Elektro-Automatik

Programming Guide

# ModBus & SCPI

## For USB, GPIB, Ethernet and AnyBus modules

**Attention! This document is only valid for the device series listed in section *1.1.2* and also only valid as from the firmware version listed there.**

Doc ID: PGMBEN
Revision: 13
Date: 10-31-2016

# TABLE OF CONTENTS

# 1. General

## 1.1 About this document

### 1.1.1 Copyright

Reprinting, copying, also partially, usage for other purposes as foreseen of this manual are forbidden and breach may lead to legal process.

### 1.1.2 Validity

This manual is valid for the below listed equipment including derived variants. This programming guide is connected to the firmware version of the communication unit KE. Older KE firmwares can thus be partially incompatible with this document. It is recommended to remember the abbreviation listed in the table below, as it is a reference for below parts of this document when programming your particular device.

| Series | Models | Firmware | Abbreviation |
|---|---|---|---|
| ELR 9000 | all | KE: 2.15 (Anybus) / KE: 2.04 (GPIB) or higher | ELR9 |
| PSI 9000 2U / 3U (series from 2014) | all | KE: 2.15 (Anybus) / KE: 2.04 (GPIB) or higher | PSI9 |
| PSI 9000 15U / 24U | all | KE: 2.15 (Anybus) | PSI9 |
| PS 5000 | all | KE: 2.02 or higher | PS5 |
| PSI 5000 | all | KE: 2.06 or higher | PSI5 |
| PS 9000 1U/2U/3U (series from 2014) | all | KE: 2.06 or higher | PS9 |
| EL 9000 B | all | KE: 2.15 (Anybus) / KE: 2.04 (GPIB) or higher | EL9R |
| ELR 5000 (ELM 5000) | all | HMI: 2.03 or higher | ELR5 |
| PSE 9000 | all | KE: 2.01 | PSE |
| PSI 9000 DT | all | KE: 3.02 | DT |
| EL 9000 DT | all | KE: 3.02 | DT |

## 1.2 Explanation of symbols

Warning and safety notices as well as general notices in this document are shown in a box with a symbol as follows:

| | |
|---|---|
|  | Symbol for general safety notices (instructions and damage protection bans) |
|  | *Symbol for general notices* |

## 1.3 Warranty

The manufacturer guarantees the functional competence of the applied technology and the stated performance parameters. The warranty period begins with the delivery of free from defects equipment.

Terms of guarantee are included in the general terms and conditions of the manufacturer.

## 1.4 Limitation of liability

All statements and instructions in this manual are based on current norms and regulations, up-to-date technology and our long term knowledge and experience. The manufacturer accepts no liability for losses due to:

- Usage for purposes other than designed
- Use by untrained personnel
- Modification of devices by the customer
- Technical changes of devices
- Use of not authorized spare parts

# 2. Anybus modules

## 2.1 Overview

| Module | Type / Connectors | LED indication | | Front view |
|---|---|---|---|---|
| IF-AB-CAN | CAN 2.0B,<br>1x Sub-D 9pole,<br>male | RUN | Indicates data traffic (green, flickering) |  |
| | | ERR | Will be lit (green) while a communication error is present. | |
| IF-AB-CANO | CANopen,<br>1x Sub-D 9pole,<br>male | RUN | Indicates status with flash sequences according to DR303-3 (CiA) |  |
| | | ERR | Indicates status with flash sequences according to DR303-3 (CiA) | |
| IF-AB-RS232 | RS 232,<br>1x Sub-D 9pole, male, for null modem cable | PWR | Module is powered |  |
| IF-AB-PBUS | Profibus DP-V1 Slave,<br>1x Sub-D 9pole, female | OP | Operation mode:<br>on (green) = Connection established<br>flashing (green) = Ready<br>flashing (red, 1x) = Parameter error<br>flashing (red, 2x) = Profibus error |  |
| | | ST | Status<br>off = Not initialised<br>on (green) = Initialised<br>flashing (green) = Extended diagnosis<br>on (red) = Exception error | |
| IF-AB-ETH1P | Ethernet,<br>1x RJ45 | NS | Network status:<br>flashing (green) = default, can be ignored<br>on (red) = Double IP, fatal error<br>flashing (red) = Connection time-out |  |
| | | MS | Module status:<br>flashing (green) = default, can be ignored<br>on (red) = Exception error<br>flashing (red) = Recoverable error | |
| IF-AB-ETH2P | Ethernet,<br>2x RJ45 | | |  |
| | | LINK | Connection status:<br>on (green) = Connection established<br>flashing (green) = Data traffic | |
| IF-AB-PNET1P | ProfiNET IO,<br>1x RJ45 | NS | Network status:<br>on (green) = Online with controller in RUN<br>flashing (green) = Controller in STOP |  |
| | | MS | Module status:<br>on (green) = Everything OK<br>on (red) = Exception error<br>flashing (red, 1x) = Config error<br>flashing (red, 2x) = IP address not set<br>flashing (red, 3x) = Station name not set<br>flashing (red, 4x) = Internal error | |
| IF-AB-PNET2P | ProfiNET IO,<br>2x RJ45 | | |  |
| | | LINK | Connection status:<br>on (green) = Connection established<br>flashing (green) = Data traffic | |

| Module | Type / Connectors | LED indication | | Front view |
|---|---|---|---|---|
| IF-AB-MBUS1P | ModBus TCP,<br>1x RJ45 | NS | Network status:<br>on (green) = Module active<br>flashing (green) = Module waiting for connection<br>on (red) = Double IP or fatal error<br>flashing (red) = Process time-out |  |
| | | MS | Module status:<br>on (green) = Everything OK<br>on (red) = Primary error<br>flashing (red) = Secondary error | |
| IF-AB-MBUS2P | ModBus TCP,<br>2x RJ45 | | |  |
| | | LINK | Connection status:<br>on (green) = Connection established<br>flashing (green) = Data traffic | |
| IF-AB-ECT | EtherCAT Slave,<br>2x RJ45 | RUN | Indicates status with flash sequences according to DR303-3 (CiA) |  |
| | | ERR | Indicates status with flash sequences according to DR303-3 (CiA) | |

## 2.2 Anybus module support

Those device support the in *2.1* listed Anybus modules (date: 10-31-2016):

- ELR 9000
- EL 9000 B
- PSE 9000
- PSI 9000 2U / 3U

*It might be required to install a firmware update for your device in case the device won't support a specific interface module after installation.*

## 2.3 Before you begin

If you plan to integrate a device into an existing network or field bus with any of these interface installed, notice following:

- All modules, but especially the Ethernet types which provide a web site, require a certain startup time each time the device is powered, which will delay them to be addressed in the network. Usually, the interface module is ready for communication as soon as the device is ready for operation.
- The readiness for operation may be indicated by the modules (with one of the LEDs) before the required startup time has run out. If one would try to contact an Ethernet module in order to access the website, the website might not be loaded completely or the browser might stop with a time-out error.

## 2.4 Installation of an Anybus module

The installation itself is described in the operating guide of your device, as well as the required setup. Further information about installation and connection to field buses and networks can be found in publicly available documentation and similar sources.

*The CANopen module IF-AB-CANO does not feature an internal termination resistor. Thus the required bus termination resistor has to be applied by the user according to the CAN bus requirements.*

## 2.5 Network with linear topology

The Ethernet based modules for standard LAN, ModBus TCP and Profinet/IO are also available in a version with two ports. These provide the possibility to connect multiple devices in a linear topology, like a bus, and even to build a ring (DLR, device level ring) for extended safety against interruption. External switches can be spared and the many long network cables, like when having a star-shaped topology, can be reduced to a minimum.

The EtherCAT module, however, has two port by default and always builds a ring because of the standard setup with EtherCAT system. It is also Ethernet based, but cannot be considered as LAN port.

## 2.6 Network access via HTTP

Most of the Ethernet based modules, like for standard LAN, ModBus TCP or Profinet, and the integrated LAN port of selected series (PS 9000 from 2014, PSI 5000) offer a website. It is accessible with a browser (Firefox, Chrome, Safari) by simply entering the IP address or the host name which has been assigned to the device. The default IP is **198.168.0.2**. All network parameters for the device/network interface can be changed or reset to defaults in the setup menu of the device (where featured).

The currently active IP address, along with other network related parameters like gateway, DNS address, subnet mask and MAC address, can also be read from an overview in the setup menu of devices where the series features an on-screen setup menu.

Accessing the website via the host name (default: Client) is only possible if the network runs a DNS or, at direct connection, the PC does and the domain/host name is already registered.

The website gives the user full control over the device by manually typing SCPI commands. It primarily serves for testing purposes only. In case you want to continuously control a device or at least monitor it, please continue reading in section *„5. The SCPI command language“ on page 29*.

The website, precisely the second page CONFIGURATION, allows for setting up network specific parameters, like when doing it in the device setup menu, and to write them to the device remotely, requiring prior activation of remote control by command SYST:LOCK ON.

