zu klein) oder deren Standardwertebereich (0..0x6400 bei Spannung, Strom usw.) ist zusätzlich durch Grenzwerte (nur bei PSI 9000 / PSI 8000) eingeschränkt. Es wird dann eine Fehlermeldung gesendet.

### 3.7 Fehlermeldungen

In der Tabelle in 3.7.2 befindet sich eine Übersicht über mögliche Fehlermeldungen, die vom anzusprechenden Gerät an den PC geschickt werden können. Sie dienen als Hinweis und zur Fehlerfindung.

Fehlermeldungen haben Telegrammformat, d.h. sie bestehen aus Startdelimiter (entfällt bei CAN), Objektnummer (hier als Kennzeichnung für einen Fehler immer **0xFF**), Datenbereich mit Fehlercode und Prüfsumme (entfällt bei CAN).

Beispiel: wenn man z. B. mit Objekt 50 bei einem Gerät die Spannung setzen will und das Gerät nicht im Remote-Modus ist, dann würde sich bei einer Geräteadresse 7 das Fehlertelegramm **C0 07 FF 09 01 CF** ergeben.

#### 3.7.1 Erläuterungen zu einigen Fehlercodes

**Code 0x7:** die Objektnummer im Telegramm ist dem Gerät unbekannt. In so einem Fall die zu sendende Nachricht mit der Objektliste abgleichen.

**Code 0x8:** die Länge des Datenfeldes im Telegramm ist in der [Objektliste](#) definiert. Dieser Fehler kommt z. B. wenn ein Sollwert (immer 2 Bytes bei Typ „Int“) gesendet werden soll, das Datenfeld aber nur ein Byte enthielt. Selbst wenn der Startdelimiter die richtige Telegrammlänge enthält, dient dies zusätzlich zum Schutz davor, daß falsche Werte gesetzt werden.

**Code 0x9:** es wurde z. B. ein Objekt zum Setzen eines Sollwertes gesendet, das Gerät ist aber nicht im Fernsteuerbetrieb. Daher nur Leserecht, kein Schreibrecht.

**Code 0xE:** bei CAN werden Strings gesondert übertragen. Wenn deren Länge größer als 8 Zeichen ist, müssen geteilte CAN-Nachrichten verwendet werden und der Anfang des Datenfeldes im Telegramm das String-Startkennzeichen 0xFF, 0xFE usw. enthalten. Siehe auch „1.9.3 Geteilte Telegramme“.

**Codes 0x30/0x31:** beziehen sich auf Sollwerte. Alle Sollwerte haben eine obere und untere Grenze, die z. B. bei einem Netzgerät einstellbar sind. Standardmäßig ist die obere für z. B. einen Stromsollwert 0x6400 (=100%) und die untere ist 0. Grenzen gelten auch für Zeitwerte.

**Code 0x32:** für einen Zeitwert wurde der falsche Zeitbereich gewählt. Die obere und untere Grenze werden dadurch nicht verletzt, jedoch dieser Fehler erzeugt.

**Codes 0x36:** Bedingungen für den Zugriff nicht eingehalten. Siehe [Objektlisten](#) und die Zugriffsbedingungen in deren Spalte 4.

**Code 0x38:** wird erzeugt, wenn versucht wird auf ein Nur-Lese-Objekt (ro) zu schreiben.

### 3.7.2 Fehlercodes Kommunikation

Fehlercode		
Hex.	Dez.	Beschreibung
01	1	RS232: Paritätsfehler
02	2	RS232: Frame Error (Startbit o. Stopbit nicht erkannt)
03	3	Prüfsumme nicht korrekt
04	4	Startdelimitier falsch
05	5	CAN: max. Nodes überschritten
06	6	Device node falsch / keine Gatewayfunktion
07	7	Objekt nicht definiert
08	8	Objektlänge nicht korrekt
09	9	Schreib-Leserechte verletzt, kein Zugriff
0A	10	Zeit zwischen zwei Bytes zu lang / Anzahl Bytes in Nachricht falsch
0C	12	CAN: geteilte Message abgebrochen
0F	15	Gerät ist in "Local" Modus oder analoger Fernsteuerung
10	16	CAN-Treiber-IC: Stuffing Fehler
11	17	CAN-Treiber-IC: CRC-Summenfehler
12	18	CAN-Treiber-IC: Übertragungsfehler
13	19	CAN: erwartete Datenlänge stimmt nicht
14	20	CAN-Treiber-IC: Puffer voll
20	32	Gateway: CAN Stuffing Fehler
21	33	Gateway: CAN CRC-Summenfehler
22	34	Gateway: CAN Übertragungsfehler
30	48	Obere Grenze des Objektes überschritten
31	49	Untere Grenze des Objektes unterschritten
32	50	Zeitdefinition nicht eingehalten
33	51	Zugriff auf Menüparameter nur bei "Ausgang aus"
36	54	Zugriff auf Funktionmanager/Funktionsdaten verweigert
38	56	Objektzugriff nicht möglich

Legende

- Kommunikationsfehler
- Userfehler
- Interner Fehler

Tabelle: Kommunikationsfehler

### 3.8 Objektlisten

#### Hinweis

Die Objektliste für Ihr Gerät befindet sich in einer externen PDF-Datei und sollte zusammen mit diesem Dokument sein.

Link: [Objektliste Serien PSI 8000 T / DT / 2U / 3U](#)

Link: [Objektliste Serie PSI 9000](#)

Link: [Objektliste Serie PSI 800 R](#)

Link: [Objektliste Serie BCI 800 R](#)

Link: [Objektliste Serien EL 3000 und EL 9000](#)

Link: [Objektliste Serien PS 8000 T / DT / 2U / 3U](#)

Objektlisten sind die Referenz für die Erstellung eigener Applikationen, die unsere Geräte abseits von der Verwendung der angebotenen LabView-VIs steuern sollen. Für LabView-Anwender, die das Telegrammformat selbst umsetzen möchten, dienen die Listen gleichermaßen als Referenz.

#### 3.8.1 Bedeutung der Spalten

Die **1. Spalte** enthält die Objektnummer als Dezimalwert. Diese Nummer muß im Telegramm dem Byte **OBJ** (siehe 1.8) zugewiesen werden. Die Nummer ist einem Befehl gleichzusetzen.

Die **3. Spalte** gibt Auskunft darüber, ob das Objekt nur gelesen (read only, ro), nur geschrieben (write only, w) oder geschrieben und gelesen werden kann (read/write, rw), auch „Setzen“ genannt. Lesen, also reines Anfragen, ist immer zulässig. Man kann es auch Monitoring nennen. Das Setzen von Sollwerten oder eines Status dagegen erfordert eine Freigabe, d.h. das Gerät ist momentan nicht „local“ oder anderweitig gesperrt und die vorherige Aktivierung der Fernsteuerung ist getan (siehe auch 3.2).

Die **4. Spalte** (sofern vorhanden) beschreibt eine besondere Zugriffsbedingung für ein Objekt. Die Ausführung des Objekts ist dann zusätzlich von einer der unten genannten Voraussetzungen abhängig. Ist die Voraussetzung nicht erfüllt, wird das Objekt nicht ausgeführt und das Gerät sendet als Antwort eine Fehlermeldung, die einen Fehlercode enthält. Bedeutung der Bedingungs-werte:

**1** = Der Ausgang/Eingang des Gerätes muß abgeschaltet sein (Objekt wird nur vom Gerät akzeptiert, wenn der Leistungsausgang/ingang auf OFF steht)

**2** = Option „Innenwiderstand“ muß freigeschaltet sein (Objekt wird nur vom Gerät akzeptiert, wenn die Option Innenwiderstandsregelung freigeschaltet ist)

**3** = Übertragung des Funktionsablaufs ist freigeschaltet (Objekt wird nur vom Gerät akzeptiert, wenn es vorher durch ein anderes Objekt angewiesen wurde, daß Daten für den Funktionsmanager gesetzt werden sollen)

**4** = Funktionsmanager aktiviert (Objekt wird nur vom Gerät akzeptiert, wenn der Funktionsmanager aktiv ist, sprich am Gerät über das Menü oder über ein anderes Objekt aufgerufen wurde)

**5** = Funktionsmanager nicht aktiviert (Objekt wird nur vom Gerät akzeptiert, wenn der Funktionsmanager nicht aktiviert ist)

**⚠ Achtung!**

**Es ist generell erforderlich das Gerät VOR dem Senden von Objekten, die Werte im Gerät ändern, in den Fernsteuerbetrieb (Remote) zu setzen.**

Die **5. Spalte** gibt den Typ der Daten im Telegrammteil **Daten** an. Es werden allgemein bekannte Datentypen verwendet.

Die **6. Spalte** gibt die Datenlänge eines Objekts an. Bei Objekten mit dem Datentyp „string“ bezieht sich die Angabe auf die maximal mögliche Länge. Der String muß entweder mit „EOL“ (end of line = 0) abgeschlossen werden oder endet nach der Übertragung der maximal angegebenen Bytes. Strings werden bei CAN in bis zu drei geteilten Nachrichten übertragen. Siehe auch „1.9 Telegrammaufbau bei CAN“.

Die **7. Spalte** wird zur Maskierung von Daten des Typs „char“ verwendet. Die Maske (immer 1. Datenbyte im **Datenfeld**) gibt an, welche Bits überschrieben werden sollen. Siehe auch „3.3.2 Die Maske“.

Die **8. + 9. Spalte** erläutern genauer die einzelnen Informationen im Telegrammteil **Daten**.

Manche Objekte verwenden ein Zeitformat. Lesen Sie im Abschnitt „3.4 Das Zeitformat“ mehr über dessen Definition.

**3.8.2 Objektbeispiele- und erläuterungen**

☞ Eine Beschreibung der Spalten finden Sie im Abschnitt 3.8.1.

☞ Alle Angaben sind dezimal, sofern nicht mit Präfix „0x“ als hexadezimal gekennzeichnet.

**I. Funktionsmanager (Objekte 56, 58, 73-75, 78, 90-146)**

Beim Funktionsmanager ist die Reihenfolge der Objekte wichtig. Da Ansteuerung und Konfiguration hier recht aufwendig sind, finden Sie näheres dazu in einer Application Note (AN001) im Ordner „\manuals\application notes“ auf der zur Schnittstellenkarte gehörigen CD oder auf unserer Webseite.

**II. Objekt 54**

Die vorrangige Funktion dieses Objektes ist es, das Gerät in den Fernsteuerbetrieb zu versetzen bzw. diesen zu verlassen, oder den Eingang/Ausgang (je nach Gerät) ein- oder auszuschalten. Die Maske muß beim Senden als erstes Datenbyte mitgesendet werden und die Bits maskieren, die gesetzt/gelöscht werden soll. Beispiel: Fernsteuerung aktivieren: Bit 4 ->Wertigkeit = 0x10 -> Maske 0x10 -> Steuerbyte auch 0x10. Das Objekt 54 enthält also die Daten 0x1010. Fernsteuerung deaktivieren genauso: Maske 0x10 -> Steuerbyte 0x00 -> Daten: 0x1000.

**⚠ Achtung!**

**Um Kollisionen der Bitfunktionen zu vermeiden, sollte stets nur ein Bit pro Telegramm verändert werden, auch wenn es möglich ist, mehrere Bits auf einmal zu verändern!**

Beim Auslesen wird die Hauptmaske, eine Kombination der Einzelmasken (siehe [Objektlisten](#)), auch mitgesendet, hat aber nur informativen Charakter.

**III. Objekt 56**

Die Bits dürfen hier nur einzeln gesetzt sein. Ansonsten werden die gewünschten Aktionen nicht oder nicht richtig ausgeführt.

**IV. Objekt 73**

Der Zeitstempel ist nur bei Nutzung des Funktionsmanagers verfügbar, ansonsten 0. Er gibt einen Zählwert zurück, der die abgelaufene Zeit der Funktion in 2ms-Schritten darstellt. Dieser Zähler startet nach 65536 x 2ms wieder bei 0.

**V. Objekt 77**

Das Auslesen des Alarmbuffers löscht selbigen (bei EL 3000, PS 8000). Bei anderen Serien muß das Löschen des Buffers durch Setzen eines Bits in Objekt 54 initiiert werden (PSI 9000, PSI 8000, PSI 800R, BCI 800R). Da er sehr klein ist (3 Ereignisse), werden die ersten drei aufgetretenen Fehler gespeichert und weitere überschreiben jeweils den zuletzt aufgetretenen auf Position „Letzter Alarmtyp“.

Beispiel für einen Fehler: 0x0120 -> bedeutet mit Fehlertyp 0x01, daß der Fehler noch anliegt und der Fehlercode 0x20 besagt (siehe Tabelle in Abschnitt 3.9), daß es ein Übertemperaturfehler im oberen Leistungsteil eines Netzgerätes PSI 9000 (9HE-Gerät) ist.

**VI. Objekte 39-47**

Beziehen sich auf Abschnitt 7.6 der PSI 9000 / PSI 8000 Handbücher. Ereignisse, die durch die überwachten Werte ausgelöst werden, erzeugen je nach Konfiguration (Objekte 44-46) Alarme, Warnungen oder Meldungen im Alarmbuffer, der wiederum mit Objekt 77 (siehe oben) ausgelesen wird.

Die Zeit, die hier vorzugeben ist, ist eine Verzögerungszeit, nach der ein Ereignis ausgelöst wird. Zeitbereich 2ms...99h:59m. Zeitformat siehe Abschnitt 3.4.

**VII. Objekte 21-29**

Hiermit werden Sollwertsätze geladen (preset list), wie Sie sie auch am Gerät auswählen und konfigurieren können. Jedoch können hier keine Sollwertsätze ausgewählt werden, um ferngesteuert Sollwertsprünge zu erzeugen. Dazu sind dann andere Objekte zu verwenden, die Sollwerte setzen.

**3.8.3 Über die Benutzerprofile**

Die Geräteserien PSI 8000 und PSI 9000 haben mehr als ein Benutzerprofil. Bis zu vier können abgelegt und abgerufen werden. Jedes Profil enthält z. B. eine Liste mit Sollwertsätzen. Das bedeutet, daß der Anwender zu beachten hat, welches Profil gewählt ist, wenn Werte für die Sollwertsätze geschrieben bzw. gelesen werden sollen.

Die zu einem Profil gehörigen Werte (und somit Objekte) sind in den [Objektlisten](#) farbig markiert, sofern es sich um eine Geräteserie handelt, die mehr als ein Profil hat.

### 3.9 Gerätefehler, Alarmcodes und Alarmtypen

Fehlercode Error code	Kürzel Abbr.	Fehlertext oder -beschreibung / Error text or description
0		Kein Fehler / No error
1	OV	Überspannung am Ausgang (Eingang) / Overvoltage at output (input)
2	OT	Übertemperatur im Gerät / Overtemperature inside the device
3	SYS	Systemfehler / System error
4	U>	Obere Spannungsgrenze überschritten / Upper voltage threshold exceeded
5	U<	Untere Spannungsgrenze unterschritten / Lower voltage threshold exceeded
6	I>	Obere Stromgrenze überschritten / Upper current threshold exceeded
7	I<	Untere Stromgrenze unterschritten / Lower current threshold exceeded
8	SIO2	System Link Mode: Kommunikation gestört / Communication disturbed
9	MS1	System Link Mode: Ein oder mehrere Gerät sind "offline" / One or more units are offline
10	S-OV	System Link Mode: Slave meldet Überspannung / Slave is reporting an overvoltage
11	S-OT	System Link Mode: Slave meldet Übertemperatur / Slave is reporting overtemperature
12	S-PH	System Link Mode: Slave meldet Netzfehler / Slave is reporting mains voltage error
13	S-PD	System Link Mode: Slave ist in Leistungsbegrenzung / Slave reduces max output power
14	S-?	System Link Mode: Slave antwortet nicht / Slave does not answer
17	F01	Interner Fehler / Internal error
19	F03	Interner Fehler / Internal error
20	CAN	CAN: Kommunikation gestört / Communication disturbed
21	FCT	Funktionsmanager: Funktion konnte nicht gesetzt werden / Function manager: function could not be set
22	UDU	Überwachung Sprungantwort: Anstieg U / Step response supervision: U rise
23	UDD	Überwachung Sprungantwort: Abfall U / Step response supervision: U fall
24	IDU	Überwachung Sprungantwort: Anstieg I / Step response supervision: I rise
25	IDD	Überwachung Sprungantwort: Abfall I / Step response supervision: I fall
26	PDU	Überwachung Sprungantwort: Anstieg P / Step response supervision: P rise
27	PDD	Überwachung Sprungantwort: Abfall P / Step response supervision: P fall
28	PH1	Phasenausfall oberes Leistungsteil / Phase loss of upper power stage
29	PH2	Phasenausfall unteres bzw. mittleres Leistungsteil / Phase loss of lower resp. middle power stage
30	PH3	Phasenausfall unteres Leistungsteil / Phase loss of lower power stage
31	OT1	Übertemperatur oberes Leistungsteil / Overtemperature of upper power stage
32	OT2	Übertemperatur unteres bzw. mittleres Leistungsteil / Overtemperature of lower resp. middle power stage
33	OT3	Übertemperatur unteres Leistungsteil / Overtemperature of lower power stage
34	CC	nur UIR Betrieb: Betrieb des Stromreglers nicht erlaubt
35	CP	nur UIR Betrieb: Betrieb des Leistungsreglers nicht erlaubt
36	-	Bat.temp. out of range (Batterietemperatur zu hoch)
37	-	Bat.temp. out of range (Batterietemperatur zu niedrig)
38	-	Bat.voltage out of range (Batteriespannung zu hoch)
39	-	Battery deeply discharged ( Batteriespannung zu niedrig bzw. tiefentladen)
40	-	Cell fault in battery (Zellenschluß)
41	-	Temp sensor fault (Temperaturfühler fehlt oder defekt)
42	-	Reverse polarity (Batterie verpolt)
43	-	Battery not connected (keine Batterie angeschlossen)

  nur bei Mehrphasengeräten / only at multi-phase models

  Gilt für BCI 800 R Serie / Applies only to BCI 800 R series

Tabelle: Alarmcodes

**! Hinweis**

Bei bestimmten Geräteserien können zwei Fehler gleichzeitig auftreten. Zum Beispiel für der Fehler „OV“ auch zu einem „Powerfail“ (PF, Nummer 28)

#### Wie funktioniert das Alarmmanagement?

Tritt ein Fehler nach nebenstehender Tabelle auf, wird dieser nach Typ eingeordnet und in einem 3stufigen Alarmbuffer abgelegt. Dieser kann mittels Objekt 77 ausgelesen werden. Siehe auch die [Objektlisten](#) für die Aufteilung.