## 2.7 Network access via TCP

All Anybus network modules, as well as the integrated LAN port of selected series offer standard TCP access via the default port **5025** (user-selectable, see device operating guide for setup menu or similar). TCP data transfer is used for the external communication via ModBus RTU or SCPI protocol. Some Anybus modules feature additional ports, which are reserved and cannot be changed.

The port and other network related parameters can either be adjusted in the device's setup menu (if featured) or from outside via USB or on the website (see *2.6*).

TCP/IP socket connection (IP:port) is intended for normal remote control access to the device.

*The TCP connection is automatically disconnected by the device after a certain amount of time elapsed with no data transmission. This is called keep-alive timeout and is currently set to: 5 seconds.*

# 3. Introduction

## 3.1 Communication with a device in general

After connecting the device via a digital interface to a PC, it is usually ready for access. Access can happen in several ways:

- Via a control and monitoring software supplied by the device manufacturer
- Via LabView VIs, supplied by the device manufacturer
- Via a custom programmed application which is usually created by the user
- Via other software, like a terminal program that can send text messages (SCPI)
- Via internationally standardised software for CAN, CANopen, Profibus or EtherCAT etc.

## 3.2 Control locations

Control locations are those locations from where a device is accessed. There are basically two: direct access (manual control) and external access (remote control). The user can switch between control locations just as the situation requires.

Following control locations are defined for a device:

| Control location as displayed | Description |
|---|---|
| **-** | If there is no specific control location indicated, the device is free for external access and all interface for remote control are enabled |
| **Remote Analog, Remote USB etc.** | Remote control is active |
| **Remote** | Remote control is active via any interface (PS 5000 / PSI 5000 only) |
| **Local** | Remote control is blocked, the device can only be controlled manually |

Certain device series offer a feature to interrupt or to block remote control of the device completely. This is a setting named „**Allow remote control**“ or similar. In blocked state, the device will indicate “**Local**” in the display. Activating this block may be required in an emergency situation where a software permanently controls the device from remote and thus may prevent user interaction. With this, the user can temporarily interrupt remote control in order to access the device for switching the DC input/output off or changing a setting.

Activating „**Local**“ condition will cause following:

- In case remote control via one of the digital interfaces is currently active („**Remote Ethernet**“ etc.), remote control will be deactivated and can be activated again later, once the „**Local**“ condition has been deactivated, too.
- In case remote control via analog interface is currently active („**Remote Analog**“), then the remote control is only interrupted as long as condition „**Local**“ is active and returns automatically once “**Local**” is deactivated, because the analog interface usually has steady signals like on pin REMOTE, which still says “Remote on” unless the plug on the analog interface connector has been removed.

## 3.3 Message timing and command execution time

The timing of communication, more precisely the control over the chronological run of two subsequent messages, is not done by device and lies in the responsibility of the user.

Rule of thumb: the device can not process incoming messages as fast they can be technically transferred by the hardware of the used interface and its specifications. Thus it is important to time communication and wait a certain time before the next command is sent, no matter what interface is used. This doesn’t include protocol related data traffic, like it occurs for example between a Profibus slave and its Profibus master, because this traffic is handled by the interface itself, not by the device.

It also applies:

- Queries to the devices, i.e. commands that read something, are executed faster and may be sent more often and in shorter intervals
- Write commands, i.e. commands that set a value or a status, are not immediately executed when received and the delay before execution varies

### 3.3.1 Execution time when writing

Due to different internal design, different types of microcontrollers, which control the hardware and besides do communication with the PC, and also different firmwares the time for processing a set command, i.e. write command, constantly varies. It means, there are definite values how long it takes to finally see a value on the DC output/input after you wrote the command. An average value for the given situation where the device is working in can be determined by the user by triggering and measuring.

### 3.3.2 Response time when reading

Reading something from a device is usually responded immediately, with a certain response time. There are generally two methods of communicating with a port:

1) Open port -> write to port -> read from port -> close port

2) Open port -> write to port -> read from port -> repeat write/read x times -> close port

Reading a response from the port is generally required with ModBus, because there can be an expected response message or an unexpected error message. Using SCPI, reading from the port is only required when querying something with a command ending with a "?".

Both methods have advantages and disadvantages. The primary advantage of method 2 over method 1 is that writing and reading can result in an even faster response time. The primary advantage of method 1 over method 2 is that closing the port also closes the connection which makes communication more stable, especially if the time between to write-read cycles is very long. The values in the table below are measured using method 1.

*The response time of both, USB and Ethernet, may vary depending on the KE firmware version installed on your device and can better or worse than listed here. Please compare the KE version in the table, where given, with the one on your device.*

| Series | Typical response time |
|---|---|
| ELR 9000 / EL 9000 B | SCPI: via USB <5 ms (2, via Ethernet <10 ms (1<br>ModBus: via USB <5 ms (2, via Ethernet <10 ms (1 |
| PSI 9000 (2014 series) | SCPI: via USB <5 ms (2, via Ethernet <10 ms (1<br>ModBus: via USB <5 ms (2, via Ethernet <10 ms (1 |
| PS 5000 | ~1 ms |
| PSI 5000 / PSI 9000 DT /<br>EL 9000 DT | SCPI: via USB 1~3 ms, via Ethernet 5~8 ms<br>ModBus: via USB ~1 ms, via Ethernet ~10 ms |
| PS 9000 (2014 series) | SCPI: via USB 1~3 ms, via Ethernet 5~8 ms<br>ModBus: via USB ~1 ms, via Ethernet ~10 ms |
| ELR 5000 | SCPI / ModBus: 1~3 ms |
| PSE 9000 | SCPI: via USB <5 ms (2, via Ethernet <10 ms (1<br>ModBus: via USB <5 ms (2, via Ethernet <10 ms (1 |

1) Acquired from a measurement of 200,000 command cycles of method 1 via interfaces IF-AB-ETH (Ethernet), IF-AB-PNET (Profinet, TCP via port 5025) and IF-AB-MODB (standard Ethernet port 5025)

2) Acquired from a measurement of 200,000 command cycles of method 1 via virtual COM port of USB driver

### 3.3.3 Timing of messages

The minimum between two messages, as listed below, primarily depends on the typical response time as listed in *3.3.2*.

| Series | Minimum time between two messages | Recommended time between two messages |
|---|---|---|
| ELR 9000 /<br>EL 9000 B | USB: 10 ms, Ethernet: 15 ms | USB: 20 ms, Ethernet: 30 ms |
| PSI 9000<br>(from 2014) | USB: 10 ms, Ethernet: 15 ms | USB: 20 ms, Ethernet: 30 ms |
| PS 5000 | 2 ms (ModBus over USB only) | 5 ms (ModBus over USB only) |
| PSI 5000 /<br>PSI 9000 DT /<br>EL 9000 DT | SCPI: via USB 2 ms, via Ethernet 8 ms<br>ModBus: via USB 2 ms, via Ethernet 20 ms | SCPI: via USB 5 ms, via Ethernet 15 ms<br>ModBus: via USB 5 ms, via Ethernet 30 ms |
| PS 9000<br>(from 2014) | SCPI: via USB 2 ms, via Ethernet 8 ms<br>ModBus: via USB 2 ms, via Ethernet 20 ms | SCPI: via USB 5 ms, via Ethernet 15 ms<br>ModBus: via USB 5 ms, via Ethernet 30 ms |
| ELR 5000 | 5 ms | 10 ms |
| PSE 9000 | USB: 10 ms, Ethernet: 15 ms | USB: 20 ms, Ethernet: 30 ms |

## 3.4 Overview of the communication protocols

Basically, there are two protocols: **ModBus RTU** and **SCPI**. When using one of the ModBus TCP interface modules, which are optionally available for selected series, and depending on the addressed port, there is a distinction between ModBus RTU and ModBus TCP regarding the message format. While ModBus TCP uses a different frame format and a dedicated port (here: 502), a ModBus RTU message can be sent over Ethernet (every other port) or other interface types.

*With **option 3W** (GPIB+USB+Analog) installed, as it is optionally available since Q3/2014 for selected series, **only the SCPI protocol** can be used via GPIB. With USB, ModBus RTU is additionally supported.*

Which device series support what protocol(s)?

| Series | ModBus RTU | SCPI |
|---|---|---|
| ELR 9000 | ✓ | ✓ |
| PSI 9000 (from 2014) | ✓ | ✓ |
| PS 5000 | ✓ | - |
| PSI 5000 | ✓ | ✓ |
| PS 9000 (from 2014) | ✓ | ✓ |
| EL 9000 B | ✓ | ✓ |
| ELR 5000 | ✓ | ✓ |
| PSE 9000 | ✓ | ✓ |

The integrated **USB-B port** (usually on the rear side) can be used for **both protocols**. Most series support both of the above listed protocols and distinguish them by the first byte of a message, except when using the ModBus TCP protocol. According to the ModBus TCP standard, a six byte MBAP header is added to any ModBus format message and is usually sent using the dedicated port 502. A SCPI message sent to this port would cause a communication error.

Depending on the selection of interface and protocol you are going to use, a different part of this documentation will become relevant.

Some interface standards, such as Profibus/Profinet, may require a different protocol format and transmission scheme towards a bus or network and the interface modules then act as translators. In this case, the user can't use any of the default protocols, but is tied to the standard given by the interface specifications.

The overview below show which **Anybus interface** module can use which protocol:

| Interface | ModBus RTU? | SCPI? | Other protocol |
|---|---|---|---|
| CAN | modified | no | no<br>(see *„8. CAN“*) |
| CANopen | no | no | CANopen<br>(see *„7. CANopen“*) |
| RS232 | yes<br>(see *„4. The ModBus protocol“*) | yes<br>(see *„5. The SCPI command language“*) | no |
| Profibus | no | no | Profibus<br>(see *„6. Profibus & Profinet“*) |
| Ethernet | yes<br>(see *„4. The ModBus protocol“*) | yes<br>(see *„5. The SCPI command language“*) | no |
| ProfiNet | no | no | Profibus<br>(see *„6. Profibus & Profinet“*) |
| ModBus TCP | yes<br>(see *„4. The ModBus protocol“*) | yes<br>(see *„5. The SCPI command language“*) | ModBus TCP<br>(see *4.2*) |
| EtherCAT | no | no | EtherCAT<br>(see *„9. EtherCAT“*) |

## 3.5 Special characteristics of remote control

When using remote control via digital interface, some things have to be taken into account:

- Configuration or control of the function generator (where available) requires a certain procedure. This is described in *4.10.7* for ModBus resp. *5.4.12* for SCPI protocol. The described procedure for ModBus basically also applies to any other protocol used by buses like CAN, CANopen, EtherCAT etc.
- Some ModBus registers resp. SCPI commands are intended for the setup of the device exactly like when doing it manually in the device setup menu (where featured). Those registers/commands are not particularly grouped or marked with colours and should only be used in case to switch between configurations of the device.
- The adjustment limits („Limits“, where available, see device manual), as adjustable in the device's setup menu, limit related set values from every control location, i.e. also in remote control via digital interface. This can lead to unexpected communication errors coming from the device when a value is too high/low. With SCPI language being used, these errors are not even returned automatically, because it is typical with SCPI to receive error messages only upon request. In order to be sure whether a set value has been accepted by the device or not, reading the set value back is the only way.

## 3.6 Fragmented messages on serial transmissions

With RS232, Ethernet or even with USB it is possible, that the device receives fragmented messages. It means, a command is received in pieces together with a certain time gap and then interpreted by the device as multiple, but single and corrupt commands. Primarily SCPI commands are affected, because they are strings consisting of ASCII characters and do not have a checksum. Those strings could be sent by a serial interface character by character or as one single block, depending on the situation on the control side (PC). If a certain timeout elapses between to consecutive bytes, the message is considered as "completely received" by the device, due to the lack of a termination character, which is not generally required for ModBus or SCPI.

Since firmware versions KE 2.07 (series PSI 9000, ELR 9000) and KE 2.04 (series PSI 5000, PS 9000) the devices now feature a ModBus register (see updated register list) resp. SCPI command (see section *5.4.11*) to override the factory default timeout of 5 ms. There is also a corresponding setting in the HMI's setup menu (where a menu is featured).

If the communication between PC and device has a lot of communication errors due to possibly fragmented messages, the timeout should be increased step by step to eliminate the problem. It is advised to keep the timeout setting as low as possible, because at the end of every message the timeout also has to elapse before the device can process the command.

When using SCPI, sending an additional termination character (typical LF, CR, or CRLF accepted) , which is here not necessarily required, will terminate the timeout immediately and let the device consider the message as completely received, so it can process.

## 3.7 Connection timeout

Certain communication lines, in this case it concerns Ethernet / ModBus TCP connections, means network connections in general, can have a connection timeout. This variable and user-adjustable timeout (see user manual of the device) closes the socket connection automatically on the device side once there was no communication between device and controlling unit (PC, PLC etc.) for the adjusted time.

After the connection has been cancelled, it can be established again anytime.

# 4. The ModBus protocol

*Before you continue reading this section, please verify if your device generally supports ModBus and also if ModBus is supported via the interface you are going to use. See section 3.4*

*Our devices are always ModBus slaves with address 0. This address cannot be changed.*

## 4.1 General information about ModBus RTU

A message or telegram as defined by the ModBus RTU protocol consists of hexadecimal bytes, of which the first byte, the ModBus address, should always be 0 (zero) for single units. The first byte of a telegram is used to distinguish the telegram between ModBus and SCPI. A value between 1 and 41 in the first byte will cause a ModBus communication error, whereas from 42 (ASCII character: *) the telegram is considered as text message of a SCPI command.

Format and length of a telegram are defined. The telegram has to be transmitted according to the specifications of the particular interface that is used. Normally, the user only has to take care for a correct message, rather than correct transmission. But there are also interfaces, like for example RS232, which do no feature communication safety and do guarantee flawless transmission. Other interfaces support flawless transmission by using a checksum and/or software handshaking.

## 4.2 General information about ModBus TCP

The message protocol according to ModBus TCP/IP standard (short: ModBus TCP) is only available with the Anybus interface modules ModBus TCP 1-Port and 2-Port and there only via the default ModBus TCP port 502. By definition, a ModBus TCP message requires an additional header of 6 resp. 7 bytes, compared to ModBus RTU. This makes it impossible to use SCPI commands via this port. The rest of the message is identical to ModBus RTU, except for the checksum at the end, which is not necessary with ModBus TCP. Information about the header can be found below. The following sections are related to the core part of ModBus messages, which is identical for both protocols. Also see *„4.9. ModBus TCP in detail“*.

## 4.3 Format of set values and resolution

Set values, as transmitted via digital interfaces, are always per cent values of the device's nominal values (U, I, P, R) and correspond at 100% to the hexadecimal value 0xCCCC (decimal: 52428). The total usable range is 0%...102% (0x0000...0xD0E5). The register lists for a particular series defines the range for all settable values.

It means, you can set a per cent value between 0% and 100% by sending hexadecimal values of 0x0000...0xCCCC resp. for supervision thresholds of device alarms like OVP it will be 0x0000...0xE147 for 0% to 110%.

This means 52429 possible values for 0-100%. This is internally halved (bit 16 is reserved for sign) **and so the usable resolution results in 26214**.

## 4.4 Translating set values and actual values

Real values have to be translated to per cent values before transmitting them to the device, as well as per cent values read from the device are usually translated into real values in order to process them further. It always applies: 0xCCCC (hexadecimal) = 52428 (decimal) = 100% nominal value (U, I, P)

Translation is done by implementing these formulas into custom software:

| Per cent value to real value | Real value to per cent value |
|---|---|
| $\text{Real value} = \frac{\text{Nominal device value} * \text{per cent value}}{52428}$ | $\text{Per cent value} = \frac{52428 * \text{real value}}{\text{Nominal device value}}$ |
| Example: The nominal voltage of your device is 80 V and the percentaged actual voltage was read as 0x2454 = 9300. According to the formula above, the real actual value will be (80 * 9300) / 52428 = 14,19 V. | Example: the power set value shall be 3150 W, the nominal power of your device is 3500 W. According to the formula above we get: Power set value = (52428 * 3150) / 3500 = 47185 = 0xB851. |

*All set values are not only limited by the device's nominal values, but can also be limited by the adjustable "Limits" (where available)! Values exceeding the minimum or maximum of the adjusted range are rejected by the device.*

*When translating real values into per cent values (decimal or hexadecimal), it is often required to round up or down. We recommend to round naturally. Note that natural rounding can result in a translation value which is by 1 higher than expected.*

## 4.5 Communication with the device via AnyBus modules

### 4.5.1 Profinet / Profibus

The Profinet/IO module (1 or 2 ports) can be used to control and monitor a device using a network system, usually combined with an integrated PLC and proper software. For Profinet, the software selects the necessary Ethernet port, because this port can not be adjusted on the device. The standard Profinet communication is different and is handled by the field bus protocol via special software. The implementation of the device into Profinet resp. Profibus is described in section *„6. Profibus & Profinet“*.

### 4.5.2 CANopen

Refer to section *„7. CANopen“*

### 4.5.3 CAN

Refer to section *„8. CAN“.*

### 4.5.4 ModBus TCP

The protocol used here is standard ModBus in a ModBus TCP frame. TCP/IP transmission is not explained herein. Skip to section *4.9*.

### 4.5.5 EtherCAT

EtherCAT uses proprietary software and usually CANopen over Ethernet (CoE) protocol. Refer to sections *„7. CANopen“* and *„9. EtherCAT“*.

## 4.6 Communication with the device via USB port

After connecting the device via USB cable and successful USB driver installation, the device is ready for access. The COM port, which is assigned to the new USB device (see Windows device manager) does not need configuration. It is based upon a so-called CDC driver (Communications Device Class), which is available for Windows XP and 7 (also Embedded) and for other operating systems, too. This driver generates the COM port for simplicity and can run data transmissions as fast as USB 2.0 can handle it. The typical serial settings are not effective and are ignored.