Dabei gilt, daß der erste Fehler durch einen weiteren im Buffer nach hinten geschoben wird. Ist der Buffer voll, wird der zuletzt aufgetretene Fehler immer wieder überschrieben. Das Auslesen des Buffers leert diesen entweder automatisch (EL 3000/9000, PS 8000) oder muß durch das Setzen eines Bits in Objekt 54 veranlaßt werden (PSI 9000, PSI 8000, PSI 800 R, BCI 800 R).

#### Was ist ein Alarmtyp?

Über die Unterscheidung bzw. die Bedeutung von Alarmen, Warnungen und Meldungen bei Netzgeräten der Serie **PSI 9000, PSI 8000, PSI 800R**, sowie Batterieladern der Serie **BCI 800 R** lesen Sie bitte im Benutzerhandbuch Ihres Gerätes nach. Die elektronischen Lasten der Serien **EL3000** oder **EL 9000** benutzen nur Alarme und nur die Alarmtypen 0x01 bzw. 0x02.

#### Alarmtypen:

- 0x01 - Alarm ist momentan aktiv
- 0x02 - Alarm ist nicht mehr aktiv
- 0x10 - Warnung momentan aktiv
- 0x20 - Warnung nicht mehr aktiv
- 0x40 - Meldung steht an

Der Fehlertyp wird bei Anfrage, ob Fehler aufgetreten sind, vom Gerät zusammen mit dem Fehlercode gesendet und kann dann ausgewertet werden. Warnungen und Meldungen habe geringere Priorität als Alarme, werden ggf. überschrieben und sind daher weniger wichtig bzw. können sogar ignoriert werden.

## 4. Profibus (IF-PB1)

### 4.1 Allgemeines

Der Profibus, kurz für Programmable Field Bus, ist ein herstellerunabhängiger Feldbus-Standard für einen weiten Anwendungsbereich in Meßtechnik und Automatisierung. Durch die Standardisierung nach IEC 61158 garantiert Profibus die einwandfreie Kommunikation zwischen Geräten unterschiedlicher Hersteller. Verschiedenste Schnittstellen für z. B. PCs, SPS usw. sind am Markt verfügbar.

Die Schnittstellenkarte IF-PB1 unterstützt den Profibus-DP (DP = dezentrale Peripherie), welcher speziell für die schnelle Kommunikation auf der Feldebene vorgesehen ist. Die Datenübertragung erfolgt mittels einer RS485 Verbindung und arbeitet mit Geschwindigkeiten bis zu 12 MBaud.

Ein Profibus-Netz ist in Segmente eingeteilt. Jedes Segment besteht aus max. 32 Teilnehmern. Teilnehmer sind Master, Slaves, Repeater und Konverter. Repeater und Konverter benötigt man bei einem Netz, das über die Teilnehmerzahl von 32 hinausgeht. Jeder Profibusteilnehmer bekommt eine feste Adresse. Daraus folgt, daß im ersten und letzten Segment max. 31 Teilnehmer und in jedem mittleren Segment 30 Teilnehmer zulässig sind. Reserviert sind die Adressen 0 (üblicherweise für ein Programmiergerät), 126 und 127.

Fazit: Insgesamt können an einem Profibus 125 Schnittstellenkarten des Typ IF-PB1 betrieben werden.

In Abhängigkeit von der Leitungslänge kann zwischen Baudraten von 9,6kBaud bis zu 12MBaud gewählt werden. Die Verantwortlichkeit in der Leitungslänge obliegt dem Anwender. Zu beachten ist, daß bei der Wahl einer falschen Busgeschwindigkeit eine fehlerfreie Kommunikation nicht gewährleistet ist.

Die Schnittstellenkarte IF-PB1 unterstützt alle möglichen Übertragungsraten, besitzt eine galvanische Trennung zur Feldbusseite von 1000V DC und wird bei Einsatz im dafür vorgesehenen Einschubplatz von den Geräten mit Spannung versorgt.

### 4.2 Technische Daten

Bestellbezeichnung	IF-PB1
Kommunikationsprofile	Profibus DP-V0 Profibus DP-V1 Klasse 1
Schnittstellentyp	9-polige Sub-D-Buchse
Kommunikationsmedium	RS485
Netzwerktopologie	Linie (ohne Repeater), Baum/Linie (mit Repeatern)
Teilnehmeranzahl	max. 31 (ohne Repeater), max 125 (mit Repeatern)
Teilnehmertyp	Slave
PNO-Ident-Nummer	A000 (hex)

### 4.3 Unterstützte Geräteserien

Unterstützt werden zurzeit (Stand 08/2012) folgenden Geräteserien:

- PSI 9000 (alte Serie bis 2012)
- PSI 8000
- PS 8000

### 4.4 Master-Slave-Kommunikation

#### 4.4.1 Zyklisch

Der Profibus-DP unterscheidet zwischen Master- und Slaveteilnehmern. Der Master steuert die Kommunikation auf dem Bus und fordert die ihm zugeteilten Slaves auf, Daten zu senden oder zu empfangen. In einem typischen Master-Slave-System werden Eingangs-, Ausgangs- und Diagnosedaten zyklisch zwischen dem Master und allen zu ihm zugeordneten Slaves ausgetauscht. Der Master (z. B. eine SPS) hält die jeweils gelesenen Daten der Slaves in seinem internen Speicher für das Steuerprogramm bereit. Die Ausgabedaten werden mit dem nächsten Übertragungszyklus zu den Slaves übertragen.

Auf diese Weise liegt im Master immer, mit der Verzögerung einer Zykluszeit, ein Abbild der Daten aus den ihm zugeordneten Slaves.

Im Fall der IF-PB1 werden stetig (=zyklisch) die Istwerte und der Gerätezustand des Geräts an den Master übertragen.

#### 4.4.2 Azyklisch

Ergänzend zum zyklischen Datenaustausch können Daten auch azyklisch übertragen werden. Dies bietet den Vorteil, unabhängig von der zyklischen Übertragung auf einzelne Parameter bzw. Sollwerte des Feldgerätes zuzugreifen. Die azyklische Übertragung kann dann auf die wesentlichen Daten reduziert werden. Hiermit wird die durchschnittliche Busbelastung erheblich gesenkt.

### 4.5 Verkabelung / Terminierung

Die Verbindung der Schnittstelle IF-PB1 zu weiteren Slave-Geräten oder einem Master-System erfolgt über eine 2-adrige, abgeschirmte Leitung. Zwei Varianten, Typ A und Typ B, der Busleitung sind in der IEC 61158 spezifiziert. Der Leitungstyp B ist nicht zu empfehlen und sollte bei Neuinstallationen nicht verwendet werden.

Leitungstyp A:

- Leitungsaufbau: Twisted Pair, geschirmt 1x2
- Wellenwiderstand  $R_w$ : 135-165Ω bei 3..20MHz
- Kapazitätsbelag: < 30pF/m
- Schleifenwiderstand  $R$ : 110Ω/km
- Adernquerschnitt: > 0,34mm<sup>2</sup>

An ihr werden die ankommende Busleitung (ingoing) und die weiterführende Busleitung (outgoing) angeschlossen. Die Leitungen „Ingoing“ und „Outgoing“ dürfen nicht vertauscht werden, da ein gebräuchlicher Profibus-D-Sub-Stecker den Kabelabgang ausschaltet, wenn an einem Teilnehmer terminiert wird. Beim Trennen (Abschaltung oder Stecker abziehen) einer IF-PB1 vom Bus wird die Busleitung nicht unterbrochen. Dieses Szenario trifft beim Einsatz von mehreren Netzgeräten mit IF-PB1 zu.

Die Aktivierung der Abschlußwiderstände verhindert Reflexionen an den Leitungsenden und sorgt für einen sauberen Ruhepegel am Bus.

### 4.6 Übertragungsgeschwindigkeiten und Laufzeiten

Die verfügbaren, für die Schnittstellenkarte IF-PB1 am Gerät einstellbaren Übertragungsgeschwindigkeiten sind in der Tabelle in 4.6.1 aufgelistet. Für den Zugriff auf die schnittstellenspezifischen Parameter am Gerät siehe Gerätehandbuch bzw. allgemeines Schnittstellenhandbuch.

Die **Übertragungsgeschwindigkeit** bestimmt lediglich, wie schnell Daten zwischen dem Profibus-Slave (IF-PB1 mit Profibus-Gateway-Chip) und dem Profibus-Master übertragen werden, jedoch nicht, wie schnell die Daten zwischen dem Gerät und dem Profibus-Gateway (siehe schematische Darstellung unten zur Verdeutlichung) übertragen oder wie oft die Daten selbst aktualisiert werden. Die Übertragung zwischen Gerät und Profibus-Gateway findet mit einer festen Baudrate von 57600 statt. Die Aktualisierungszeit der Daten hängt in erster Linie vom Gerät ab und variiert von Serie zu Serie.

Prinzipiell legt der Slave bestimmte Daten (Istwerte, Status) über DP-V0 in einem festgelegten Intervall (hier: 1x pro Zyklus) auf den Bus. Dazu werden aus dem Gerät gelesene Daten zwischengespeichert. Aktualisiert werden diese Daten auch pro Zyklus einmal. Ein Zyklus dauert ca. 660ms.

Desweiteren, wenn Daten über DP-V1 an das Gerät geschickt werden sollen (Sollwerte etc.), werden diese auch einmal pro Zyklus an das Gerät weitergegeben.

Daraus ergeben sich gewisse **Laufzeiten** bis z. B. ein Sollwert am DC-Ausgang/DC-Eingang des Gerätes „ankommt“. Diese ergeben sich aus Übertragungszeit vom Master (abhängig von Baudrate) plus Verarbeitungszeit im Profibus-Gateway (max. 1 Zyklus von 660ms) plus Verarbeitungszeit im Gerät (einige Millisekunden). Wenn man zu einem gesendeten Sollwert den zugehörigen Istwert via DP-V0 empfangen möchte, so kann das im schlimmsten Fall drei Zykluszeiten dauern. Warum? Weil im ersten Zyklus der Wert an das Gerät gesendet wird und der im selben Zyklus vom Gerät gelesene Istwert noch nicht dem gesendeten Sollwert entspricht. Im nächsten Zyklus kann der ausgelesene Istwert dem vorher gesendeten Istwert entsprechen oder, wenn der Istwert (Spannung, Strom oder Leistung) noch dabei ist, sich dem Sollwert zu nähern, wird er erst im dritten Zyklus dem Sollwert entsprechen.

Ähnlich verhält es sich mit einem Vorgang „Ausgang aus setzen“ an das Gerät schicken und danach Status auslesen, ob Ausgang auch aus ist.

In der Standardkonfiguration der IF-PB1-Schnittstelle sind für einen Vorgang „Sollwert setzen und danach Istwert auslesen, bis Istwert = Sollwert“ Zeiten von 1,2s...1,5s typisch. Das kann nur durch dafür optimierte Scripts für den Profibus-Gateway auf ein Minimum von etwa 0,35s für solch einen Vorgang reduziert werden. Diese Scripte liefern weiterhin pro Zyklus einen neuen Istwert, jedoch nur jeden zweiten oder dritten Zyklus einen neuen Status bzw. Alarmzustand. Die optimierten Scripte können vom Anwender auf das Profibus-Gateway aufgespielt werden und sind auf Anfrage erhältlich.

#### 4.6.1 Einfluß der Kabellänge

Bei der reinen Datenübertragung zwischen Profibus-Master und Profibus-Slave bestimmt die Datenleitungslänge zusätzlich die max. nutzbare Übertragungsrate (Baudrate) laut dieser Tabelle, die gängige Profibusgeschwindigkeiten auflistet, von denen einige von der IF-PB1-Schnittstelle unterstützt werden:

Baudrate (kBit/s)	Max. Kabellänge (m)
9.6	1200
19.2	1200
45.45	1200
93.75	1200
187.5	1000
500	400
1500	200
3000	100
6000	100
12000	100

### 4.7 Betriebsmodi

Über ein Kodierfeld, welches sich auf der Schnittstellenplatte befindet, ist das Gerät bzw. die Profibus-Schnittstelle in die verschiedenen Betriebsmodi zu setzen.

Diese dienen der Profibus-Kommunikation, der Wartung bzw. dem Service und den Update-Möglichkeiten.

#### RS 485/Sub-D-Port:

- Profibus-Anwendung (Feldbuskommunikation)

#### USB-Port (Serieller Port):

- Script-Update über WINGATE Software oder Script Programming Tool (SPT)
- Firmwareupdate des/der Gerätecontroller (Update Tool)
- Firmwareupdate des Profibus-Stacks über Firmware Download Tool (FDT)

Die IF-PB1 ist mit einer Leuchtdiode ausgestattet, die erlischt wenn die Schnittstellenkarte für den Datenaustausch mit einem Master-System bereit ist. Sie leuchtet grundsätzlich während der Initialisierungsphase der Schnittstelle und bei Fehlern, die bei der Initialisierungsphase auftreten.

Beim Einschalten des Gerätes leuchtet sie zunächst, bis das Gerät für den Datenaustausch bereit ist.

Wenn das/ein Mastersystem vom Bus getrennt wird leuchtet sie ebenfalls. Eine detaillierte Beschreibung des momentanen Status der Kommunikation bieten in der Regel die Software-Konfiguratoren der Master-Systeme.

### 4.8 Profibus-Slave-Adresse

Jeder Teilnehmer im Profibus ist über eine eindeutige Adresse anzusprechen. Dazu muß bei der Inbetriebnahme an jedem Netzgerät zunächst die Profibusadresse (nicht zu verwechseln mit der Geräteadresse „device node“), wie im Gerätehandbuch beschrieben, eingestellt werden. Es dürfen demnach keine Adressen doppelt vergeben werden. Am sinnvollsten ist es, die Adresse(n) in aufsteigender Reihenfolge zu vergeben, wobei man für einen PC/PG während der Inbetriebnahme die Adresse 0 vorsehen sollte.

Die Zuordnung aller Adressen an die Profibusteilnehmer korrespondiert mit der Adressvergabe bei der Projektierung an einem Mastersystem oder am PC. Somit ist eine eindeutige Identifizierung der Netzgeräte gegeben.

**Hinweis:** Die über das Gerätemenü eingestellte Adresse wird von der IF-PB1 entweder durch ein Aus- und Wiedereinschalten des Gerätes oder durch einen Reset der Schnittstellenkarte über den Reset-Knopf, der sich an der Schnittstellenblende befindet, übernommen.

## 4.9 Kommunikation mit dem Master

### ! Hinweis

Werte, so wie mit den unten beschrieben Datagrammen übertragen, sind unterschiedlichen Typs. Soll- und Istwerte sind z. B. vom Typ Integer und enthalten einen Prozentwert. Siehe Abschnitte 1.7 und 3.3 - 3.8 für weitere Informationen.

### 4.9.1 Zyklische Feldbusdaten

Zyklische Feldbusdaten, die vom Slave (IF-PB1) an den Master übertragen werden, sind in diesem Fall Istwerte und der Gerätezustand. Dies geschieht jedoch nur, wenn die Konfiguration seitens des Masters von der IF-PB1 akzeptiert wurde. Die Art der Daten des zyklischen Datenaustauschs sind der IF-PB1 mit dem Konfiguriertelegramm bekannt gemacht worden. Die Datenbreite des jeweiligen Moduls kann aus einer, der Geräteserie entsprechenden, Objektliste entnommen werden. Sie ist auch mit Hilfe eines Hardwarekonfigurators aus den Eigenschaften der einzelnen Module in der Steckplatzbelegung zu entnehmen. Sie entspricht der Adressbreite bei z. B. einer SPS-Steuerung.

Desweiteren gelten die vorgegebenen Regeln in den Objektlisten bezüglich der Datenformate und Auflösung der Istwerte des Netzgerätes. Dies muß bei der Entwicklung des Anwenderprogramms berücksichtigt werden.

### 4.9.2 Azyklische Feldbusdaten

Die azyklische Übertragung zwischen Master und Slave erfolgt nur auf Anforderung der Masters. Die IF-PB1 unterstützt die Kommunikation zu einem Klasse 1 Master. Für den azyklischen Datenaustausch sind die in 4.9.2.1 aufgeführten Datensätze geeignet (siehe auch Objektliste des Netzgerätes), die in unterschiedliche, funktionelle Gruppen eingegliedert werden können.