### 4.6.1 USB driver installation

The USB driver for the rear side USB port type B (where available) is included with the device on USB stick. It installs a signed driver for virtual COM ports on 32 bit or 64 bit Windows operating systems since version 7. Alternatively it is available as download from the website of the device manufacturer.

### 4.6.2 First steps

In order to communicate with the device, it actually just requires a software on the PC side which is able to open a COM port and send messages in either binary (for ModBus RTU protocol) or textual (ASCII strings for SCPI) format.

For the latter one, simple terminal softwares suffice. For binary telegrams in hexadecimal format other tools are required, like Docklight (www.docklight.de). The device manufacturer can provide ready-to-use example project files for Docklight upon request. Those can help for a start and to see how the communication works correctly or at all. The project files contain a few basic messages in form of macros which can be sent by the click of a button.

To finally establish communication and access the device via USB, you just need to...

1. connect the device via USB (alternatively via any other digital interface).

2. install the USB driver (see *4.6.1*).

3. run a terminal program.

The device also “understands” the internationally standardised and wide-spread SCPI command language (see *„5. The SCPI command language“*), which consists of pure textual commands. These can usually be entered manually via keyboard, in terminal programs or similar.

- *In order to control a device, i.e. changing values or status by switching the DC input/output on or off, it is required to activate remote control first, **by a command**. This doesn’t happen automatically when sending the very first command!*
- *Switching to remote control may be blocked by the device. When using SCPI language, the device never returns error messages automatically, but only upon request.*

## 4.7 About the register lists

Along with this programming guide, there are so-called register lists (one for each device series) included as PDF files. These give an overview of the remote programming features that are available for a certain device series when using binary communication protocols like ModBus. These lists differ in the number of available registers, i.e. commands. They are primarily made for ModBus protocol communication, but are also a reference when controlling a device via a field bus or accessing it in programming environments like LabView or MatLab, for example when trying to interpret values or to understand the function of a certain command.

The lists explain in compact format how the data in a binary message has to be interpreted or how a register (with CANopen resp. EtherCAT it is called "index") is specified. This will help the user to implement the device communication into custom software applications. Users who decide to work with SCPI command language usually do not need those lists. Later in this document, the SCPI commands are referenced in a separate chapter.

### 4.7.1 Column "Data type"

*The bytes of a ModBus message are read from left to right (big endian format), except for the 16 bit ModBus checksum where low byte and high byte are switched.*

| Data type | Length | |
|---|---|---|
| char | 1 Byte | Single byte, used for strings |
| uint(8) | 1 Byte | Single byte |
| uint(16) | 2 Bytes | Double byte, also called word or 16bit integer |
| uint(32) | 4 Bytes | Double word, also called long or 32bit integer |
| float | 4 Bytes | Floating point value according to IEEE745 standard |

### 4.7.2 Column "Access"

This column defines for every register whether the access is read only, write only or read/write.

R = Register is read only

W = Register is write only resp. would not return a reasonable value when read from

RW = Register can be read or written

*It applies generally: Writing to a register which allows W(rite) access is only possible during remote control!*

### 4.7.3 Column "Number of registers"

With ModBus, a register always has a length of 2 bytes or a multiple of 2 bytes. This column tells how many 2-byte values are used by the register. The values is always the half of the value in column “Data length in bytes”.

### 4.7.4 Column "Data"

This column tells additional information about the data which has to be written to or which can be read from the register. Two, four or more bytes can be interpreted in different ways, depending on data type and content.

### 4.7.5 Columns "Profibus/Profinet slot & index"

These columns (where available) are used by Profibus/Profinet users to link the registers in the register list via two values „index” and „slot” to data blocks (SFBs) in the PLC software. While „index“ is a direct parameter for the data block, the value „slot“ has to be used to find the address of a slot, which is variable, in order to get the parameter „ID“.

For more see *„6. Profibus & Profinet“*.

### 4.7.6 Column "EtherCAT SDO/PDO?"

This column is only available in register lists for those series which support the optionally available Anybus interface modules, here in particular the EtherCAT interface.

The column marks which of the ModBus registers can be accessed by the CANopen over Ethernet (CoE) protocol as being used by EtherCAT as communication protocol in form of indexes. Some of the marked registers are connected to PDOs, the rest is connected as SDOs. Devices supporting the EtherCAT interface contain a downloadable data object list. Which of the registers are connected to PDOs is described in section *„9. EtherCAT“*.

## 4.8 ModBus RTU in detail

*ModBus RTU can also be transferred using the optionally available standard Ethernet interface modules from the Anybus series or the built-in Ethernet port of some series. This is called "ModBus RTU over Ethernet", which is not "ModBus TCP". These built-in or standard Anybus Ethernet interface don't support the ModBus TCP frame.*

This protocol can be used with the USB-B interface (where available) and with most of the optionally available AnyBus modules. The addressed information when using ModBus protocol is also called register. This document uses the terms **address**, **register** or **register address**.

### 4.8.1 Message types

Basically, the message system distinguishes between **query messages**, **control messages** and **response messages**.

### 4.8.2 General

The devices also support the text based SCPI language, with automatic detection of the used protocol. When using ModBus, the first of every message has to be 0.

### 4.8.3 Functions

The second byte of a message contains a ModBus function code (FC, marked blue below), which determines whether the message is a READ or WRITE message. It also determines, whether one or multiple registers are accessed or only single bits.

The protocol, as described below, supports following ModBus functions (Date: 10-31-2016):

| Function | | Function name | | Description | Example of use |
|---|---|---|---|---|---|
| Hex | Dec | Long | Short | | |
| 0x01 | 1 | READ Coils | RSC, RC | Always reads 1 bit, but is returned as a full register with a 16 bit value. For example, the value 0xFF00 can mean "logic 1" or TRUE. At this point it is different from the ModBus standard and may cause problems with ModBus software tools. | Query the input / output condition |
| 0x03 | 3 | READ Holding Registers | RHR | Used to read n subsequent registers. Results in n*2 bytes of data in the response message. | Read the model name string (1-40 bytes) |
| 0x05 | 5 | WRITE Single Coil | WSC | Used to write n subsequent bits. If only 1 bit is defined for a register, other “coils” in the transmitted will be ignored. | Switch device to remote control. |
| 0x06 | 6 | WRITE Single Register | WSR | Used to write one register. | Set values (U, I, P etc.) |
| 0x10 | 16 | WRITE Multiple Registers | WMR | Used to write n subsequent registers. Cannot be used to write beyond the limits of a register block, for example when trying to write multiple set values (U, I, P) at once. | Write multiple values at once within a register block or write the so-called user text |

*The register list defines which of the above functions may be used with every register.*

### 4.8.4 Control messages (write)

When sending a status, a value, multiple values or a text, the data part of the message requires to define at least the target register and one value to write. The protocol checks the message only regarding the max. length of the register. After the data part, the checksum is expected. So in case the data part would only contain the minimum two bytes and thus the message would fulfil the protocol requirements for the selected function code, the checksum would be expected at the position of the 7th byte. If there were further data bytes at that position or zeros and the checksum would be at a different position in the message, the device would return an error. Hence the device will return an error, no matter if the telegram is too short or too long, because the checksum is wrong. For message examples see *„4.8.9. Examples of ModBus RTU messages“.*

**WRITE Single Register**

| Byte 0 | Byte 1 | Bytes 2+3 | Bytes 4+5 | Bytes 6+7 |
|---|---|---|---|---|
| Addr | FC | Start reg. | Data word | CRC |
| 0x00 | 0x06 | 0...65535 | Value to write | Checksum ModBus-CRC16 (1 |

**WRITE Multiple Registers**

| Byte 0 | Byte 1 | Bytes 2+3 | Bytes 4+5 | Byte 6 | Bytes 7-253 | Last 2 Bytes |
|---|---|---|---|---|---|---|
| Addr | FC | Start reg. | Number | Count | Data bytes | CRC |
| 0x00 | 0x10 | 0...65535 | 0...123 | 2*number | n values to write | Checksum ModBus-CRC16 (1 |

**WRITE Single Coil**

| Byte 0 | Byte 1 | Bytes 2+3 | Bytes 4+5 | Bytes 6+7 |
|---|---|---|---|---|
| Addr | FC | Register | Data word | CRC |
| 0x00 | 0x05 | 0...65535 | 0x0000 (FALSE) or 0xFF00 (TRUE) | Checksum ModBus-CRC16 (1 |

*Attention! The entire 16 bits of the data word repesent 1 coil only, for writing and reading.*

### 4.8.5 Query message

When querying something from the device, the response is expected to be immediately and will be of varying length, but always of the same construction. For the query, the start register and the number of registers or coils to read are required. The base of the ModBus data format is a register, a 16 bit integer value, means a group of two bytes. Thus, when querying one register with function READ Holding Registers, the device will return two bytes and when querying two registers it returns 4 bytes etc. For READ Coils, the response will always be two bytes.

For message examples see *„4.8.9. Examples of ModBus RTU messages“.*

**READ Holding Registers**

| Byte 0 | Byte 1 | Bytes 2+3 | Bytes 4+5 | Bytes 6+7 |
|---|---|---|---|---|
| Addr | FC | Start reg. | Number | CRC |
| 0x00 | 0x03 | 0...65535 | Number of regs to read (1...125) | Checksum ModBus-CRC16 (1 |

**READ Coils**

| Byte 0 | Byte 1 | Bytes 2+3 | Bytes 4+5 | Bytes 6+7 |
|---|---|---|---|---|
| Addr | FC | Start reg. | Number | CRC |
| 0x00 | 0x01 | 0...65535 | Must always be 1 | Checksum ModBus-CRC16 (1 |

*Attention! This is not according to the standard here: reading a coil always returns a 16 bit value, which doesn't contain 16 coils, but all 16 bits represent 1 coil.*

(1 See *„4.8.7. The ModBus checksum“*

### 4.8.6 Response message (read)

A response from the device is usually expected after a query or if something has been set and the device confirms the execution.

**Expected response for WRITE Single Register:**

| Byte 0 | Byte 1 | Bytes 2+3 | Bytes 4+5 | Bytes 6+7 |
|---|---|---|---|---|
| Addr | FC | Start reg. | Data | CRC |
| 0x00 | 0x06 | 0...65535 | Written value echoed | Checksum ModBus-CRC16 (1 |

**Expected response for WRITE Single Coil:**

| Byte 0 | Byte 1 | Bytes 2+3 | Bytes 4+5 | Bytes 6+7 |
|---|---|---|---|---|
| Addr | FC | Start reg. | Data | CRC |
| 0x00 | 0x05 | 0...65535 | Written value echoed | Checksum ModBus-CRC16 (1 |

**Expected response for WRITE Multiple Registers:**

| Byte 0 | Byte 1 | Bytes 2+3 | Bytes 4+5 | Bytes 6+7 |
|---|---|---|---|---|
| Addr | FC | Start reg. | Data | CRC |
| 0x00 | 0x10 | 0...65535 | Number of written registers | Checksum ModBus-CRC16 (1 |

**Expected response for READ Holding Registers:**

| Byte 0 | Byte 1 | Byte 2 | Bytes 3-253 | Letzte 2 Bytes |
|---|---|---|---|---|
| Addr | FC | Data length in bytes | Data | CRC |
| 0x00 | 0x03 | 2...250 | Queried registers content | Checksum ModBus-CRC16 (1 |

**Expected response for READ Coils:**

| Byte 0 | Byte 1 | Byte 2 | Bytes 3+4 | Bytes 5+6 |
|---|---|---|---|---|
| Addr | FC | Data length in bytes | Data | CRC |
| 0x00 | 0x01 | 2 | Queried bit as 1 register (always 16 coils) | Checksum ModBus-CRC16 (1 |

**Unexpected response (communication error):**

| Byte 0 | Byte 1 | Byte 2 | Letzte 2 Bytes |
|---|---|---|---|
| Addr | FC | | CRC |
| 0x00 | Function code + 0x80 | Error code | Checksum ModBus-CRC16 (1 |

*A communication error can have several reasons, like a wrong checksum or when attempting to switch a device to remote control that has been set to "Local" or if it is already remotely controlled by another interface. See the communication error code list in „4.8.8. Communication errors".*

### 4.8.7 The ModBus checksum

The checksum at the end of ModBus RTU messages is a 16 bit checksum, but is not calculated as the usual CRC16 checksum. Furthermore, **the byte order** of the checksum in the message i**s reversed**. Information about ModBus CRC16 and source code for implementation and calculation are available on the Internet, for example here:

http://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf, page 39.

(1 See *„4.8.7. The ModBus checksum"*

### 4.8.8 Communication errors

Communication errors are only related to digital communication with the device. Other alarms or errors of any kind which can be generated and indicated by the device must not be mixed up with these.

The device will return unexpected error messages in case the previously sent message is in wrong format or if the function can not be executed by some reason. For example, when trying to write a set value with WRITE SINGLE REGISTER while the device is not in remote control. Then the message is not accepted and the device will return an error message instead of a confirmation message. The message format can be wrong if the checksum is bad or if you try to read a bit with function READ Holding Registers instead of READ COILS.

In case of an error, the response message contains the original function code + 0x80, in order to identify the response as error message.

Overview of function codes in error messages:

| FC error | Belongs to |
|---|---|
| 0x81 | READ COILS |
| 0x83 | READ HOLDING REGISTERS |
| 0x85 | WRITE SINGLE COIL |
| 0x86 | WRITE SINGLE REGISTER |
| 0x90 | WRITE MULTIPLE REGISTERS |

Overview of the communication error codes which can be returned by the device:

| Code | | Error | Explanation |
|---|---|---|---|
| 0x01 | 1 | Wrong function code | The function code in byte 1 of the ModBus message is not supported. See *„4.8.3. Functions“* for supported codes. The error also occurs when trying to read or write a register with a function code for which the register is not defined. |
| 0x02 | 2 | Invalid address | The register address you were trying to access with read or write is not defined for your device. Every device series may have a different number of registers. Refer to the separate ModBus register list of the series your device belongs to. |
| 0x03 | 3 | Wrong data or data length | The length of data in the message is wrong or the data itself. For example, a set value always requires two bytes of data. If the data part of the message would be one byte only or three bytes, then the data length would be wrong. Otherwise, when sending a set value of, for example, 0xE000 to a register for which the maximum value is defined as 0xCCCC, this would be wrong data. |
| 0x04 | 4 | Execution | Command could not be executed, depends on the situation |
| 0x05 | 5 | CRC | The CRC16 checksum at the end of the ModBus RTU message is wrong or has been transmitted in wrong byte order (high byte first instead of low byte) |
| 0x07 | 7 | Access denied | Access to a certain register is not allowed or read only while trying to write, or vice versa. The error also occurs when trying to write to a writable address while the device is not in remote control or in remote control from a different interface |
| 0x17 | 23 | Device in local | Indicates, that write access to the device is blocked by local condition, so only read access is possible as long as local is activated. |

**An example:** You try to switch the device to remote in order to control it by a PC, but instead of an echo of your message it returns something like this: 0x00 0x85 0x07 0x52 0x92. This is an error message. The position of the function code contains the value 0x85. According to the first table above, this is related to the function WRITE SINGLE COIL. The error code in the message is 0x7 which means, according to the second table above, the device has denied the access. This can have different reasons, for example that the device is already in remote control via a different interface.

### 4.8.9 Examples of ModBus RTU messages

*The examples can also be used for ModBus TCP, but they need to be extended by the required ModBus TCP header and stripped from the unnecessary checksum.*

#### 4.8.9.1 Writing a set value

*Set values are adjustable limits for the physical values Current, Voltage, Power and Resistance (where available). The can only be written to a device, if it has been switched to remote control before via a digital interface.*

Example: You want to set the current to 50%. According to the register lists, the „Set current value" is at address 501 (0x1F5) and assigned function is WRITE Single Register. Expecting the device to already be in remote control mode, the message to build then has to be like this:

Message to send:

| Addr | FC | Start | Data | CRC |
|---|---|---|---|---|
| 0x00 | 0x06 | 0x01F5 | 0x6666 | 0x325F |

▶ Expected response:

| Addr | FC | Start | Data | CRC |
|---|---|---|---|---|
| 0x00 | 0x06 | 0x01F5 | 0x6666 | 0x325F |

In this case, the device is expected to return an echo of your message, indicating successful execution of the command. The display of the device should now show 50% of what's the maximum current of your device. For a power supply or electronic load with 510 A nominal current, it should show 255.0 A, or for a model with 170 A nominal current it should show 85 A.

#### 4.8.9.2 Query all actual values at once

The device holds three readable actual values of voltage, current and power. Electronic loads feature an additional actual resistance value in their displays, which can not be read via interface, but is calculated from the actual voltage and current. Hence the user can calculate the actual resistance himself.

Actual values can be queried separately or all at once. The advantage of a combined query is, that you gain a snapshot of the most recent actual values of the DC input or output. When querying separately, values may have changed already when sending the next query.

According to the register list, the actual values start from register 507. Three registers shall be read:

Message to send:

| Addr | FC | Start | Data | CRC |
|---|---|---|---|---|
| 0x00 | 0x03 | 0x01FB | 0x0003 | 0x7417 |

▶

Possible response:

| Addr | FC | Len | Data | CRC |
|---|---|---|---|---|
| 0x00 | 0x03 | 0x06 | 0x2620 0x0C9B 0x091B | 0x9EC0 |

#### 4.8.9.3 Read the nominal voltage of a device

The nominal voltage, like the other nominal values of current, power or resistance, is an important value to read from a device. They're all referenced for translating set values and actual values. It is recommended to read them from the device right after opening the digital communication line, unless the software shall not be universal.

According to the register list, the nominal voltage is a 4-byte float value in register 121.