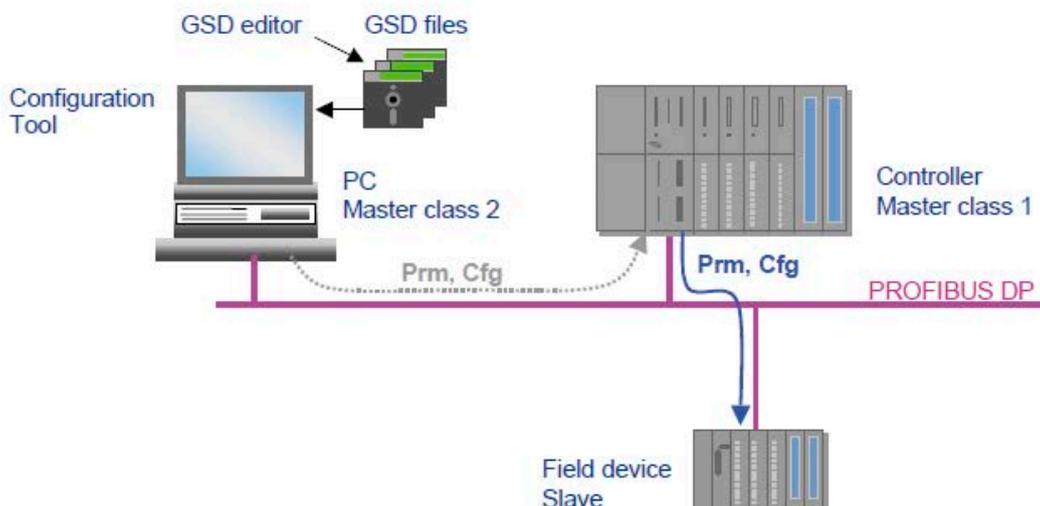


Abbildung 1: Profibus-Schema

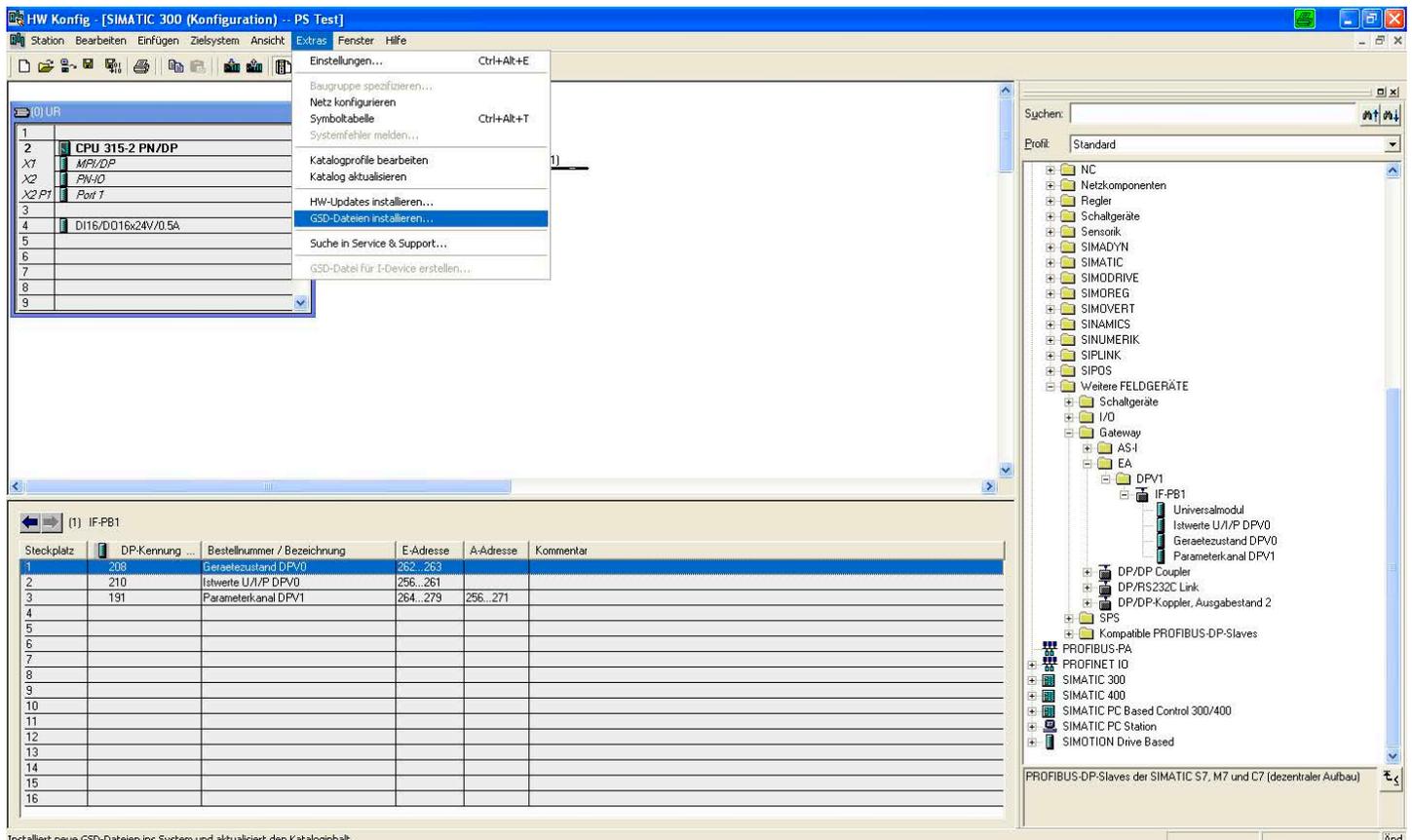
4.9.2.1 Datensatzgruppen

Geräteinformationen DP-V1 (Ausgänge vom Slave)

Datensatz	Slot	Index	Länge	Typ	Objekt in Liste *	Zugriffsart
Gerätetyp	3	0	16	String	0	Read
Seriennummer	3	1	16	String	1	Read
Nennspannung	3	2	4	Float	2	Read
Nennstrom	3	3	4	Float	3	Read
Nennleistung	3	13	4	Float	4	Read
Artikelnummer	3	4	16	String	6	Read
Firmwareversion	3	5	16	String	9	Read
Aktuelle Sollwerte (U,I,P)	3	9	6	Integer	72	Read
Gerätemeldungen / Alarme	3	10	6	Byte	77	Read

Steuerung DP-V1 (Eingänge zum Slave)

Datensatz	Slot	Index	Länge	Typ	Objekt in Liste *	Zugriffsart
Zustand setzen	3	6	2	Byte	54	Write
Sollwert Spannung	3	7	2	Integer	50	Write
Sollwert Strom	3	8	2	Integer	51	Write
Sollwert Leistung	3	15	2	Integer	52	Write



Installiert neue GSD-Dateien ins System und aktualisiert den Kataloginhalt.

Abbildung 2: Vorgesehene Slot-/Modulkonfiguration für den IF-PB1-Slave

Meldungen und aktuelle Werte DP-V0 (Ausgänge vom Slave)

Datensatz	Slot	Index	Länge	Typ	Objekt in Liste *	Zugriffsart
Istwerte (U,I,P)	2	-	6	Integer	71	Read
Gerätezustand	2	-	2	Byte	70	Read

\* Siehe externe Objektlisten (PDF) und siehe Abschnitt 3.8

4.9.3 Anwendung der Datensätze

Möchte der Anwender Datensätze azyklisch lesen bzw. senden, d.h. mit DP-V1, muß der gewählte Datensatz über seinen INDEX (siehe Tabellen oben), den Slot und eine ID (wo benötigt, ergibt sich aus der Adresse des Modulslots) im Anwenderprogramm angesprochen werden. Weiterhin wird beim Lesen die Länge der zu lesenden Daten gebraucht. Die Länge ist ein Eingangsparameter der normierten Systemfunktionsbausteine. Die zu schreibenden Daten selbst sind hexadezimale Bytes. Inhalt und Anzahl ist mit den **Objektlisten** definiert, in Anlehnung an das Kommunikationsprotokoll das benutzt wird, wenn eine andere Schnittstelle im Gerät zum Einsatz kommt. In der Profibus-Nachricht an den Slave ist hier jedoch nur der reine Datenteil enthalten, siehe auch die Beschreibung in den Abschnitten 3.2 und 3.3.2.

**4.9.3.1 Beispiele**

1) Fernsteuerung aktivieren/deaktivieren

Die Fernsteuerung zu aktivieren ist zwingend nötig, wenn Sie das Gerät steuern wollen. Das ändert den Zustand des Gerätes, also wird Datensatz „Zustand setzen“ benötigt. Dieser bezieht sich auf Objekt 54 und hat eine Länge von 2 Bytes. Laut Objekt 54 wird für das Ein/Aus der Fernsteuerung das Maskenbyte 0x10 benötigt und das Steuerbyte 0x10. Demzufolge werden auf Slot 3 bzw. ID 264 (Adresse von Slot 3, laut Standardkonfiguration) und Index 6 die beiden Bytes 0x10 0x10 geschickt. Zur Deaktivierung der Fernsteuerung müsste man dann 0x10 0x00 schicken.

2) DC-Ausgang ein- oder ausschalten

Den Ausgang ein-oder ausschalten ändert den Zustand des Gerätes, also wird Datensatz „Zustand setzen“ benötigt. Dieser bezieht sich auf Objekt 54 und hat eine Länge von 2. Laut Objekt 54 wird für das Setzen des Ausganges das Maskenbyte 0x01 benötigt und das Steuerbyte 0x01. Demzufolge werden auf Slot 3 bzw. ID 264 (Adresse von Slot 3, laut Standardkonfiguration) und Index 6 die beiden Bytes 0x01 0x01 geschickt. Um den Ausgang auszuschalten, müsste man dann 0x01 0x00 an Slot 3, Index 6 schicken.

3) Spannung auf 40V setzen

Hierzu wird ein Spannungswert als Prozentwert benötigt. Mal angenommen, Sie hätten ein 80V-Gerät, dann wären 40V Sollwert genau 50%. Bei einem 720V-Gerät wären 40V dann 5,55%. Für die Umrechnung der Sollwerte in und aus dem Prozentwert siehe Abschnitt „1.7 Sollwerte und Istwerte umrechnen“. Mit 50% ergäbe sich ein Hexwert 0x3200, der dann über Slot 3 bzw. ID 264 (Adresse von Slot 3, laut Standardkonfiguration), Index 7 und den beiden Bytes 0x32 0x00 an das Gerät geschickt wird.

**4.10 Projektierung am Beispiel einer Siemens Steuerung (SPS)**

**4.10.1 Einbindung des Netzgerätes an den Profibus**

Um ein Netzgerät mit Hilfe der IF-PB1 an den Profibus anzubinden, benötigt man einen Hardwarekonfigurator, der Bestandteil des Softwarepakets von der Fa. Siemens ist. Weitere Konfiguratoren bieten die Firmen Beckhoff und Hilscher an. Zusätzlich ist eine GSD (Generic Station Device) zu verwenden, die auf der bei der Schnittstellenkarte IF-PB1 mitgelieferten CD zu finden ist. Die Datei steht auch zum Download unter dem Namen PBPSA000.GSD bzw. PB-PSA000.GSE auf der Webseite des Geräteherstellers bereit.

Mit Hilfe dieser Datei werden dem Anwenderprogramm im Konfigurator die Kommunikationsmodule des Netzgerätes und die notwendigen Parameter bekanntgemacht. Der zyklische Datenverkehr auf dem Bus wird definiert.

**4.10.2 GSD-Datei-Installation**

Die mitgelieferte GSD Datei wird in der Mastersoftware über den Menübefehl „Extras -> GSD-Dateien installieren“ im System installiert. Die Abbildung 3 zeigt sich dem Anwender bei der Installation der GSD-Datei.

Nach erfolgreicher Installation ist die Schnittstellenkarte mit ihren Modulen im Hardwarekatalog unter der Rubrik „Profibus-DP -> Weitere Feldgeräte -> Gateway -> <firma> ->IF-PB1“ zu finden. Siehe auch Abbildung 4.

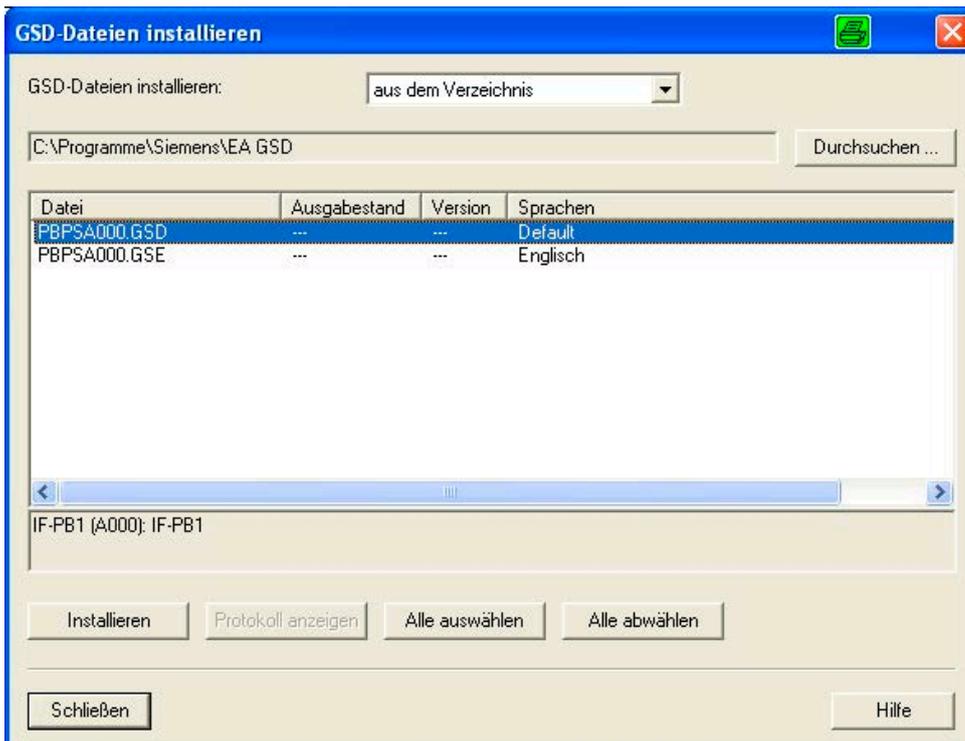


Abbildung 3: GSD-Installation

### 4.10.3 Platzierung der Module

Die Module, die unter dem o. g. Pfad zu finden sind, müssen nun der Steuerung für den Datenaustausch, z. B. über Drag & Drop mit Hilfe der Steckplatzliste, bekannt gemacht werden. Für eine vollständige Kommunikation mit der Schnittstellenkarte sind die Module wie folgt zu platzieren.

- Steckplatz 1 : Istwerte U/I/P DPV0
- Steckplatz 2 : Gerätezustand DPV0
- Steckplatz 3 : Parameterkanal DPV1

Siehe auch Abb. 4 unten.

Bei Nichtverwendung des Moduls Parameterkanal sind eine Steuerung und somit auch keine sinnvolle Nutzung des Netzgeräts mehr möglich.