Query message:

| Addr | FC | Start | No. | CRC |
|---|---|---|---|---|
| 0x00 | 0x03 | 0x0079 | 0x0002 | 0x1403 |

▶ Possible response:

| Addr | FC | Len | Data | CRC |
|---|---|---|---|---|
| 0x00 | 0x03 | 0x04 | 0x42A00000 | 0xFEA9 |

Also see *4.8.6*. The response contains a float value according to IEEE754 format, which translates to 80.0.

#### 4.8.9.4 Read device status

All device report their device status in a register, for series ELR 9000 and PSI 9000 this is register 505. Devices of other series can have a different address for this status register.

Query message:

| Addr | FC | Start | No. | CRC |
|---|---|---|---|---|
| 0x00 | 0x03 | 0x1F9 | 0x0002 | 0x1417 |

▶ Possible response:

| Addr | FC | Len | Data | CRC |
|---|---|---|---|---|
| 0x00 | 0x03 | 0x04 | 0x00000483 | 0xA992 |

Also see *4.8.6*. The response contains the value 0x483 which states that the device is in remote control via the USB port, that the DC input/output is switched on and that CC (constant current) mode is active.

#### 4.8.9.5 Switch to remote control or back to manual control

Before you can control a device from remote, it is required to switch it to remote control. This is done by sending a certain command.

| | |
|---|---|
|  | *The device will never switch to remote control automatically and can not be remote controlled with being in this condition. Reading status and values is but always possible.* |

| | |
|---|---|
|  | *The device will never exit remote control automatically, unless it is switched off or the AC supply is otherwise interrupted. Remote control can be left by a certain command. It then switches back to manual control.* |

Switching to remote control may be inhibited by several circumstances and is usually indicated by an error message:

- Condition „**Local**“ is active (check the display on the front of your device or read the device status), which will prevent any remote control
- The device is already remotely controlled by another interface
- The device is in setup mode, means the user has accessed the setup menu and not left it yet

**► How to switch a device to remote control:**

1. If you are using the ModBus RTU protocol, you need to create and send a message according to the description above, for example 00 05 01 92 FF 00 2D FA
2. Once the switchover to remote control has been successful, the device will usually indicate the new condition in the display or with a LED, as well as it echoes the message as a confirmation

In case switching to remote control would be denied by the device, because option “Allow remote control = No” is set (example from ELR 9000 series, other series may differ), then the device will return an error message like 00 85 17 53 5E. According to ModBus specification, this is error 0x85 with error code 0x17.

Leaving remote control can be done in two ways: using the dedicated command or by switching the device to “**Local**” condition. We will consider the first option, because this is about programming.

**► How to exit remote control:**

1. If you are using the ModBus RTU protocol, you need to build and send a message according to the description above, for example 00 05 01 92 00 00 6C 0A.

## 4.9 ModBus TCP in detail

ModBus TCP is based upon ModBus RTU and this section is only about the differences. The core of a ModBus TCP message is ModBus RTU. Refer to *„4.8. ModBus RTU in detail“* for more information. For ModBus TCP applies:

- It requires an additional 6 bytes long MBAP (ModBus Application Protocol) header for the ModBus RTU message
- The checksum of the ModBus RTU message is omitted
- It can only be used via port 502 and only with ModBus TCP interfaces. The port is reserved and must thus not be adjusted in the setup MENU of your device. That port is for standard socket connections.

As a result, Modus TCP messages are always 4 bytes longer than the ModBus RTU messages (byte 0 of the RTU message is replaced by the actual 7th byte of the additional MBAP header).

The MBAP header is specified like this:

| Bytes | Meaning | Explanation |
|---|---|---|
| 0 + 1 | Transaction identifier | This identifies the message. It is copied by the device in the response and is used to identify a certain message in a pool of incoming transmissions if multiple device are communicating with the PC and the response is not immediately. The identifier is an arbitrary value between 0 and 65535. |
| 2 + 3 | Protocol identifier | Here always 0 = ModBus protocol |
| 4 + 5 | Length | Number of remaining bytes in the message, i.e. the length of the ModBus RTU core message minus 2. |

### 4.9.1 Example for a ModBus TCP message

The example for READ Holding Registers from *„4.8.9.3. Read the nominal voltage of a device“*, extended by the MBAP header (arbitrary transaction identifier 0x4711 used):

Query message:

| MBAP header | Addr | FC | Start | Data |
|---|---|---|---|---|
| 0x4711 0x0000 0x0006 | 0x00 | 0x03 | 0x0079 | 0x0002 |

Possible response:

| MBAP header | Addr | FC | Length | Data |
|---|---|---|---|---|
| 0x4711 0x0000 0x0007 | 0x00 | 0x03 | 0x04 | 0x42A00000 |

The example is a query for reading the device’s nominal voltage. The response contains a floating point value in “Data”, which translates to 80(V).

## 4.10 Detailed explanation about specific registers

For the abbreviations of the devices series see *„1.1.2. Validity“*.

Many of the commands resp. register related options are self-explaining, but not all of them. Some of the not self-explaining ones will be handled below.

### 4.10.1 Register 171

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PS5 | PSE | DT |
|---|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

This allows to write and read an arbitrary string of up 40 characters, which can be used to uniquely identify a device alternatively to the serial number. It can be especially useful when multiple units of the same model are in remote control by one software and need to be distinguished.

### 4.10.2 Register 408

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PS5 | PSE | DT |
|---|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | – | – | ✓ | ✓ |

This register defines the condition of the DC output resp. input of the device after the device has been powered. Register 408 is connected to the MENU setting “Input after power on” (electronic loads) resp. ”Output after power on” (power supplies). Also see the device operating guide.

### 4.10.3 Register 411

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PS5 | PSE | DT |
|---|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Described for SCPI in *„5.4.16 Commands for alarm management“ on page 61.*

When using ModBus, this register is used to reset alarm bits as represented in the device status (register 505, see below). Until these are not reset, which is considered as an acknowledgement, the bits from previously occurred alarms remain set, even if the alarms are gone since long. Alarms which are still present while register 411 is used to reset the alarm bits will of course be excluded from resetting. There is an exception, the device alarm OT (bit 19, overtemperature). This will be cleared automatically once the unit has cooled down. After resetting the alarm bits, device alarm can only be read in form of alarm counter (registers 520 - 524).

### 4.10.4 Registers 500-503 (Set values)

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PS5 | PSE | DT |
|---|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

These are the most important registers to work with, because they define the DC output/input values of voltage, current, power and resistance (where featured). With ModBus, any set value is transmitted as per cent value of the nominal device values (0...100%), whereas for SCPI real values are used.

Generally, before you can use R mode with devices where internal resistance is featured, it has to be activated (register 409), else the set value is ignored.

For power supply devices of series PSI 9000 (as from 2014) and the PV function, which is only featured there, the set value for current (register 501) is interpreted as irradiation value, as long as the device is in "PV mode". Means, while the function is running, this register does not define the current limit for your device, but a parameter called irradiation, which is commonly used in solar panel simulation. In manual operation, irradiation can be adjusted in 1% steps between 0% and 100%. With the set value of current it is also 0-100%, according to definition of register 501, but with a significantly higher resolution.

### 4.10.5 Register 505 (Device status)

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PS5 | PSE | DT |
|---|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Another important register, as it represents the device condition in one 32 bit value (ModBus). Some bits are grouped and have to be interpreted like that. According to the register list, bits 0-4 of registers 505 are a group that represents the so-called control location (see *„3.2. Control locations“*). By reading this register you can furthermore detect if the device is already in remote control to see if command “Remote mode = on” was executed by the device.

With SCPI, some but not all of these 32 bits of this register are represented in the status registers "Questionable" and "Operation". See *„5.4.2. Status registers“.*

#### 4.10.5.1 When running master-slave

During master-slave operation (where featured), the status register uses bit 29 ("MSS") to indicate the so-called master-slave safety mode, which is activated every time the master detects any problem in the communication with the slave(s), which can occur due to a connection failure or heavy electrical interferences. The master unit will then set this bit and switch off all DC outputs/input of the slaves being still online. Offline slaves will put themselves into a similar state and switch off DC.

After removal of the problem cause, the MS system has to be re-initialised, which also clears the bit.

### 4.10.6 Registers 650 - 662 (Master-slave configuration)

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PS5 | PSE | DT |
|---|---|---|---|---|---|---|---|
| ✓ | – | – | ✓ | – | – | ✓ | – |

This block of registers is used to configure the master-slave operation mode (short: MS) the same as you can do it in the setup MENU of your device. Refer to the device's operating guide about how the MS works and what do to in preparation of its remote control. For remote control of a MS system, it is expected to be fully wired. Before MS operation, slave units can be configured remotely, but during MS operation they can only be monitored, if required. It is, however, recommend to only control the master unit. Configuration and activation of MS operation can also be done manually and remote control can be taken over later after the master has initialised the system.

With the MS system not being set up yet, these registers have to be used in a certain order on any unit:

1. Switch to remote control with register 402.
2. Activate MS operation mode with register 653.
3. Select with register 650 whether the unit you are configuring will be Master or Slave.
4. Define the slave address (1-15) with register 651, if the unit has been selected to be slave.