### 4.10.4 Adressierung der zyklischen Module

Grundsätzlich gilt für die Module des zyklischen Datenaustausches:

- Die Module aus dem Hardwarekatalog bekommen eine logische Adresse
- Die logischen Adressen bilden ein Teil der Gesamtperipherie der Steuerung
- Die Peripherieadressen der einzelnen Module dürfen nicht mit einander kollidieren, also nicht doppelt vergeben werden
- Die jeweils vergebene Peripherieadresse eines Moduls ist für das Ansprechen der Daten im Anwenderprogramm zu benutzen

### 4.10.5 Adressierung des azyklischen Moduls

Für das Modul des azyklischen Datenaustauschs gilt:

- Die Peripherieadresse(n) dürfen sich nicht mit denen der zyklischen Module überschneiden.
- Da es sich bei diesem Modul um ein so genanntes Mischmodul handelt (Eingang/Ausgang) dürfen sich auch die Adressen der Ein- und Ausgänge nicht überschneiden. Dies ist in der Eigenschaft der Siemens-Kommunikationsbausteine (SFB52 und SFB53) für den azyklischen Datenverkehr begründet
- Für das Ansprechen der Daten über eine nach der PNO AK 1131 normierte Schnittstelle im Anwenderprogramm wird zunächst nach einer ID verlangt, die die niedrigere der beiden Adressen darstellt. Die Adresse ist die Basis und die erste von zwei Informationen, die für eine Nutzung des azyklischen Datenaustauschs gebraucht werden
- Die zweite Information ist die Datensatznummer, die gelesen bzw. geschrieben werden soll. In der Norm wird hier von dem sogenannten INDEX gesprochen

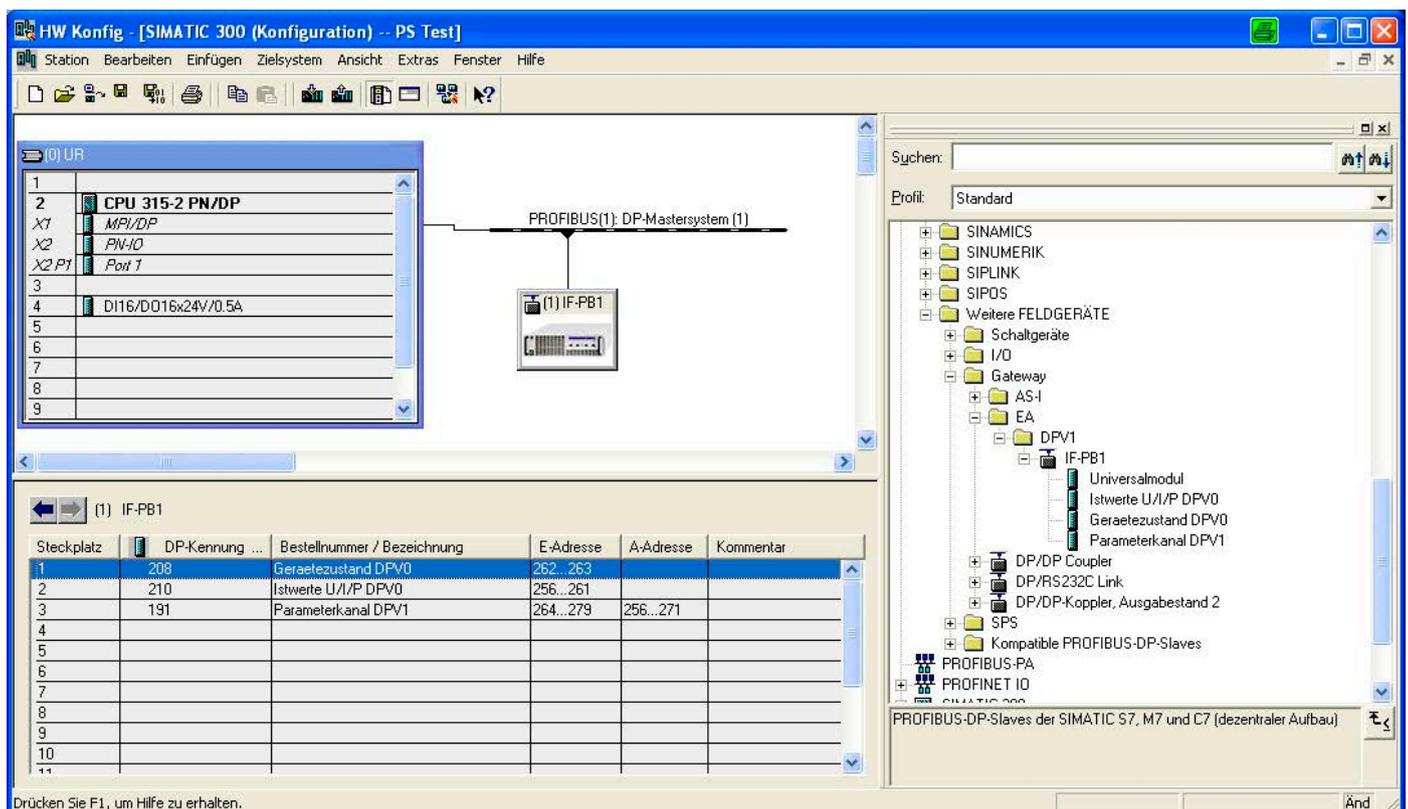


Abbildung 4: Erforderliche Platzierung der Module

## 4.11 Diagnose

### 4.11.1 Standarddiagnose

Die Standarddiagnose ist ein Telegramm, das aus 6 Bytes besteht. Dieses Telegramm wird während der Initialisierung der IF-PB1 und vom Anwender unbemerkt mehrmals zwischen der Steuerung und der Schnittstellenkarte ausgetauscht und kann mit Hilfe eines geeigneten Engineering Tools ausgewertet werden. In STEP7 geht das über „Zielsystem -> Erreichbare Teilnehmer anzeigen“ und kann durch Anwahl des entsprechenden Feldgerätes über „Zielsystem -> Diagnose/Einstellung -> Baugruppenzustand“ angezeigt werden (s. Abb.). Desweiteren ist dieses Diagnosetelegramm im laufenden Kommunikationsaustausch Bestandteil der Daten, die im Fehlerfall von der IF-PB1 mit ergänzenden Informationen (s. u.) über den Feldbus geschickt werden.

Siehe Abb. 5 unten.

Im Fall der IF-PB1 lautet das Diagnosetelegramm im normalen Betriebszustand (DATAEXCHANGE):

0x80 | 0x0C | 0x00 | 0x02 | 0xA0 | 0x00

Für die Bedeutung der Diagnosedaten siehe die Tabelle unten.

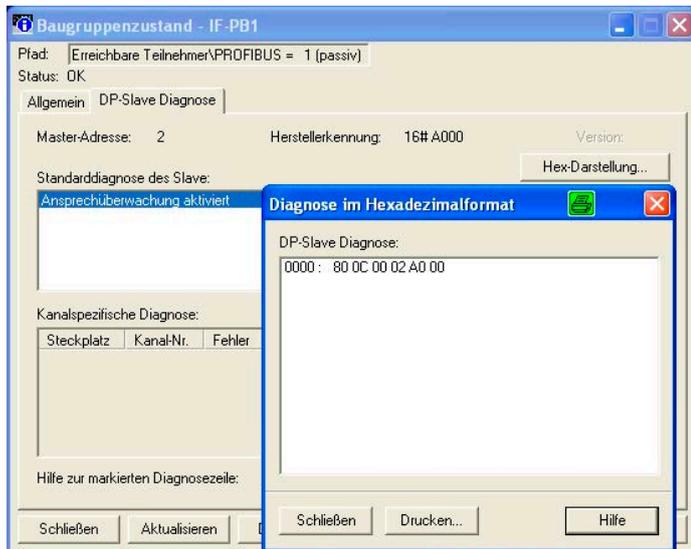


Abbildung 5: Standarddiagnose

### 4.11.2 DP-V1 Alarmmanagement (erweiterte Diagnose)

Über die Standarddiagnose hinaus, wie sie in der Betriebsart DPV0 spezifiziert ist, kann der Anwender erweiterte Diagnosesmeldungen von der IF-PB1 erhalten.

In dem Kommunikationsprinzip DP-V1 ist die sogenannte gerätebezogenen Diagnose aus Alarmblöcken und/oder Statusmeldungen aufgebaut. Zu Aktivierung dieser Syntax muß im Parametertelegramm das Bit DPV1\_Enable auf 1 gesetzt sein.

Die IF-PB1 nutzt die Möglichkeit der Statusmeldung. In der IEC61158 ist sie wie folgt kodiert:

Byte	Bit	
1	0	Station existiert nicht (vom Master gesetzt)
	1	Slave ist nicht bereit für den Datenaustausch
	2	Konfigurationsdaten stimmen nicht überein
	3	Slave hat erweiterte Diagnosedaten
	4	Angeforderte Funktion wird vom Slave nicht unterstützt
	5	Ungültige Antwort vom Slave (vom Master gesetzt)
	6	Falsche Parametrierung
2	7	Slave ist von einem anderen Master parametrierung (vom Master gesetzt)
	0	Slave muß neu parametrierung werden
	1	Statische Diagnose
	2	Fest auf "1" gesetzt
	3	Watchdog aktiv
	4	Freeze Kommando erhalten
	5	Sync Kommando erhalten
3	6	Reserviert
	7	Slave ist deaktiviert (vom Master gesetzt)
4	0...7	Diagnose-Überlauf: Slave hat mehr Diagnose-Informationen als in ein Telegramm passen
5	0...7	Masteradresse nach Parametrierung (0xFF" ohne Parametrierung)
6	0...7	Identnummer High-Byte
7	0...7	Identnummer Low-Byte
8		Header: - Im Header wird die Blocklänge der erweiterten Diagnose inklusive des Headerbytes dargestellt - Bei diesem Modul beträgt der Wert 0x05 (Byte 7...11 = 5 Bytes)
9		Status_Type: Der Wert des Eintrags ist fest und beträgt „0x81“ bei folgenden Bit-Belegungen: Bit 7 = 1: „Status“ Bit 0 = 1: „Statusmeldung“
10		Slot_Nummer: Der Wert der Slot Nummer ist „0x00“
11		Specifier - Ein gemeldeter Fehler wird im Specifier mit der Kennung „0x0“ (Status kommt) eingetragen. - Ein beseitigter Fehler wird im Specifier mit der Kennung „0x02“ (Status geht) eingetragen - Wenn kein Fehler gemeldet wurde, hat der Eintrag im Specifier den Wert „0x00 (keine weitere Unterscheidung)
12		Benutzer-Byte: Unten näher erläutert, enthält für gewöhnlich entweder einen Kommunikationsfehlercode oder einen Alarmcode

Daraus ergibt sich für ein erweitertes Diagnosetelegramm für die IF-PB1:

0x05 | 0x81 | 0x00 | 0x00 | **0x??**

Das letzte Byte gibt die Statusinformation selber an. Es ist entweder ein Kommunikationsfehler (siehe Tabelle in 3.7.2, Seite 13) oder, je nach Anfragetyp, ein Gerätefehler (siehe Tabelle in 3.9).

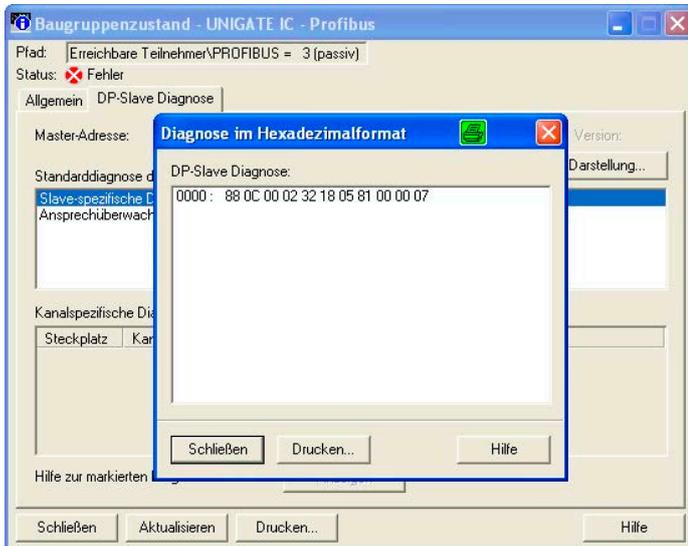


Abbildung 5: Erweiterte Diagnose



	Page
1. General.....	27
1.1 Terms explained .....	27
1.2 Prologue.....	27
1.3 General notes about the communication .....	27
1.4 About the USB driver .....	27
1.5 Structure of the communication .....	27
1.6 Serial transmission settings .....	27
1.7 Translating set values & actual values.....	28
1.8 Message structure USB/RS232 .....	28
1.9 Message structure for CAN.....	29
1.9.1 Former CAN system with device node and RID .....	29
1.9.2 New CAN ID system .....	29
1.9.3 Split messages.....	29
1.10 Message structure IF-G1 .....	30
1.11 Message structure IF-ExB (Ethernet).....	30
1.12 Message structure IF-ExB (USB).....	30
1.13 Timing of messages .....	30
2. Communication in LabView.....	30
2.1 Overview about the Labview VIs.....	30
2.2 Installation .....	30
2.3 Short overview about the VIs .....	30
3. Communication with a device.....	31
3.1 Basic stuff.....	31
3.1.1 Note about the USB driver library .....	31
3.2 Standard procedure .....	31
3.3 Building telegrams.....	31
3.3.1 The start delimiter .....	31
3.3.2 The mask .....	32
3.3.3 Examples .....	32
3.4 The time format.....	32
3.4.1 Time formats for electronic loads .....	33
3.4.2 Time format for power supplies.....	33
3.4.3 Time format for battery chargers.....	33
3.5 Tips.....	34
3.6 Trouble-shooting .....	34
3.7 Error messages.....	34
3.7.1 Explanation about certain error codes .....	34
3.7.2 Communication error list .....	35
3.8 Communication object lists .....	35
3.8.1 Column definition .....	35
3.8.2 Object examples and explanations .....	36
3.8.3 About user profiles .....	36
3.9 Device errors, alarm codes and alarm types.....	37
4. Profibus (IF-PB1).....	38
4.1 General .....	38
4.2 Technical specifications.....	38
4.3 Supported devices .....	38
4.4 Master-Slave communication.....	38
4.4.1 Cyclic.....	38
4.4.2 Acyclic.....	38
4.5 Cabling / Termination .....	38
4.6 Transfer speed and delays.....	39
4.6.1 Influence of cable lengths .....	39
4.7 Operation modes.....	39

4.8	Profibus slave address.....	40
4.9	Communication with the master.....	40
4.9.1	Cyclic field bus data.....	40
4.9.2	Acyclic field bus data.....	40
4.9.3	Using the datagrams.....	41
4.10	Example project with a Siemens PLC.....	42
4.10.1	Integrating the power supply into a Profibus.....	42
4.10.2	GSD file installation.....	42
4.10.3	Module placement.....	42
4.10.4	Addressing the cyclic modules.....	43
4.10.5	Addressing the acyclic module.....	43
4.11	Diagnosis.....	43
4.11.1	Standard diagnosis.....	43
4.11.2	DP-V1 alarm management (extended diagnosis).....	43

 **Attention!**

This document does not deal about the text based communication with SCPI commands via GPIB card IF-G1 or network cards IF-E1/IF-E2! There are separate documents for these interface cards.

## 1. General

### 1.1 Terms explained

**Message:** Chain of bytes with varying length. Is either sent to or received from the device.

**Singlecast:** Query or simple message to a single unit. If devices are linked in a chain, like for instance with CAN, this message is received by all units, but only accepted by the addressed one. It is related to the start delimiter (1<sup>st</sup> byte of a message) and device addressing.

**Broadcast:** Query or simple message to all units. If devices are linked in a chain, like for instance at CAN, this message is received and accepted by all units. Can be used to access devices via ports other than CAN by using a device node of 0. For other interfaces than CAN, broadcast type messages can be used to simplify device identification by leaving the device node (2<sup>nd</sup> byte of a message) always 0, no matter what the device node of the device is set to.

**Object:** with its properties it describes the object address and initiates defined reactions at the target unit. Comparable with a command.

**Message:** CAN specific data packet, like a message without start delimiter and checksum.

**Signal:** Part of a message (term used in Vector software)

### 1.2 Prologue

The communication protocol explained in here with its object orientated message structure is very complex. It is thus recommended to use the ready-made LabView components, if the communication is programmed in LabView. The integration into other environments like Visual Basic, C or .NET requires programming knowledge about the setup and use of hardware interfaces like CAN or USB and the addressing of related drivers.

**Here only the structure of the data packet (the message) is explained and not how it is transmitted correctly.**

### 1.3 General notes about the communication

The firmware of the different types of devices is programmed to consider any circumstances, as far as possible, that may occur when controlling multiple units at once. Thus it is not always possible to perform any action at any time and any state of the device. For example, the data of the function manager of the series PSI 9000 and PSI 8000 (see user manual) are only transmittable in standby state of the unit. Else an error message would be returned, which is pointing the user to the fact that the device is not in standby mode.

### 1.4 About the USB driver

In **Windows XP, Windows 2003, Windows Vista resp. Windows 7** the driver will install two device. One is a USB device called „USB Serial Converter“, the other is a virtual COM port (VCP) called „USB Serial Port“.

The VCP function of the driver is activated by default and will create a new COM port for every USB port that is connected to the PC at least once.

Current LabView VIs (date: 06-2011) can communicate with devices with USB port of either IF-Ux, IF-ExB or IF-PB1 cards only via the virtual COM port. Apart from that, the VIs support the Ethernet port of IF-ExB cards and GPIB (IF-G1 card).

More recent versions of the driver, than the one on the included CD, can be found on the web site of the USB chip manufacturer FTDI at [www.ftdichip.com](http://www.ftdichip.com).

### 1.5 Structure of the communication

The communication with the controlled units is based on these message types:

a) **Message:** an object is sent which shall, for instance, set the output voltage. As long as this action is permitted by the current state of the device, the object is accepted and executed. The device won't send any reply. If it's not permitted it will send a reply in form of an error message.

b) **Query:** a query is sent by using a certain object, for instance „get actual values“, and a reply is expected. If the query is permitted for the current state of the device it is executed and replied. The reply contains the requested data. If not permitted it will send an error message as reply.

### 1.6 Serial transmission settings

Subject: RS232 and USB ports when using VCP (also see section 1.4).

With the serial transmission of one byte following bits are sent:

Start bit + 8 Data bits + Parity bit + Stop bit

The parity is checked for „odd“.

The USB port of IF-Ux, IF-ExB and IF-PB1 is internally working with RS232 characteristics. For these card types it is required to set at least these communication parameters for the particular driver and, when using a RS232 card, on the device, too.

Baud rate RS232 card:	9600Bd ... 57600Bd
Baud rate USB card:	57600Bd
Parity:	odd
Stop bits:	1

## 1.7 Translating set values & actual values

The set values and actual values (see external [object lists](#)) are, with a few exceptions, transmitted as percentage values, whereas 0x6400 corresponds to 100.00%. Hence any real set value has to be translated before and any received actual value after transmission.

If a device has a nominal voltage of 80V and the queried actual value is 0x3200 (0x3200 = 50.00%) then it corresponds to 40V output voltage.

The high byte is the percentage number (0x64 = decimal 100) and the low byte is the decimal places of it.

$$\text{Real value} = \frac{\text{Nom. value} * \text{Per cent value}}{25600}$$

Example: Max. value of the device is 80V, the percentage actual value came in as 0x2454 = 9300. It results in: Actual value = (80 \* 9300) / 25600 = 29.06V

$$\text{Value to send} = \frac{25600 * \text{Real value}}{\text{Nom. value}}$$

Example: the set value for power shall be 500W, the max. value of the device is 640W. With the formula it results in:  
Percentage set value = (25600 \* 500) / 640 = 20000 = 0x4E20.

## 1.8 Message structure USB/RS232

The interface cards IF-Rx (RS232) and the USB ports of IF-Ux, IF-ExB and IF-PB1 are using a slightly different message structure than the CAN cards IF-Cx. Skip to section 1.9 if you're using a CAN card.

The message is structured like this

**SD + DN + OBJ + DATA + CS**

and is built of these bytes:

**Byte 0: SD (start delimiter)**

The start delimiter determines how to handle the message furthermore. Meaning of the bits:

**Bits 3-0:** Data length (Bytes 3-18)

These define the data length - 1 of the data in the message. At a query, the data length of the expected data is given here.

Example: the [object list](#) states a data length of 6 for a certain object, so the lower nibble of SD will be 5.

*Exception: object of type „string“ do not care for the data length as long as it is not higher than given in the object list, so any value between 0 and object length can be given here.*

**Bit 4**

- 0= Message from device to PC
- 1= Message from PC to device

**Bit 5**

- 0= Singlecast, message to a certain device node
- 1= Broadcast message to all device nodes (with CAN), else like singlecast with DN = 0)

**Bits 7+6:** Transmission type

- 00 = Reserved
- 01 = Query data
- 10 = Answer to a query
- 11 = Send data

**Byte 1: DN (device node)**

The device node identifies and addresses devices inside a bus system like CAN or GPIB. Each node number must only be assigned once. This is used to address a particular device. Value range: 1...30, others are invalid. Using CAN, the CAN IDs are calculated from the device node (depends on the device type and firmware, too). See section 1.9 for details. In point-to-point systems like RS232 or USB, this address is only important when using message type „single cast“ (see Byte 0). When using broadcast, the DN can always be 0.

**Byte 2: OBJ**

The communication objects for a device are addressed by this byte. In the communication [object lists](#), the objects and their function(s) are explained in detail.

**Byte 3 - 18: Data field**

The data field can be up to 16 bytes long, hence the length of the message varies. If a query is sent (PC → device) there will be no data (=data field empty) and the checksum of the message (see below) follows directly after byte 2. Only if an answer (device → PC) is sent, an expected one or an error, there will be data of a specific length.

**Last 2 bytes: CS (checksum)**

The checksum is always located at the end of the message. It is built by the simple addition of all preceding bytes of the message and attached to the end. It is two bytes long (16 bit value). The high byte is placed before the low byte.

**Example of a message:**

Object no. 71 (query actual values) shall be sent to a device with device node 1. The message has to be like this in hexadecimal values:

**55 01 47 00 9D**

And the answer could look like this:

**85 01 47 64 00 1E 00 50 00 01 9F**

(this translates to 80V, 30A and 2400W actual values for a 80V, 100A and 3000W power supply, like the PSI 9080-100)

Also see section 1.7 for the conversion of set values and actual values.

For more examples see sections 3.4 and 3.8.2.

## 1.9 Message structure for CAN

### Attention!

The series PS 8000 (from firmware 6.01), PSI 8000 (from firmware 3.14) and EL 3000/EL 9000 (from firmware 5.01) feature two CAN ID systems, as described below. Other series only support the system as described in 1.9.1.

The interface cards IF-C1 and IF-C2 support the CAN V2.0a standard. The extended address format is not used.

Refer to the external user guides of your device and the interface card for the CAN specific communication settings.

### 1.9.1 Former CAN system with device node and RID

*Note: this CAN system is implemented into every device series that supports a CAN interface card type IF-Cx, except in PS 8000 models with firmware 6.01 or up.*

The CAN driver chip requires the **identifier**, up to 8 data bytes and the **data length** for a transmission. The **identifier** is 11 bits long (CAN 2.0a) and specified by the **device node**, the relocatable identifier segment (**RID**) and the type of the message. For every unit there are two identifiers:

[RID\*64 + device node \* 2] (**1<sup>st</sup> identifier**) and

[RID\*64 + device node \* 2 + 1] (**2<sup>nd</sup> identifier**),

whereas the 1<sup>st</sup> identifier is used to send messages to the device to set something and the 2<sup>nd</sup> one to send messages in order to query something and to receive replies.

A message can contain a maximum of 8 bytes. The first byte is the number of the communication object. After this you can put or receive up to 7 data bytes (see communication [object lists](#)).

In order to send an object with a 16 bytes long data field it is thus required to send at least three message and the data field has to be split up over those three messages. See section 1.9.3 for more.

Two examples:

a) The device shall be set to remote control mode. This is required to control the device by a status command or to set values. The **device node** was set to 15 and the **RID** to 3. The message is of „send only“ type. The send identifier calculates as  $3 * 64 + 15 * 2 = 222_{10}$  or  $0xDE$  with the above formula. According to the [object lists](#) we use object 54 (hex:  $0x36$ ) with the data bytes  $0x10$  (mask) and  $0x10$  (set remote). The resulting data length is 3, so the data to set up the CAN controller is like this (hex values):

```
ID DL DATA
DE 03 36 10 10
```

b) In case you don't want to set the state but query it, the identifier  $0xDF$  is used (2<sup>nd</sup> identifier) and because it is a query, the object number is sufficient as data. The CAN data are thus like this (hex values):

```
DF 01 36
```

and the answer should be like this:

```
DF 01 36 10 10
```

### 1.9.2 New CAN ID system

#### Note

*Note: the new CAN ID system can be made available by a firmware update for the series PS 8000, PSI 8000, EL 3000 and EL 9000.*

The new CAN ID system uses 3 basic IDs plus a broadcast ID (per unit) for communication. It is Vector software compatible, such as CANalyzer or CANoe. Vector databases in format \*.dbc are available upon request or on the CD that is included with the CAN interface card. There will also be ready-to-use configurations to be opened in those softwares for demonstration purposes.

In the device setup and if CAN interface IF-C1 or IF-C2 is equipped, the user adjusts a base ID. Then the device uses three CAN IDs (base ID, base ID + 1 and base ID + 2). Those IDs have to be unique in the bus and thus the base ID adjustment is done in steps of 4. If multiple units of the same type are used within a bus, they all need to have a different base IDs.

Regarding the broadcast ID, this is vice versa. Here it is intended for multiple units to have the same broadcast ID adjusted. The purpose is to send the same message to x units on the bus, in order to set, for example, the same current set value at the same time. The broadcast ID is adjusted separately from the base ID in steps of 1 and has to be unique, too. It means, it has to be different from the other CAN IDs of the device.

The three standard CAN IDs are assigned as:

Base ID = Used by the PC to only send messages that set a value or condition and don't create an answer message

Base ID + 1 = Used by the PC to query anything from the device by just sending the queried object (see documentation about the available objects (i.e. commands))

Base ID + 2 = Used by the remote device to send answers to queries or to send error messages

The broadcast ID is assigned as:

Broad ID = Used by the PC to only send messages to multiple units, in order to set a value or condition and don't create an answer message

### 1.9.3 Split messages

A split message is a message, which is split into multiple submessages (only concerns objects in „string“ format). An extra identifier is inserted here, after the object number (=object address). The extra identifier of the first message is  $0xFF$ , for the second message it is  $0xFE$  and  $0xFD$  for the third one. The order of these messages is not specified. The message has to be composed again later from these messages. When using the gateway function the split telegrams are not composed by the gateway. This has to be done by a superior control unit.

### 1.10 Message structure IF-G1

The message structure for the text based communication via a GPIB card is described in external user guides.

Link: [SCPI command list for power supplies](#)

Link: [SCPI command list for electronic loads](#)

### 1.11 Message structure IF-ExB (Ethernet)

The message structure for the text based communication via the Ethernet port of an IF-ExB is described in external user guides.

Link: [SCPI command list for power supplies](#)

Link: [SCPI command list for electronic loads](#)

### 1.12 Message structure IF-ExB (USB)

See sections 1.6 and 1.8. Communication in LabView

### 1.13 Timing of messages

After every query the device typically needs between 5ms and maximum 50ms for the answer. Basically you are allowed to send queries directly after another. A time of 100ms between two messages is recommended in order to not slow down the device's operation by too much communication.

Commands, which set something (output/input condition, set values) also have a variable delay (5-50ms) until they are processed and transferred to the power stage's DC output/input, with an additional time for regulation.

#### Gateway (only old PSI 9000 series until 2012)

When using the gateway function you need to consider the time that will be consumed by transferring the message from one bus system to the other. The answer may be delayed up to 200ms here.

After receiving an error message over this gateway you should consider to wait at least 100ms until the next transmission.

#### Broadcast with CAN

After every broadcast query all bus sharing units can only answer consecutively. Depending on the bus system, the baud rate and the number of units, as well as the extra bus traffic, the answers can be delayed more or less. The time is not specifiable and can only be estimated by the formula *bus sharing units \* response time at singlecast*. In most cases the response time will be shorter.

## 2. Communication in LabView

### 2.1 Overview about the Labview VIs

For an easy integration of multiple and even different devices into existing LabView applications we provide a set of Labview VIs.

Those virtual instruments (VI) enable a simple implementation into and the programming of an application without the need for the user to learn about the lower levels of communication.

In order to use the functionality of these VIs it is required to use and run the software development tool LabView from the company National Instruments. The LabView VIs require version 7.0 or higher.

Following minimum system requirements have to be considered:

- Pentium 3 CPU with 256 MB memory
- Windows operating system (Win98 and WinXP)

Recent versions of these VIs can be downloaded from our website.

### 2.2 Installation

To install and use the VIs in LabView in your environment, please read the file „installation\_english.pdf“ on the included CD for instructions.

After the installation you can find the VIs in the context menu of the LabView IDE in „Instrument I/O -> Instrument drivers -> IF-XX“.

Some VIs are only for a specific series and will only work with these. They are located in subfolder of the installation. The other VIs are for common use. Usage and functionality are described in the user guide of the VIs. You can access it the usual way via the LabView context help (Ctrl+H) or open the CHM directly. Depending on the Windows version it can be necessary to copy the help file to a local hard drive in order to read it correctly.

#### Note

*Please read the LabView VIs help file on the included CD in order to get an overview and a clue about the handling.*

### 2.3 Short overview about the VIs

Date: 03/2015

After unpacking the VI set, several subfolders become available.

The common VIs in folder „Common“:

- **Device\_close.vi** - closes the communication with a specific device. Before exiting your application, any open communication port should be closed, if device communication is not used any further.
- **Device\_scan.vi** - scans certain hardware ports (RS232, GPIB, Ethernet, USB(VCP)) for compatible devices and returns the number of found devices as well as a list of device information, which is required for the other VIs.

- **Device\_select.vi** - is used to select and open a specific device from the list of devices that is returned by „Device\_scan.vi“. Multiple devices can be selected and used in parallel.

Series specific VIs in the folders:

- „\BCI8 Series“ - for models of series BCI 800 R
- „\PS2 Series“ - for models of series PS 2000 B
- „\PS5 Series“ - for models of series PS 5000 A
- „\PS8 Series“ - for models of series PS 8000
- „\PSI5 Series“ - for models of series PSI 5000 A
- „\PSI8 Series“ - for models of series PSI 8000 and PSI 800 R
- „\PSI9 Series“ - for models of series PSI 9000 (old and new)
- „\EL Series“ - for models of series EL 3000 and EL 9000
- „\ELR Series“ - for models of series ELR 9000

The examples in folder „**Examples**“ are intended to demonstrate the use and placement sequence of the common and specific VIs. The series specific VIs in the example VI can be replaced by other specific VIs in order to test a different device type.



### Attention!

**Whenever a series specific VI is replaced by another series specific VI, the enum constant on input „command list“ must be deleted and created again!**

## 3. Communication with a device

### 3.1 Basic stuff

The following sections deal about the composition of the communication telegrams, the dependency of the communication from the state of a device and the problems related to that topic, without explaining how to use the USB driver when using the USB port in low level or how to correctly send a CAN message. This has to be learned and done by the user.

#### 3.1.1 Note about the USB driver library

When using the USB card IF-Ux or the USB port of the Ethernet card IF-E1B or Profibus card IF-PB1, a USB driver has to be installed. After this, the user can choose to communicate via the USB driver on low-level access or via a virtual COM port (VCP), whereas accessing the VCP is much simpler.

On the included CD in the folder \manuals\other\ftdi, there is a PDF which describes the functions of the USB driver DLL in detail. In general it applies, that a device (in this case the USB hardware) has to be opened first (FT\_Open or similar), then configured (FT\_SetBaudRate, FT\_SetDataCharacteristics etc.) and then it can be written (FT\_Write) or read (FT\_GetQueueStatus, FT\_Read). As soon as the device is not used anymore it is advised to close it (FT\_Close), while it is advised not to open and close it for every read-write cycle. Configuration of the USB hardware needs to be done only once as long as it is powered. The functions FT\_Write and FT\_Read serve to transport the actual message bytes of the object orientated communication described in the following sections.

### 3.2 Standard procedure

The programming of the various devices, in which the interface cards are used, always follows the same scheme. It only differs in number and functionality of the communication objects that are supported by a specific device series.

Generally applies:

- Monitoring, i.e. only querying actual values and status, is always possible. The devices don't require the remote mode.
- Setting of status and set values is considered as controlling and requires the activation of the remote mode (remote in this case means that the device is remotely controlled via a digital interface card)
- The remote mode can be blocked by certain circumstances. For instance, the explicit local operation (only PSI 9000) or a different mode the device is in and which does not allow remote control. For further details refer to the user manual of your device.

In order to start controlling a device, for example by sending a set value, you need to

#### 1. activate the remote mode (object 54)

#### 2. send the set value

and, if not already done,

#### 3. set the output/input to on

The remote mode should be left again, if not used any further. As long as it is active, the device can not be operated manually or only restrictedly. The mode is indicated on the front.

### 3.3 Building telegrams

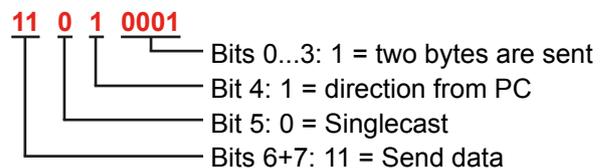
#### 3.3.1 The start delimiter



### Note

*The start delimiter is not used with CAN*

According to the message format (also see 1.8), the first byte is the start delimiter (SD), which depends on the type and direction of the message. For this example the SD will be 0xD1, and look like this in single bits :



Alternatively to the bitwise assembly, this can be simplified by using hex values. Starting from bits 6 + 7 we get:

**SD = Message type + Cast type + Direction + Length**

whereas the message type is either

**0xC0** Send data or  
**0x40** Query data

and the cast type is either

**0x00** Singlecast (device node is checked) or  
**0x20** Broadcast (device node is ignored)

and the direction is either

- 0x10** from PC to the device or
- 0x00** from device to the PC

and the data length - 1 can be

**0x00...0x0F** up to 16 bytes of data (at CAN see „1.9.3 Split messages“)

**Note**

*Always note, that the data length is defined as number of bytes to send -1!!!*

**3.3.2 The mask**

Some objects require a mask byte, which is sent before the control byte. The required mask byte is given in column 7 of the [object lists](#). The bits of the control byte have various functions, so it has to be determined which bits have to be changed. This is determined by the mask byte with a 1 for every bit that shall be changed by the control byte. Example: bit 4 shall be changed to 1. Then the control byte would be 0x10 and the mask would be 0x10, according to the position value of bit 4. In case there is only one mask given, like 0x0F, it has to be always used as 0x0F. Then all masked bits are changed at once, for example like with object 56 (function manager of PSI 8000 and PSI 9000). A possible control byte could then be 0x02, i.e. command STEP.

**Attention!**

**In order to avoid logical collisions of functions that are assigned to the bits, it is advised to only set 1 bit per message, though setting of multiple bits at once is possible.**

**3.3.3 Examples**

**Example 1: Switch device to remote mode**

The address (node) of the contacted device is set to 5, the object to use is 54 (hexadecimal: 0x36), the mask for the remote mode (also see [object lists](#)) is 0x10 and the control byte has to be 0x10 (remote = on). Then we get this message of hex bytes (prefixes removed for a better view):

**D1 05 36 10 10 01 2C**

In order to reverse this command, i.e. deactivation of the remote mode, you need to send

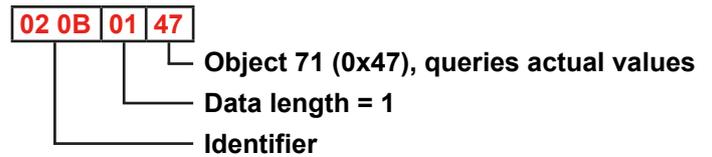
**D1 05 36 10 00 01 1C**

The mask stays the same, only the control byte changes.

**Example 2: Querying actual values via CAN card**

Using CAN, the start delimiter and the extra checksum are not used. So we only need the object (according to the [object lists](#) it is 71 (hex: 0x47) for actual values), the identifier and the length of the bytes to send. In a CAN message, the object is included in the data length, so this message would have a data length of 1, because we only send the object number to query the actual values. Using the old CAN system (see sections 1.9.1 and 1.9.2) we use device node 5 and RID=8. According to the formula from section 1.7, the identifier calculates as  $8 * 64 + 5 * 2 + 1 = 523$  (hex = 0x20B). The +1 is because it is a message of type „query“.

We now send one byte to CAN ID 0x20B. The CAN message could look like this (depending on how it displayed on the software GUI that you are using):



**Attention!**

**The example above DOES NOT show the data content of the CAN message that is actually sent over the CAN bus. Identifier and data length are not part of the data content.**

An answer to this query could look like this:

**02 0B | 06 | 64 00 0A 00 42 AA**

Same identifier, data length is 6, because three actual value of 16 bits size each are sent. The actual values are transmitted as percentage values and need to be translated to real values. See section „1.7 Translating set values & actual values“ for details. For an EL 9080-200, the actual values would translate to 100% for voltage (=80V), 10% for current (=20A) and 66,7% for power (=1600W).