Further steps, only to be performed on the master unit:

5. Initialise the MS system with register 654.
6. (with electronic loads only and also only required when running two-quadrants operation)
   Set the unit to be Share bus slave (else it would be master) with register 652.
7. Optional: check with register 655, whether the initialisation has been successful.
8. Optional: Query the number of initialised units with register 662 --> in case the number does not match the units you want to use in the MS system, check the settings of all units and the cabling and repeat the initialisation.
9. Optional: read the nominal values (registers 656-660) of the previously initialised MS system to be used as value translation reference while you use the MS.
10. Optional: configure alarm thresholds, event thresholds and set value limits.

During MS operation, the remotely controlled master unit can be accessed like a single unit, with a few exceptions (see device manual). Set values and actual values are always per cent values related to certain nominal values. Access to those registers is described in the other sections.

### 4.10.7 Registers 850 - 6695 (Function generator)

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PS5 | PSE | DT |
|---|---|---|---|---|---|---|---|
| ✓ | – | – | ✓ | – | – | – | ✓ |

The integrated function generator is a complex feature. It is configured and loaded with a lot of registers. Before you can run a function, setup is required every time and in a certain order.

First of all, you need to decide which one of the two basic function generators you want to use: arbitrary or XY.

All further steps depend on this selection.

*All function generator settings and loaded data (sequences, XY table) are not stored inside the device and have to be loaded into the device every time before you can use the function generator. These data and settings are completely separate from what you can setup and define for the function generator manually when using the control panel and touch display.*

#### 4.10.7.1 Procedure for the arbitrary generator

This generator is used to create wave functions like sine, square, triangle or trapezoidal.

**Step 1:**

Select, whether to apply the function to the voltage U (register 851) or the current I (register 852). Before you haven't made this selection, the device can not accept sequence data, because the sequence data is run through a plausibility check against the device's nominal values.

**Step 2:**

Define start sequence (register 859), end sequence (register 860) and number of cycles of that sequence block to repeat (register 861) and submit the settings with register 856, which is not required anymore since KE firmware 2.01.

**Step 3:**

Load the sequence data for the sequences (x out of 100) you want to use (registers 900-2484, 8 values per sequence).

**Step 3.1:**

When using SCPI, it is required to submit the sequence data with command [SOURce:]FUNCtion:GENerator:WAVE:SUBmit. This causes the device to transfer the sequence data to the target components (internal controller, HMI). In the contrary, if ModBus has been used to transmit sequence data, which happens in blocks, no extra submit is required.

**Step 4:**

Set global limits for voltage (register 500 or VOLT command), current (register 501 or CURR command) and power (register 502 or POW command).

**Step 5:**

Control the function generator with start/stop (register 850).

**Step 6:**

When finished, leave the function generator by deselecting your former selection of either U (register 851) or I (register 852) again.

#### 4.10.7.2 Programming example for the arbitrary generator

Before you can configure the arbitrary generator for a ramp it is necessary to think about the best way to achieve the ramp generation. It is important to keep in mind that the arbitrary generator stops at the end of the function run, unless you set the repetition to infinite. After a stop, the DC input/output remains switched on. In case of a ramp, this is wanted, because the end value shall usually remain set for time x. However, the device will go to static mode again, setting the static set values of U, I and P. The static values also apply for the period before the function run and for situations when the DC output/input is already switched on.

The stop action and the static values are thus a little problematic for the ramp function. Why? Supposed, you wanted to have a power supply generate a ramp starting from 0 V. The static value for U (voltage) would then be set to 0. But after the function stop, the device would also set 0 V and the voltage would drop from whatever value has been set during the function run. Conclusion: the static value of voltage has to be part of the function.

In order to achieve this, the function has to consist of two parts: one for the rising or falling ramp and the other for the static value. This can be done using two sequences of the arbitrary generator.

Assumption: the ramp shall start from 0 V and rise to 50 V within 6 seconds. The end voltage shall remain constant for 3 minutes (the time can be varied at will). Sequences 1 and 2 will be used. Remote control is already active, we only need to configure the sequences. Since the ramp will make the voltage rise linearly, using only the DC part of a sequence, the parameters for the AC part (indexes 0 - 4) should be set to zero in order to avoid remainders of wrongly set AC parameters which could disturb the correct wave generation.

The first step is to **activate function generator mode**, in this case we select arbitrary generator for U:

| Addr | FC | Start | Data | CRC |
|---|---|---|---|---|
| 0x00 | 0x05 | 0x0353 | 0xFF00 | 0x7DBE |

Next step is to create the **ModBus message to configure sequence 1**, **the rising ramp**. According to the register list start register 900 (WMR, function code 0x10) is assigned to sequence 1. Because the data part would not the width of this document's page size, the 8 float values are below each other:

| Addr | FC | Start | Data | CRC | Description |
|---|---|---|---|---|---|
| 0x00 | 0x10 | 0x0384 | 0x00000000 | | Start value of AC part: 0 V |
| | | | 0x00000000 | | End value of AC part: 0 V |
| | | | 0x00000000 | | Start frequency of AC part: 0 Hz |
| | | | 0x00000000 | | End frequency of AC part: 0 Hz |
| | | | 0x00000000 | | Start angle of AC part: 0° |
| | | | 0x00000000 | | Start value of DC part: 0V |
| | | | 0x42480000 | | Start value of DC part: 50V |
| | | | 0x4ab71b00 | 0x5A14 | Rise time in µs: 6,000,000 (6 seconds) |

After this, the **ModBus message to configure sequence 2, the static voltage** would be next. Start register here is 916:

| Addr | FC | Start | Data | CRC | Description |
|---|---|---|---|---|---|
| 0x00 | 0x10 | 0x0384 | 0x00000000 | | Start value of AC part: 0 V |
| | | | 0x00000000 | | End value of AC part: 0 V |
| | | | 0x00000000 | | Start frequency of AC part: 0 Hz |
| | | | 0x00000000 | | End frequency of AC part: 0 Hz |
| | | | 0x00000000 | | Start angle of AC part: 0° |
| | | | 0x42480000 | | Start value of DC part: 50V |
| | | | 0x42480000 | | Start value of DC part: 50V |
| | | | 0x4d2ba950 | 0x688B | Sequenz time in µs: 180,000,000 (180 seconds = 3 minutes) |

And as last step, configuration of the arbitrary generator itself:

| Addr | FC | Start | Data | CRC | Description |
|---|---|---|---|---|---|
| 0x00 | 0x06 | 0x035B | 0x0001 | 0x384C | Register 859, WSR, Start sequence: 1 |
| 0x00 | 0x06 | 0x035C | 0x0002 | 0xC98C | Register 860, WSR, End sequence: 1 |
| 0x00 | 0x06 | 0x035D | 0x0001 | 0xD84D | Register 859, WSR, Sequence cycles: 1 |

Now the entire function setup is done and the function can be started. If the DC output/input of your device would still be off when starting the function, it will automatically switch on. Alternatively, you could switch it on separately with the corresponding command and before actually running the function. But it is not necessary here, because the voltage shall start to rise from 0 V. In other situations where the starting level is not zero, it would be required to switch on the DC output/input first and wait for the voltage to settle.

For the number of sequence cycles 1 is sufficient, but it can be changed at will. The the whole function would be repeated after 3 minutes and 6 seconds. The voltage, when using a power supply, would not instantly drop to 0 V at the end of the first function run and before the second one starts. It depends on the load how long the voltage takes to sink and the ramp, when being graphically recorded on an oscilloscope, could look different than expected. This could be circumvented by added a third sequence which only uses some time for the voltage to go down.

| Addr | FC | Start | Data | CRC | Description |
|---|---|---|---|---|---|
| 0x00 | 0x05 | 0x0352 | 0xFF00 | 0x2C7E | Register 850, WSC, Run function |

#### 4.10.7.3 Procedure for the XY generator

**Step 1:**

Select, whether the function UI (including FC function, register 854) or IU (register 855) or PV (PSI 9000 series only, register 426) shall be applied to the DC input/output of your device. Before you haven't made this selection, the device can not accept no XY table data, because that data is run through a plausibility check against the device's nominal values.

**Step 2:**

Load XY table data in blocks of 16 values (registers 2600 - 6680), max. 4096 values. If less than 4096 are loaded, the rest will be zero and transferred unchanged to the internal processing unit

**Step 3:**

Submit table data (register 858).

**Step 4:**

Set static values which are not affected by the table

UI function: current (register 501 or CURR command) and power (register 502 or POW command)

IU function: voltage (register 500 or VOLT command) and power (register 502 or POW command)

**Step 5:**

Run the function generator with start/stop (register 850) resp., if still not done, switch on the DC input/output of your device (register 405). For PV mode and function, you may also want to control irradiation while the function is running. This is done by sending set values to register 501 (current), where 100% corresponds to a factor of 1 and 0% to a factor of 0. This factor is multiplied to the simulated current value $I_{MPP}$, which you have to enter in the HMI as a parameter of the simulated solar panel or which is defined by the loaded table.