**Note**

*The nominal values for power, current and voltage can be read from the device via the corresponding objects and used to translate the actual values to real values and vice versa.*

**3.4 The time format**

The time format represents times from 1µs to 100h by a 16 bit value. Such time stamps are checked by the device for being correct. Values that are too high or too low are not accepted and will return an error message. The upper 4 bits are used as a mask to determine the time range, the rest of the bits represent the time value.

The time format is used to write (i.e. set) or read time values.

It applies to any device that features a function related to time, as long as this time value is settable. The resolution of the time ranges in the table below does not necessarily match the resolution of the particular device. In this case, the values are rounded down. An example: a time value of 0x23E7 is sent. This represents  $999 * 1\mu s = 999\mu s$ . The manually adjustable time value of the device in this time range is but 0.95ms or 1ms. The 999µs are rounded down to 950µs. Hence there will be 0x23B6 returned (=950) when read back, instead of the sent 0x23E7.

**Note**

*Not all devices use all of the masks in the table below.*

**Note**

*Not all time values that can be read from a device are necessarily built by the time format specification. See [object lists](#).*

**3.4.1 Time formats for electronic loads**

For electronic loads and the **rise time (object 92)** applies, according to the main table below:

Time range		Step width of device	Mask	Value range**	
from	to			from	to
30µs	99µs	1µs	0x2000	0x201E	0x2063
0.10ms	0.99ms	10µs	0x2000	0x2064	0x23DE
1.0ms	9.9ms	100µs	0x3000	0x3064	0x33DE
10ms	99ms	1ms	0x6000	0x6064	0x63DE
100ms	200ms	1ms	0x7000	0x7064	0x70C8

\*\* Values differing from the step width are rounded

For electronic loads and the **pulse width (objects 90 and 91)** applies, according to the big table below:

Time range		Step width of device	Mask	Value range**	
from	to			from	to
0.05ms	0.95ms	50µs	0x2000	0x2032	0x23B6
1.00ms	9.95ms	50µs	0x3000	0x3064	0x33E3
10ms	99.9ms	100µs	0x6000	0x6064	0x63E7
100ms	999ms	1ms	0x7000	0x7064	0x73E7
1.00s	9.99s	10ms	0x4000	0x4064	0x43E7
10.0s	100s	100ms	0x9000	0x9064	0x93E8

\*\* Values differing from the step width are rounded

For electronic loads and the **elapsed time of battery test (object 64)** applies, according to the big table above:

Time range		Step width of device	Mask	Value range**	
from	to			from	to
1s	3599s	1s	0x8000	0x8001	0x8E0F
1h	99h:59m	1m*	0xC000	0xC03C	0xD76F

\* Above 1h, time can only read as HH:MM

\*\* Values differing from the step width are rounded

**Example 1:** the rise time for an electronic load shall be set to 75ms. The step width of load's the time range where the 75ms belong to, is 1ms. So we need to use the 0x6000 time range. Time range resolution here is 0.1ms, so it results in a time value of 750 (75ms : 0.1ms). This translates to 0x2EE. Together with the mask you get a value of 0x62EE as time value for the rise time (object 92).

**Note**

LabView users need to provide the time in a different way. See VI documentation.

**Example 2:** the time value of the battery test has been read and shall now be translated to the normal time format. The overall resolution of the battery test time is 1s. Since the time ranges allow 1s resolution only up to 1h, the time above 1h is given in minutes and hours. A value of, for example, 0x8743 would translate into 1859s or 30m59s, whereas a value of 0xC532 would translate to 1330m or 22h10m. The seconds are omitted in this time range, so you would always read out the same time value during 1 minute.

**Example 3:** Setting the pulse width for A (object 90) to 5s. According to the 2<sup>nd</sup> table from above the time range mask is 0x4000. In combination with the resolution of 10ms for this time range, a value of 500 (5s : 0.01s) results, in hex 0x1F4. The total resulting time value then would be 0x41F4.

**3.4.2 Time format for power supplies**

Applies to: PSI 8000 / PSI 9000 (old series until 2012)

Time values for sequences of the PSI 8000 / PSI 9000 function manager are time formats in 2ms grid or a multiple of it. Other time values also use these time formats, like with objects 39-43 and 47.

Time range		Step width of device	Mask	Value range	
from	to			from	to
0.002s	9.998s	2ms	0x0000	0x0001	0x1387
10.00s	59.99s	10ms	0x4000	0x43E8	0x576F
1:00m	59:59m	1s	0x8000	0x803C	0x8E0F
1:00h	99:59h	1m	0xC000	0xC03C	0xD733

**3.4.3 Time format for battery chargers**

Applies to: BCI 800 R

Battery chargers of BCI 800 R series don't make use of the 2-byte time format when counting the charging time in seconds, minutes, hours and days in single bytes. See BCI 800 R object list.

For other parameters like the max. time of a charging phase, time format 0xC000 is used:

Time range		Step width of device	Mask	Value range	
from	to			from	to
0:00h	99:59h	1m	0xC000	0xC000	0xD76F

Mask *	Time value (bits 11..0)				Resolution	Resulting time range
	min.(dec)	min.(hex)	max.(dec)	max.(hex)		
0x2000 <sup>(1)</sup>	0	0x00	999	0x3E7	1µs	0 ... 0,999ms
0x3000 <sup>(2)</sup>	100	0x64	999	0x3E7	10µs	1ms ... 9,99ms
0x6000 <sup>(1)</sup>	100	0x64	999	0x3E7	100µs	10ms ... 99,9ms
0x7000 <sup>(2)</sup>	100	0x64	999	0x3E7	1ms	100ms ... 999ms
0x0000 <sup>(1)</sup>	0	0x00	4999	0x1387	2ms	0 ... 9,998s
0x4000 <sup>(1)</sup>	100	0x64	5999	0x176F	10ms	1,00s ... 59,99s
0x8000 <sup>(1)</sup>	1	0x01	3599	0xE0F	1s	1s ... 59min:59s
0x9000 <sup>(2)</sup>	100	0x64	1000	0x3E8	100ms	10,0s ... 100,0s
0xC000 <sup>(1)</sup>	0	0x00	5999	0x176F	1m	01:00h ... 99:59h

Table: Time format

\* If the mask is used to translate time values into real time, either bits 15...13 or 15..12 are relevant, depending on the used time range

### 3.5 Tips

#### I. Detecting a device node (not with GPIB)

If you want to, for example, control a device via USB and you don't know the device's node, you could for example use the broadcast node 0 and query the device class. The device or the devices will answer with its/their own device node, that has/have been set at the device(s). The device node(s) can be furthermore used to control and distinguish the devices.

Using an USB or RS232 interface is a point-to-point connection and here you can generally use broadcast message type with device node 0 (see section „3.3.1 The start delimiter“).

#### II. Remote and standby

The object 54 is used to either activate/deactivate the remote control or switch the input/output of a device. The object can be used to activate both at once, but it is strongly not recommended to do so, because setting the input/output requires the remote control operation already being active and else would generate an error message. You should rather activate remote control first with the corresponding bit set in the control byte and then control the input/output by sending object 54 a second time with a different control byte. When deactivating remote control it simply goes vice versa.

It is also useful to read back the state of the device with object 70, in order to check if object 54 has been set correctly.

### 3.6 Trouble-shooting

**Problem:** The device does not react or respond to commands

#### Possible causes with USB

- The USB card requires a driver. Check if the driver is installed correctly and if you can find the card as „USB Serial Converter“ in the Windows device manager in the section of „USB controllers“.
- The wrong device node (=address) is used to communicate with the device.

#### Possible causes with RS232

- You are not using a 1:1 cable for the RS232 card.
- The wrong device node (=address) is used to communicate with the device.
- Device and PC are configured for different baudrates etc.
- The communication cable is too long for the configured baudrate (also see section „2. Technical specifications“ of your interface card user guide).

#### Possible causes with GPIB

- If multiple device are connected to a IEEE bus, one or more device addresses might be double.
- A wrong syntax is used. For example, an electronic load does not react to the command OUDP, because it features an input. Or the command was not valid for the type of device you tried to contact.

#### Possible causes with CAN:

- The wrong CAN ID is used.
- Wrong baudrate set
- Wrong sample point selected (only at PSI 9000, see user guide)
- The device is located at the end of the bus and is not terminated

**Problem:** Multiple queries were sent, but not all of them were answered

**Cause:** The queries have been sent subsequently too fast. Depending on the communication type and speed and the execution time of the device, you need to include a certain latency between two queries.

Rule of thumb: Latency = Transmission time + Execution time

The execution time lies at typ. 5-20ms, depending if there only was a query or if something has to be set. The transmission time can be calculated from the baudrate and number of bits that are sent.

**Problem:** Set values and status are not set

#### Possible causes

- The contacted device is not in remote control mode or can't currently be set to this mode, because it might not be allowed in this very moment or any other condition for setting the device into remote control is not fulfilled
- The sent values are wrong (too high, too low) or the standard value range (0...0x6400 for voltage, current etc.) is additionally limited by limit values (only with PSI 9000 / PSI 8000). An error message is sent in this case.

### 3.7 Error messages

The table in section 3.7.2 lists all possible error codes that could be returned by the device with an error message. The code is used to point the user to the origin of the error. Some errors are caused by erroneous messages (i.e. wrong checksum etc.), others might come from the device, denying a command in dependency of the current condition.

Error messages are in message format, i.e. they are composed of a start delimiter (except with CAN), object number (to identify an error, **0xFF** is used as object number), error code in data field and checksum (except with CAN).

Example: in case you want to set the voltage with object 50 and the device is not in remote control, you would receive the error message **C0 07 FF 09 01 CF** from a device with device node 7.

#### 3.7.1 Explanation about certain error codes

**Code 0x7:** the object number used in the message is unknown to the device. This is because not all device types use the same object numbers. Use the object list for you device to compare.

**Code 0x8:** the length of the data field in the message is defined in the [object lists](#). This error code will be returned if a set value, which is always 2 bytes because of type „int“, should have been sent but the data field only contained one byte. Even if the start delimiter contained the correct message length. This is a protection against setting wrong values.

**Code 0x9:** an object has been sent in order to set a value, but the device is not in remote control mode. In this condition you only have read permission, but no write permission. You need to set the device to remote control mode first.

**Code 0xE:** Strings have to be transferred differently when using CAN. If the string length is greater than 8 characters, you have to use split messages that are designated with the string start tokens 0xFF, 0xFE etc. Also see „1.9.3 Split messages“.

**Codes 0x30/0x31:** these are related to set values. All set values have an upper and a lower limit, which are defineable at the power supplies. The default upper limit for e.g. the current set values is 0x6400 (=100%) and the lower limit is 0. Limits also apply to time values.

**Code 0x32:** a time value using the wrong time range has been sent. The upper or lower limits are not exceeded by the value, but it still causes this error.

**Codes 0x36:** Conditions for the access to these data are not fulfilled. See [object lists](#) about the access conditions in its column 4.

### 3.7.2 Communication error list

Error code		
Hex.	Dec.	Description
01	1	RS232: Parity error
02	2	RS232: Frame Error (Startbit or Stopbit incorrect)
03	3	Check sum incorrect
04	4	Start delimiter incorrect
05	5	CAN: max. nodes exceeded
06	6	Device node wrong / no gateway present
07	7	Object not defined
08	8	Object length incorrect
09	9	Read/Write permissions violated, no access
0A	10	Time between two bytes too long / Number of bytes in message wrong
0C	12	CAN: Split message aborted
0F	15	Device is in "local" mode or analogue remote control
10	16	CAN driver chip: Stuffing error
11	17	CAN driver chip: CRC sum error
12	18	CAN driver chip: Form error
13	19	CAN: expected data length incorrect
14	20	CAN driver chip: Buffer full
20	32	Gateway: CAN Stuffing error
21	33	Gateway: CAN CRC check error
22	34	Gateway: CAN form error
30	48	Upper limit of object exceeded
31	49	Lower limit of object exceeded
32	50	Time definition not observed
33	51	Access to menu parameter only in standby
36	54	Access to function manager / function data denied
38	56	Access to object not possible

Legend

	Communication error
	User error
	Internal error

Table: communication errors

## 3.8 Communication object lists

### Note

The object list for your device is located in an external PDF file and should be distributed along with this document.

Link: [Object list series PSI 8000 T / DT / 2U / 3U](#)

Link: [Object list series PSI 9000](#)

Link: [Object list series PSI 800 R](#)

Link: [Object list series BCI 800 R](#)

Link: [Object list series EL 3000 und EL 9000](#)

Link: [Object list series PS 8000 T / DT / 2U / 3U](#)

The object lists are a reference for programming custom applications which shall control our device remotely and apart from LabView, where ready-to-use LabView VIs are available. LabView users wanting to reconstruct the binary message communication also use these lists as reference.

### 3.8.1 Column definition

The **1st column** contains the object number (=command) as a decimal value. This number has to be assigned to the byte **OBJ** (see 1.8) in the message.

The **3rd column** defines whether the object is read only (ro), i.e. it can only be queried from the device, or if it can also be written (w, r/w). Reading from a device is generally possible, also called monitoring. Setting values or status requires enabling of the remote control mode (if device is not in „local mode“ or similar). Also see 3.2.

The **4th column** defines a special access condition of an object. The execution of these objects additionally depends on one of the below conditions. If the condition is not given, the object is not executed and the device will return an error message which contains an error code. Meaning of the numbers:

- 1** = The output/input of the device has to be switched off  
(The object is only accepted and executed by the device if the power output/input is in standby mode)
- 2** = Option „Internal resistance“ has to be unlocked  
(The object is only accepted and executed by the device if the option „Internal resistance“ is unlocked)
- 3** = Transfer of the function data has to be enabled  
(The object is only accepted and executed by the device if it has been instructed before by a different object to receive and set function data)
- 4** = Function manager activated  
(The object is only accepted and executed by the device if the function manager has already been activated manually in the device menu or by a different object)
- 5** = Function manager not activated  
(The object is only accepted and executed by the device if the function manager is not active)

### Attention!

It is always required to set the device into remote control mode **BEFORE** sending objects that will change any value or status on the device.

The **5th column** defines the type of the data in the **data field** of the message. Commonly known data types are used.

The **6th column** defines the data length of the **object's data**. For objects with data type „string“ this byte defines the maximum length of the string. The string has to be terminated with an EOL (end of line, =0 ) or it ends after the given number of bytes. Strings are transmitted in up to three split telegrams if CAN is used. See also „1.9 Message structure for CAN“.

The **7th column** is used to mask out data of type „char“. The mask defines which bits may be set or unset. Also see section „3.3.2 The mask“.

**Columns 8 & 9** explain details about the **data field** content.

Some objects use a two-byte time format, which is explained in section „3.4 The time format“.

### 3.8.2 Object examples and explanations

☞ A description of the object list columns can be found in section 3.8.1.

☞ All numbers are in decimal, if not marked as hexadecimal by a leading 0x.

#### I. Function manager (objects 56, 58, 73-75, 78, 90-146)

When using the function manager, the order of using the objects becomes very important. Since setup and control of the function manager is complex it is not handled here, but in an external application note (AN001), which you can also find on the included CD in the folder „\manuals\application notes“ or on our web site.

#### II. Object 54

The primary function of this object is to switch the device to remote control operation or to switch the power input/output on or off. The mask has to be given according which bit is going to be changed.

Example: Activate remote control -> Bit 4 ->Value of bit 4 = 0x10 -> Mask 0x10 -> control byte also 0x10. The object 0x32 will then contain the data 0x1010. Deactivate remote control, the same way: Mask 0x10 -> Control byte 0x00 -> data 0x1000.



#### Attention!

**In order to avoid logical collisions of functions that are assigned to the bits, it is advised to only change 1 bit per message, though multiple bits can be changed at once!**

When reading objects that require a mask, the main mask is also returned, but is only of informative value.

#### III. Object 56

The bits are here only allowed to be set individually. Else the actions are not executed properly.

#### IV. Object 73

The time stamp is only available when using the function manager, else it will be 0. It represents a counter's value of the elapsed time in 2ms steps. Because it is an integer value, the counter will restart at 0 after 65536 x 2ms.

#### V. Object 77

Reading the alarm buffer will clear it (EL 3000 or PS 8000). Other device series, like PSI 9000, PSI 8000, PSI 800 R and BCI 800 R require initiation to flush the buffer, by setting a bit in object 54. Because it only stores 3 events, i.e. alarms, the first three events are held and any subsequent event will always overwrite the most recent one in position „Last alarm type“.

Example: the object returned 0x0120 in the first two bytes in the data field (last recorded alarm) -> error type 0x01 means, that the error still persists and error code 0x20 says that it is an overtemperature error in the upper power stage of a multi-stage PSI 9000 power supply (see table in section 3.9),.

#### VI. Objects 39-47

These objects are related to the section 7.6 of the PSI 9000 or PSI 8000 instruction manuals. Events that are triggered by the supervised values will generate an error of type alarm, warning or notification (see PSI 9000 user manual for definition) in the alarm buffer, depending on the configuration with objects 44-46.

The time to give here is a latency for the event trigger. Valid time range: 2ms...100h. For time format see section 3.4.

#### VII. Objects 21-29

These objects are used to load preset lists into the device, just like you can enter and modify directly at the device. But there is no further control possible, like selecting a preset or switching between presets in order to generate voltage jumps. In order to perform such actions, other objects have to be used.

### 3.8.3 About user profiles

The device series PSI 8000 and PSI 9000 feature more than one user profile. Up to 4 can be stored and recalled. Every profile contains a set of settings and also a set of preset lists. It means, the user has to be aware which profile is currently selected when writing preset lists into the device or reading them.

The settings and values that belong to a profile, represented by the objects, are coloured in the **object lists** (if the list is for a device series that features multiple user profiles).

### 3.9 Device errors, alarm codes and alarm types

Fehlercode Error code	Kürzel Abbr.	Fehlertext oder -beschreibung / Error text or description
0		Kein Fehler / No error
1	OV	Überspannung am Ausgang (Eingang) / Overvoltage at output (input)
2	OT	Übertemperatur im Gerät / Overtemperature inside the device
3	SYS	Systemfehler / System error
4	U>	Obere Spannungsgrenze überschritten / Upper voltage threshold exceeded
5	U<	Untere Spannungsgrenze unterschritten / Lower voltage threshold exceeded
6	I>	Obere Stromgrenze überschritten / Upper current threshold exceeded
7	I<	Untere Stromgrenze unterschritten / Lower current threshold exceeded
8	SIO2	System Link Mode: Kommunikation gestört / Communication disturbed
9	MS1	System Link Mode: Ein oder mehrere Gerät sind "offline" / One or more units are offline
10	S-OV	System Link Mode: Slave meldet Überspannung / Slave is reporting an overvoltage
11	S-OT	System Link Mode: Slave meldet Übertemperatur / Slave is reporting overtemperature
12	S-PH	System Link Mode: Slave meldet Netzfehler / Slave is reporting mains voltage error
13	S-PD	System Link Mode: Slave ist in Leistungsbegrenzung / Slave reduces max output power
14	S-?	System Link Mode: Slave antwortet nicht / Slave does not answer
17	F01	Interner Fehler / Internal error
19	F03	Interner Fehler / Internal error
20	CAN	CAN: Kommunikation gestört / Communication disturbed
21	FCT	Funktionsmanager: Funktion konnte nicht gesetzt werden / Function manager: function could not be set
22	UDU	Überwachung Sprungantwort: Anstieg U / Step response supervision: U rise
23	UDD	Überwachung Sprungantwort: Abfall U / Step response supervision: U fall
24	IDU	Überwachung Sprungantwort: Anstieg I / Step response supervision: I rise
25	IDD	Überwachung Sprungantwort: Abfall I / Step response supervision: I fall
26	PDU	Überwachung Sprungantwort: Anstieg P / Step response supervision: P rise
27	PDD	Überwachung Sprungantwort: Abfall P / Step response supervision: P fall
28	PH1	Phasenausfall oberes Leistungsteil / Phase loss of upper power stage
29	PH2	Phasenausfall unteres bzw. mittleres Leistungsteil / Phase loss of lower resp. middle power stage
30	PH3	Phasenausfall unteres Leistungsteil / Phase loss of lower power stage
31	OT1	Übertemperatur oberes Leistungsteil / Overtemperature of upper power stage
32	OT2	Übertemperatur unteres bzw. mittleres Leistungsteil / Overtemperature of lower resp. middle power stage
33	OT3	Übertemperatur unteres Leistungsteil / Overtemperature of lower power stage
34	CC	nur UIR Betrieb: Betrieb des Stromreglers nicht erlaubt
35	CP	nur UIR Betrieb: Betrieb des Leistungsreglers nicht erlaubt
36	-	Bat.temp. out of range (Batterietemperatur zu hoch)
37	-	Bat.temp. out of range (Batterietemperatur zu niedrig)
38	-	Bat.voltage out of range (Batteriespannung zu hoch)
39	-	Battery deeply discharged ( Batteriespannung zu niedrig bzw. tiefentladen)
40	-	Cell fault in battery (Zellenschluß)
41	-	Temp sensor fault (Temperaturfühler fehlt oder defekt)
42	-	Reverse polarity (Batterie verpolt)
43	-	Battery not connected (keine Batterie angeschlossen)

  nur bei Mehrphasengeräten / only at multi-phase models

  Gilt für BCI 800 R Serie / Applies only to BCI 800 R series

Table: Alarm codes

**Note**

With some device series, two alarms can occur at once. For example, the alarm „OV“ then also generates a „Power fail“ (PF, number 28) alarm.

#### How does the alarm management work?

In case of an error according to the adjacent table, it is classified according to an alarm type (see below) and put into an alarm buffer with 3 indexes. This buffer can be read by object 77 (see [object lists](#) for the layout).

It applies, that the first error will be shifted down the buffer by the next error. If the buffer is full, the next error will always overwrite the last error. Reading the buffer will either empty it automatically (with series EL3000/9000 and PS 8000) or it is required to request action by setting a bit in object 54 (with series PSI 9000, PSI 8000, PSI 800 R and BCI 800 R).

#### What is an alarm type?

About the meaning and the differences of alarms, warnings and notifications at power supplies of the series **PSI 9000**, **PSI 8000**, **PSI 800R** and battery chargers of series **BCI 800 R** please refer to the user guide of your device. Other device types like the **EL 3000 / EL 9000** only use alarms and alarm types 0x01 or 0x02.

#### Alarm types:

- 0x01 - Alarm is currently active
- 0x02 - Alarm is not active anymore
- 0x10 - Warning currently active
- 0x20 - Warning not active anymore
- 0x40 - Notification only

The error type will be returned together with an error code if the alarm buffer is queried from the device and can then be analysed. Warnings and notifications have lower priority than alarms, are particularly overwritten and thus have to be considered as either less important or not important at all.

## 4. Profibus (IF-PB1)

### 4.1 General

The Profibus, short for Programmable Field Bus, is an independent field bus standard for a wide range of applications in measurement technique and automation. In compliance to IEC 61158, Profibus guarantees flawless communication between devices of various manufacturers. A lot of different interfaces for PCs, PLCs etc. are available on the market.

The interface IF-PB1 supports Profibus DP (i.e. *decentralised peripherals*), which is especially for fast field bus communication. The data transfer is done via a RS485 connection and operates at bus speeds of up to 12MBaud.

A Profibus is separated into segments. Every segment consists of max. 32 participants. Bus participants are the master, slaves, repeaters and converters. Repeaters and converters are required if the number of participants exceeds 32. Every Profibus participant is assigned a dedicated address, resulting in max. 31 participants for the first and last segment and max. 30 participants for middle segments. Addresses 0, 126 and 127 are reserved.

Conclusion: A total of 125 slaves with IF-PB1 cards can be used on one Profibus.

In dependency of the cable length of the bus, baud rates between 9.6kBAud and 12MBaud can be selected for the bus. Notice that a wrong selection of the bus speed, according to the cable length, might not guarantee flawless communication anymore.

The interface card IF-PB1 supports all available bus speeds, it features galvanic isolation between field bus and device of up to 1000V and is supplied with power by the device.

### 4.2 Technical specifications

Ordering code	IF-PB1
Communication profiles	Profibus DP-V0, Profibus DP-V1 Class 1
Interface type	9 pole D-SUB socket
Communication hardware	RS485
Network topology	Line (without repeater), Tree/Line (with repeater)
Number of participants	max. 31 (without repeater), max. 125 (with repeaters)
Participant type	Slave
PNO Identification number	A000 (hex)

### 4.3 Supported devices

IF-PB1 supports and is supported by following device series (date 08/2012):

- PSI 9000 (old series until 2012)
- PSI 8000
- PS 8000

### 4.4 Master-Slave communication

#### 4.4.1 Cyclic

The Profibus-DP distinguishes between master and slave participants. The master controls the bus communication and requests data from or sends data to the slaves. In a typical master-slave system, input data, output data and diagnosis data are exchanged between the master and all his assigned slaves. The master, for example a PLC, stores input data in its internal memory for the control application. Output data are transferred to the slaves with the next transfer cycle.

So the master always contains a full set of data from its slaves in its memory, but these data have a delay of one cycle.

In case of the IF-PB1, the actual values and the device condition are constantly, i.e. cyclically, transferred to the master.

#### 4.4.2 Acyclic

Additionally to the cyclic data exchange, data can also be transferred acyclically. This has the advantage to access certain single parameters or set values of the field bus device. The acyclic data exchange can reduce the amount of transferred data to a minimum and thus decrease bus load significantly.

### 4.5 Cabling / Termination

The connection of the IF-PB1 to other slaves on the bus or the master system is done with a 2-wire, screened cable. Two variants, type A and type B, of the bus cable are specified in the IEC 61158. Cable type B is not recommended and should not be used for new installations.

Cable type A:

- Cable: twisted pair, screened, 1x2
- Wave resistance  $R_w$ : 135-165  $\Omega$  at 3..20MHz
- Capacity: < 30pF/m
- Loop resistance  $R$ : 110 $\Omega$ /km
- Wire cross section: >0.34mm<sup>2</sup>

The D-Sub socket on the IF-PB1 connects the device with the Profibus cable.

It is used to connect the ingoing bus wires as well as the outgoing ones. The wires „Ingoing“ and “Outgoing“ must not be exchanged, because a standard Profibus plug cuts off the outgoing wires as soon as the bus termination is activated at a certain participant. The bus is not interrupted when disconnecting one device (power-off or disconnection). This applies when multiple units with IF-PB1 card are connected to the bus.

Activating the bus termination resistor will prevent reflections on the bus ends and provides a clean idle level on the bus.

## 4.6 Transfer speed and delays

A list of available transfer speeds for the IF-PB1 Profibus card can be found in section 4.6.1. Refer to your device operating guide resp. the interface cards manual for details on how to access and alter the communication settings.

The **transfer speed** only defines how fast data is transported between the Profibus master and the Profibus slave (i.e. gateway). The Profibus gateway chip processes the data in the Profibus transmission and forwards them to the device with a fixed baudrate of 57600. The same applies for the reverse direction. The data on the Profibus is refreshed by the gateway within a defined period of time, but the data itself is depending on how often it is refreshed by the device. That interval varies from series to series.

The slave automatically puts data (status, actual values) on the bus in a certain interval (DP-V0), in this case 1x per cycle. That data which is also read once per cycle from the device is buffered in the gateway. One cycle is about 660ms.

Furthermore, if data is sent to the device by DP-V1 (set values etc.), these are also forwarded to the device once per cycle.

This results in certain **delays**, until a set value „arrives“ on the DC output resp. DC input of the devices. The delay comes from the data transmission time to the gateway, processing time in the gateway, transmission time to the device and processing time in the device. If, for example, a set value would be sent, then the actual value which belongs to the most recent set value can not be read within the same cycle. Why? Because in the first cycle, the set value is received from the master and sent to the device. In the next cycle, the actual value is measured. That measured value may still not belong to the new set value, so it might only be read in the next cycle. So in the worst case it may take about three cycles to get the corresponding actual value. It is similar to setting a status like „output off“ and reading the device status with DP-V0 to verify that the output has been set to off.

In the standard configuration of the IF-PB1, an operation like „set a value and then wait for the corresponding actual value to appear on DP-V0 data“ usually takes 1.2s...1.5s. This can only be decreased with optimized scripts, of which one can reduce that time down to approx. 0.35s. The optimized scripts will still deliver a new actual value with every cycle, but the status and/or alarm condition is only updated every 2nd or 3rd cycle.

The optimized scripts can be uploaded to the gateway chip by the user herself/himself and are available upon request

### 4.6.1 Influence of cable lengths

For the data transmission speed and thus time between Profibus master and slave, the length of the cable determines the max. speed that can be used without problems. The table below lists common Profibus speeds of which some are available for the IF-PB1:

Transfer speed (kBit/s)	max. cable length (m)
9.6	1200
19.2	1200
45.45	1200
93.75	1200
187.5	1000
500	400
1500	200
3000	100
6000	100
12000	100

## 4.7 Operation modes

A jumper matrix on the IF-PB1 board is used to switch between different operation modes of the interface card. Those modes are for normal operation or maintenance purposes.

The **RS 485/D-Sub port** is used for:

- Profibus (field bus communication)

The **USB port** (virtual serial port) is used for:

- Script updates with WINGATE software or Script Programming Tool (SPT)
- Firmware updates of the device controller (Update Tool)
- Firmware updates of the Profibus stack with Firmware Download Tool (FDT)

The card features a LED, which indicates a proper bus connection and communication readiness by being not lit. It is lit during the initialisation sequence of the Profibus slave controller or when bus connection errors occur. Thus it will be lit for a certain time after the device is switched on.

As soon as the slave units are disconnected from the bus, it will also be lit indicating the error. A detailed description of the momentary condition of the communication is commonly provided by the software configurators of the master systems.

### 4.8 Profibus slave address

Every bus participant is required to have a unique bus address. This address has to be defined before any unit is connected to the bus or at least before the unit is running on the bus. It is advised to set the Profibus address (not to mix up with the „device node“) during the installation of the unit, as described in the user guide of the device. No address must be double. We recommend to assign addresses in ascending order, whereas address 0 is reserved for a PC or programming device.

The assignment of all addresses for the Profibus participants has to correspond to the address assignment in a project on the master system. Only then a secure identification of the power supplies is provided.

Note: The address that is selected in the setup menu of the particular power supply is stored and will be restored after the next start or after a reset by the reset button.

### 4.9 Communication with the master

**Note**

*The data, as transferred in the datagrams which are described below, is of different type. Set values and actual are, for example, of type integer and contain a per cent value. Refer to complete sections 1.7 and 3.3 - 3.8 for more information.*

#### 4.9.1 Cyclic field bus data

Cyclic field bus data, which are transferred from the slave (IF-PB1) to the master, are actual values and the device conditions, in this case. But this will only happen, if the IF-PB1 accepts the configuration that is sent by the master. The type of the data for the cyclic data exchange is sent by the master in a configuration message. The data width of the particular module can be obtained from an object list which is related to the de-vice series. The width can be obtained by help of the hardware configurator from the properties of a particular module slot. It corresponds to the address width of, for example, a PLC.

Rules given in the object list also apply for data formats and the actual values resolution of the power supply. These have to be considered when programming user applications.

#### 4.9.2 Acyclic field bus data

Acyclic data exchange between master and slave only happens upon request of the master. The IF-PB1 supports the communication to a class 1 master. The datagrams listed in section 4.9.2.1 are suitable fore this type of data exchange and are separated into several functional groups (also see the device related object list).

##### 4.9.2.1 Datagram groups

Device information DP-V1 (outputs from slave)

Datagram	Slot	Index	Length	Type	Object in list *	Access from host
Device type	3	0	16	String	0	Read
Device serial number	3	1	16	String	1	Read
Device nom. voltage	3	2	4	Float	2	Read
Device nom. current	3	3	4	Float	3	Read
Device nom. power	3	13	4	Float	4	Read
Device article number	3	4	16	String	6	Read
Device firmware version	3	5	16	String	9	Read
Momentary set values	3	9	6	Integer	72	Read
Device alerts	3	10	6	Byte	77	Read

Control DP-V1 (inputs to slave)

Datagram	Slot	Index	Length	Type	Object in list *	Access from host
Set device condition	3	6	2	Byte	54	Write
Set value of voltage	3	7	2	Integer	50	Write
Set value of current	3	8	2	Integer	51	Write
Set value of power	3	15	2	Integer	52	Write

\* See external object lists (PDF), also see section 3.8

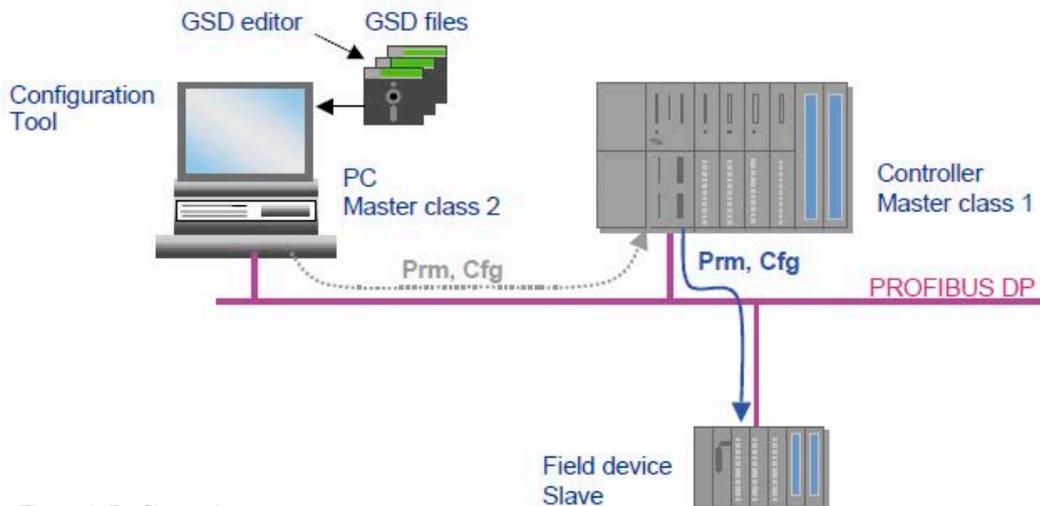


Figure 1: Profibus scheme

Alerts and actual values DP-V0 (outputs from slave)

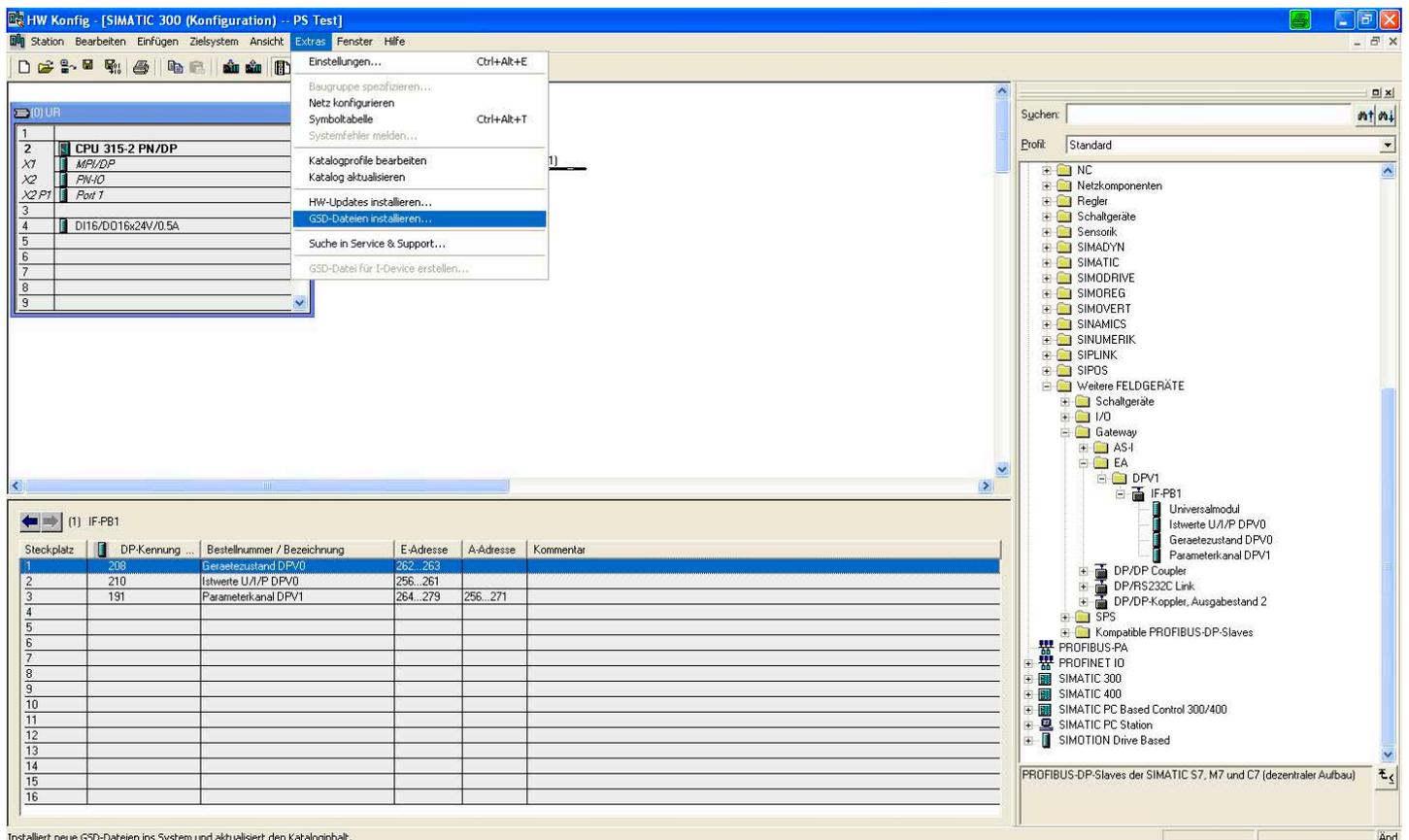
Datagram	Slot	Index	Length	Type	Object in list *	Access from host
Actual values (U,I,P)	2	-	6	Integer	71	Read
Device condition	2	-	2	Byte	70	Read

\* See external object lists (PDF), also see section 3.8

4.9.3 Using the datagrams

If you want to read or write a datagram via acyclic access in the user application, it means with DP-V1, the particular datagram required to pass its INDEX (see datagram groups above) and SLOT, along with an ID (where necessary, represents the module slot's address) to the function blocks. When reading, the length of the data to read is required as input parameter of the standard SFBs (system function block) of PLC software.

The data to write will usually be hexadecimal bytes. They can be converted to decimal values, if only decimal input is available. The contents of the data is defined in the so-called **object lists**, on the basis of the communication protocol that is used with the device. The Profibus message to and from the slave just contains the data part of the actual message. See sections 3.2 and 3.3.2 for important details.



Installiert neue GSD-Dateien ins System und aktualisiert den Kataloginhalt.

Figure 2: Required slot/module configuration for IF-PB1 slave

4.9.3.1 Examples

1) Activate/deactivate remote control

Remote control condition is required if you want to control the device. So the condition of the device is changed and „Set device condition“ has to be used. This is related to object 54 in the object list and has a length of 2 bytes. According to object 54 and in order to switch remote control on or off, the mask byte has to be 0x10 and the control byte 0x10. So the bytes 0x10 0x10 are sent to slot 3 or ID 264 (address of slot 3, according to the default slot config) and index 6. To deactivate remote control, bytes 0x10 0x00 would then be sent to slot 3, index 6.

2) Switch DC output on or off

Switching the DC output on or off changes the device condition, hence datagram „Set device condition“ is the choice. It is related to object 54 and has a length of 2. According to object 54 and in order to switch the output on the mask byte has to be 0x01 and the control byte 0x01. These bytes are sent to slot 3 or ID 264 (address of slot 3, according to the default slot config) and index 6. Setting the output off would require to send the bytes 0x01 0x00 to slot 3, index 6.

3) Set voltage to 40V

This requires to translate a real voltage value to a per cent value first. In case you have an 80V model, then 40V would be 50%. In case of a 720V model, 40V would be 5.55%. See section „1.7 Translating set values & actual values“ for details about set value translation. With 50% the hex value would be 0x3200, sent as bytes 0x32 0x00 to slot 3 or ID 264 (address of slot 3, according to the default slot config) and index 7.

## 4.10 Example project with a Siemens PLC

### 4.10.1 Integrating the power supply into a Profibus

In order to integrate a power supply or similar by means of an IF-PB1, a hardware configurator is required, which is part of the software package belonging to the Siemens PLC. Other configurators are available from companies like Beckhoff or Hilscher. Besides this, a GSD (Generic Station Device) file is required which is delivered on the CD included with the IF-PB1 or can be obtained upon request as well as from the website of the PSU manufacturer. The file is named PBPSA000.GSD (german version) resp. PBPSA000.GSE (english version).

This file is used to announce the communication modules and required parameters of the power supply to the user application in the configurator. The cyclic data exchange on the bus will be defined.

### 4.10.2 GSD file installation

The GSD or GSE file is installed in the master application with the menu command „Extras -> Install GSD files“. See figures 3 and 4 (from the german version of STEP7).

After the successful installation of a GSD file, the Profibus interface of the power supply can be found in the hardware catalogue in „Profibus-DP -> Other field devices -> Gateway ->EA ->IF-PB1.

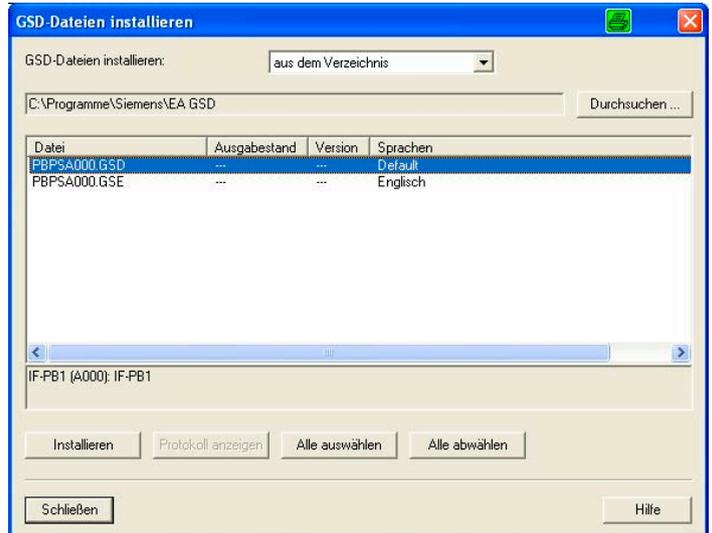


Figure 3: GSD/GSE installation

### 4.10.3 Module placement

The modules, which are now located in the profile catalogue, need to be dragged & dropped in the slot configuration window. In order to enable full communication with the device, the modules have to be placed as follows:

- Slot 1 : Actual values U/I/P DPV0
- Slot 2 : Device condition DPV0
- Slot 3 : Parameter channel DPV1

Also see figure 4 below.

If module „Parameter channel“ is not placed, the device can not be controlled and acyclic data exchange is disabled.

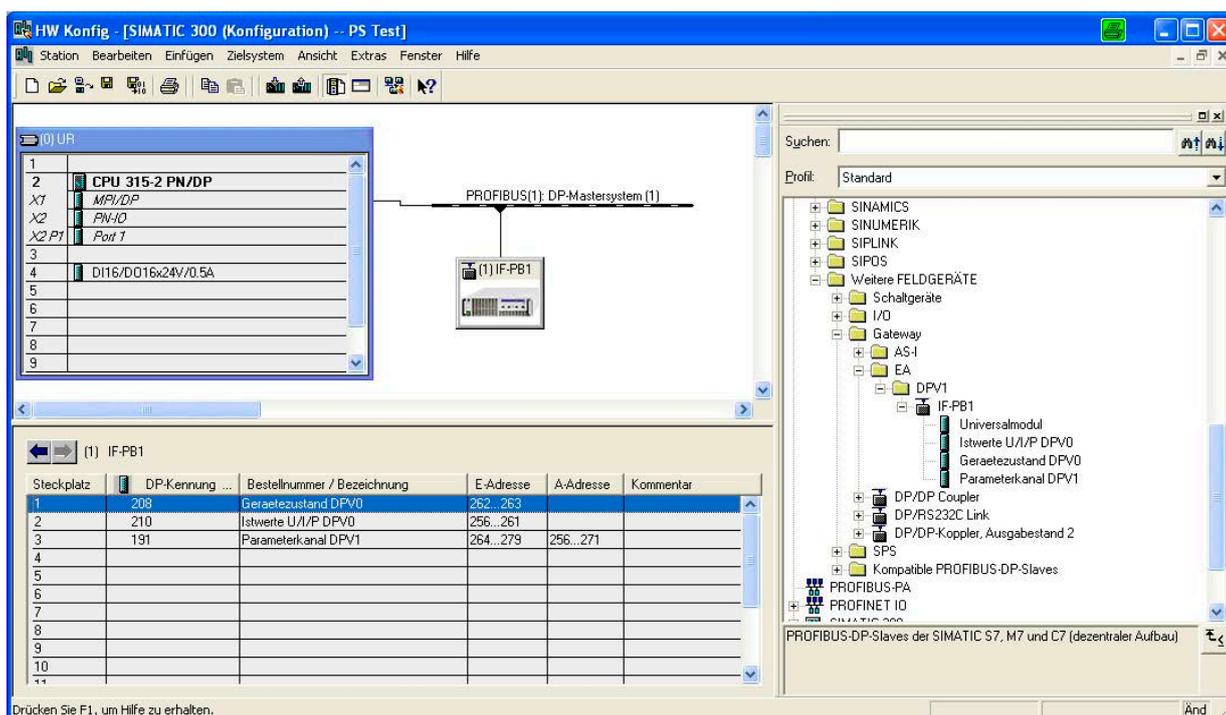


Figure 4: Required module placement

#### 4.10.4 Addressing the cyclic modules

For the modules of cyclic data exchange it applies generally:

- All modules in the hardware catalogue are assigned to a logical address
- The logical addresses build a part of the control peripherals
- The peripheral addresses of the single modules may not collide and thus not assigned twice
- The assigned peripheral address has to be used in the user application to access data

#### 4.10.5 Addressing the acyclic module

For the module of acyclic data exchange it applies:

- The peripheral address(es) must not conflict with the ones of cyclic modules
- Because this module is a mixed type (input/output), the addresses of input and outputs must also not conflict. This is based upon the design of the Siemens communication blocks (SFB52 and SFB53) for acyclic data exchange
- When accessing data access via an interface which is standardised according to PNO AK 1131, an ID will be requested first which consists of the lower one of both addresses. The address is the basis and the first part of two information, that are required for acyclic data exchange
- The second information is the datagram number that shall be read or written. The standard here speaks of a so-called INDEX

### 4.11 Diagnosis

#### 4.11.1 Standard diagnosis

The standard diagnosis is a message of 6 bytes. The message is exchanged during the initialisation of the IF-PB1 between the interface and the master, unnoticed by the user. But it can be interpreted by an adequate engineering tool. In STEP7 it is done with „Target system -> Show present participants“ and can be displayed by selecting the particular field bus device with „Target system-> Diagnosis/Setup -> Component condition“ (see fig. below). The diagnosis message is furthermore a part of the permanent data exchange that happens in case of an error where the IF-PB1 will send additional information over the field bus. See figure 5 below.

For the IF-PB1 the diagnosis message will look like this during normal operation: 0x80 | 0x0C | 0x00 | 0x02 | 0x0A | 0x00

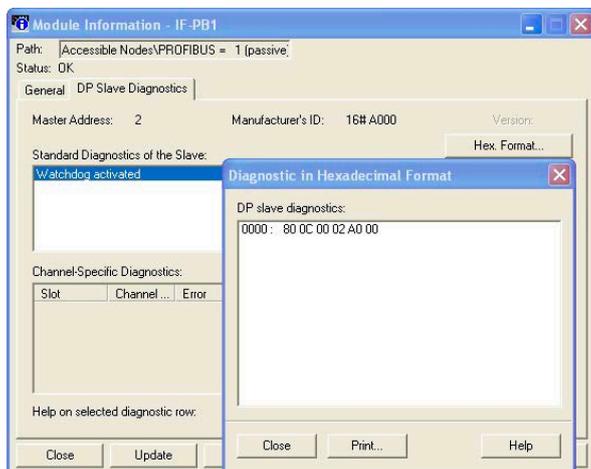


Figure 5: Standard diagnosis

#### 4.11.2 DP-V1 alarm management (extended diagnosis)

In excess of the standard diagnosis, as specified in the communication mode DPV0, the user can receive extended diagnosis information from the IF-PB1.

In communication mode DPV1 the so-called device based diagnosis is build from alarm blocks and/or condition reports. To activate this syntax the bit DPV1\_Enable in the parameter message has to be set to 1.

The IF-PB1 makes use of the condition report. It is defined in the IEC61158 as follows:

Byte	Bit	
1	0	Station does not exist (set by master)
	1	Slave is not ready for data exchange
	2	Configuration data not concurring
	3	Slave has extended diagnosis data
	4	Requested function not supported by slave
	5	Invalid response from slave (set by master)
	6	Wrong parameterisation
	7	Slave is already parameterised by a different master (set by master)
2	0	Slave has to be parameterised again
	1	Static diagnosis
	2	Set to "1"
	3	Watchdog active
	4	Received Freeze command
	5	Received Sync command
	6	Reserved
	7	Slave is deactivated (set by master)
3	7	Diagnosis overflow - Slave has more diagnosis information than fits the message
4	0...7	Master address after parameterisation (0xFF without parameterisation)
5	0...7	Ident number high-byte
6	0...7	Ident number low-byte
7		Header: - The header shows the block length of the extended diagnosis, including the header byte - For this module this value is 0x05 (bytes 7...11 = 5 bytes)
8		Status_Type: The value is fixed to „0x81“ with following bit significance: - Bit 7 = 1: „Condition“ - Bit 0 = 1: „Condition report“
9		Slot_Number: Value is „0x00“
10		Specifier - A new error will be marked in the specifier with „0x00“ (Condition coming) - A removed error will be marked in the specifier with „0x02“ (Condition leaving) - If no error is reported, the specifier value will be „0x00“
11		User byte: Detailly explained below

Continued on next page...

The extended diagnosis for an IF-PB1 will result like this, according to the table above:

0x05 | 0x81 | 0x00 | 0x00 | **0x??**

The last byte defines the actual error information in form of a code, which either represents a communication error (see table in 3.7.2 on page 35) or a device error (see table in 3.9).

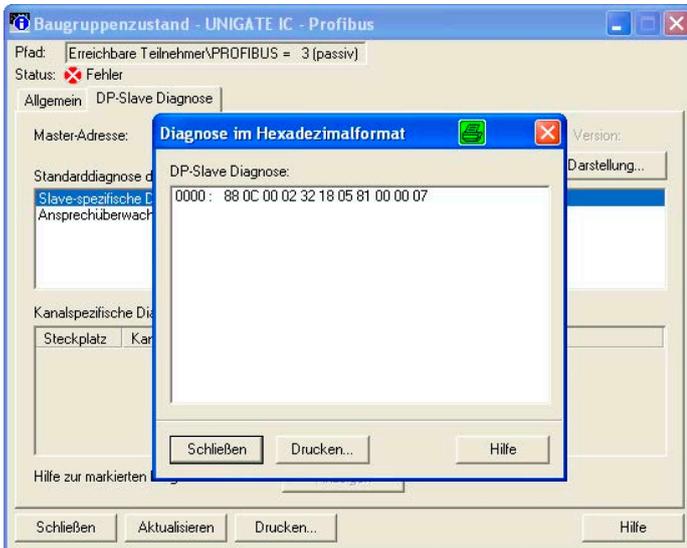


Figure 6: Extended diagnosis (german screenshot)



Elektro-Automatik

**EA-Elektro-Automatik GmbH & Co. KG**

Entwicklung - Produktion - Vertrieb

Development - Production - Sales

Helmholtzstraße 31-33

**41747 Viersen**

**Germany**

Telefon: +49 (0) 2162 / 37 85-0

Telefax: +49 /0) 2162 / 16 230

ea1974@elektroautomatik.de

www.elektroautomatik.de