**Step 6:**

Exit the function generator by deselecting your former selection of either UI (register 854), IU (register 855) or PV (PSI 9000 only, register 426, else error) again.

### 4.10.8 Registers 850 - 1692 (Sequence generator)

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PS5 | PSE | DT |
|---|---|---|---|---|---|---|---|
| – | ✓ | – | – | – | – | – | – |

The so-called sequence generator of ELR/ELM 5000 series is a simplified version of the arbitrary generator of other series, thus using some of the same registers. According to the register list for ELR 5000 series, a block of registers between 850 and 1692 is used to configure the 100 sequence points and to control the generator.

### 4.10.9 Registers 9000 - 9006 (Adjustment limits)

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PS5 | PSE | DT |
|---|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | – | – | ✓ | ✓ |

For SCPI protocl, this is explained in *„5.4.8. Commands for adjustment limits“*. ModBus users may also read that section for the general handling of these settings. Apart from that, setting these parameters is like setting a set value (U, I, P, R).

### 4.10.10 Registers from 10007

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PS5 | PSE | DT |
|---|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | – | ✓ | ✓ |

Those registers can be used to remotely configure the optionally available, digital Anybus interface modules, which can be installed in a dedicated slot (where available) or an integrated Ethernet port. The registers are connected to the corresponding settings in the device's setup menu.

Contrary to manual control, the settings can even be configured via these registers while the interface module is not installed.

# 5. The SCPI command language

| | |
|---|---|
| | *The SCPI standard does not define the use of a termination character and thus series supporting SCPI do not expect one. Exception: devices with option 3W (GPIB interface). See „5.3.4. Termination character“.*<br>*With other interfaces, if commands are fragmented (TCP) or sent as single characters (serial port), they can't be processed correctly and communication errors are put into the error queue, which is not returned automatically. Also see section 3.6.* |

## 5.1 Format of set values

In the SCPI command language, everything is text and values are **real values**, usually without unit. It means, if you wanted to set a current of, for example, 177.5 A, you would use the simple command **CURR 177.5** (or with unit: CURR 177.5 A). Below you will find more detailed information about the available commands and the syntax.

| | |
|---|---|
| | *All set values (U, I, P, R), which have dedicated single command and which you can send to the device during remote control, are not only limited by the maximum, i.e. nominal values of your particular device model, but additionally limited by those adjustment limits “Limits” that you can define for manual adjustment!* |

## 5.2 Examples for a first start

### 5.2.1 Ping

It’s always recommended to ping a device first, in order to test if it responds at all. With SCPI, this is usually done by querying the identification string:

| Protocol | Command |
|---|---|
| SCPI | *IDN? |

As an immediate response, the device might send, for example:

| Protocol | Response |
|---|---|
| SCPI | EA Elektro-Automatik GmbH&Co.KG, ELR 9080-170, 1237680002, V1.14 13.05.2013 V1.14 03.05.2013 V1.3.1 |

### 5.2.2 Switch to remote control or back to manual control

Before you can remotely control a device, you need to switch it to remote control by sending the dedicated command. Also see the SCPI command description below.

| | |
|---|---|
| | *The device will never switch to remote control automatically and can not be remotely controlled without being in this condition. Reading status and values is but always possible.* |

| | |
|---|---|
| | *The device will never exit remote control automatically, unless it is switched off or the AC supply is otherwise interrupted. Remote control can be left by a certain command. It then switches back to manual control.* |

Switching to remote control may be inhibited by several circumstances and is usually indicated by an error message:

- Condition „**Local**“ is active (check the display or control panel on the front of your device), which will prevent any remote control
- The device is already remotely controlled by another interface
- The device in setup mode, means the user has accessed the setup menu and not left it yet

**► How to switch a device to remote control:**

1. If you are using SCPI command language, send a text command (the space is required):
   SYST:LOCK␣1 or SYST:LOCK␣ON

Leaving remote control can be done in two ways: using the dedicated command or by switching the device to “**Local**” condition. We will consider the first option, because this is about programming.

**► How to exit remote control:**

1. If you are using SCPI command language, send a text command (the space is required):
   SYST:LOCK␣0 or SYST:LOCK␣OFF

## 5.3 Syntax

Specification according to „1999 SCPI Command reference”.

Following syntax formats can occur in commands and/or responses:

| | | |
|---|---|---|
| Values | This numeric value corresponds to the value in the display of the device and depends on the nominal values of the device. Rules:<br>- The value must be sent after the command and separated by a space<br>- Instead of a numeric value you can also use: | |
| | MIN | corresponds to the minimum value of the parameter |
| | MAX | corresponds to the maximum value of the parameter |
| <NR1> | Numeric values without decimal place | |
| <NR2> | Numeric values with decimal place | |
| <NR3> | Numeric values with decimal place and exponent | |
| <NRf> | <NR1> or <NR2> or <NR3> | |
| Unit | V (Volt), A (Ampere), W (Watt), OHM, s (Seconds) | |
| <CHAR> | 0..255: Decimal value | |
| <+INT> | 0..32768: Positive integer value (output from device) | |
| <B0> | 1 or ON: Function is/will be activated | |
| | 0 or OFF: Function is/will be deactivated | |
| <B1> | NONE: manual operation active, switching to remote control possible | |
| | LOCal: local (manual) operation only, reading of data possible | |
| | REMote: device is in remote control | |
| <ERR> | Error with number (-800 bis 399) and description | |
| <SRD> | String data, various formats:<br>- IP address as number string with dots as separator, for example „192.168.0.2“<br>- Key words, for example AUTO or OFF | |
| <Time> | [s]s.s[s][s][s][s][s][s] / Default format is seconds (s.s) | |
| ; | The semicolon is used separate multiple commands within one message | |
| : | The colon separates the SCPI keywords (main system, subsystems) | |
| [ ] | Lowercase letters and the content of square brackets are optional | |
| ? | The question mark identifies a message as query. A query can be combined with a control message (command concatenation). Note, that it is required to wait for the response of the query before the next control message can be sent. | |
| -> | Response from device | |

### 5.3.1 Coupled commands

It is possible to couple, i.e. concatenate up to 5 commands to one message and send them to the device. The commands have to be separated by a semicolon (;). Example:

VOLT 80;CURR 20;POW 3kW

The commands are processed from left to right in the string, so the order of commands is important to achieve correct results. When querying multiple values or parameters at once, the returned string is also in coupled format, with the queried returns separated by semicolons.

*Returned messages can be up to 512 characters. When querying coupled commands where the response would exceed 512 characters, like when querying 5x *IDN?, nothing is returned.*

### 5.3.2 Upper and lower case

SCPI uses upper case commands by default.

### 5.3.3 Long form and short form

SCPI commands have a long and a short form. The short form (eg. SOUR) and the long form (eg. SOURCE) can be used arbitrarily. To distinguish both forms, the commands as described in the following sections are written partly in upper case (indicates short form), partly in lower case letters (indicates long form).

### 5.3.4 Termination character

In general, SCPI does not require a termination character when using it via USB, Ethernet or other interfaces, except for GPIB. The IEEE standard defines the use of a termination character. Hence devices with installed option 3W, i. e. GPIB interface, require to send this character or else a but timeout error will occur.

Supported termination character(s): **0xA** (LF, line feed)

## 5.4 Command groups

Command groups are related to specific features of a device. Not all series feature the same number of commands, which is due to a different number of features within a series. Every command below will indicate to which series it is compatible. Example:

| ELR9 | PS9 | PSI9 | PSI5 |
|---|---|---|---|
| ✓ | – | ✓ | – |

For the abbreviations see *„1.1.2. Validity“*, whereas

✓ means, the command or the command group is supported by the device series.

– means, the command or the command group is not supported by the device series.

### 5.4.1 Standard IEEE commands

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

In relation to the old interface standards GPIB and IEEE 488, some of the standard commands have been implemented. They are supported in all devices which feature SCPI command language.

#### 5.4.1.1 *IDN?

Returns the device identification string, which contains following information, separated by commas:

1. Manufacturer
2. Model name
3. Serial number
4. Firmware version(s) (in case there are several, these are separated by a space)
5. User text (arbitrary user-definable text, as definable with SYST:CONFIG:USER:TEXT)

#### 5.4.1.2 *RST

When sent, this will set the device to a defined state, except remote control is denied by the device:

1. Switch to remote control (same as SYST:LOCK 1)
2. Set DC input/output to off
3. Clear alarm buffer
4. Clear status registers to default condition (QUEStionable Event, OPERation Event, STB)

#### 5.4.1.3 *STB?

Reads the STatus Byte register. The signal run of the various device conditions and events is illustrated in the register model below. The STB bits in particular:

Bit 0: not used

Bit 1: not used

Bit 2: *err*, Error Queue --> one or several error in the error buffer. By reading the error buffer it is flushed and the bit *err* is reset

Bit 3: *ques*, Questionable Status Register is active (one or several events have occurred)

Bit 4: not used

Bit 5: not used

Bit 6: not used

Bit 7: *oper*, Operation Status Register is active (one or several events have occurred)

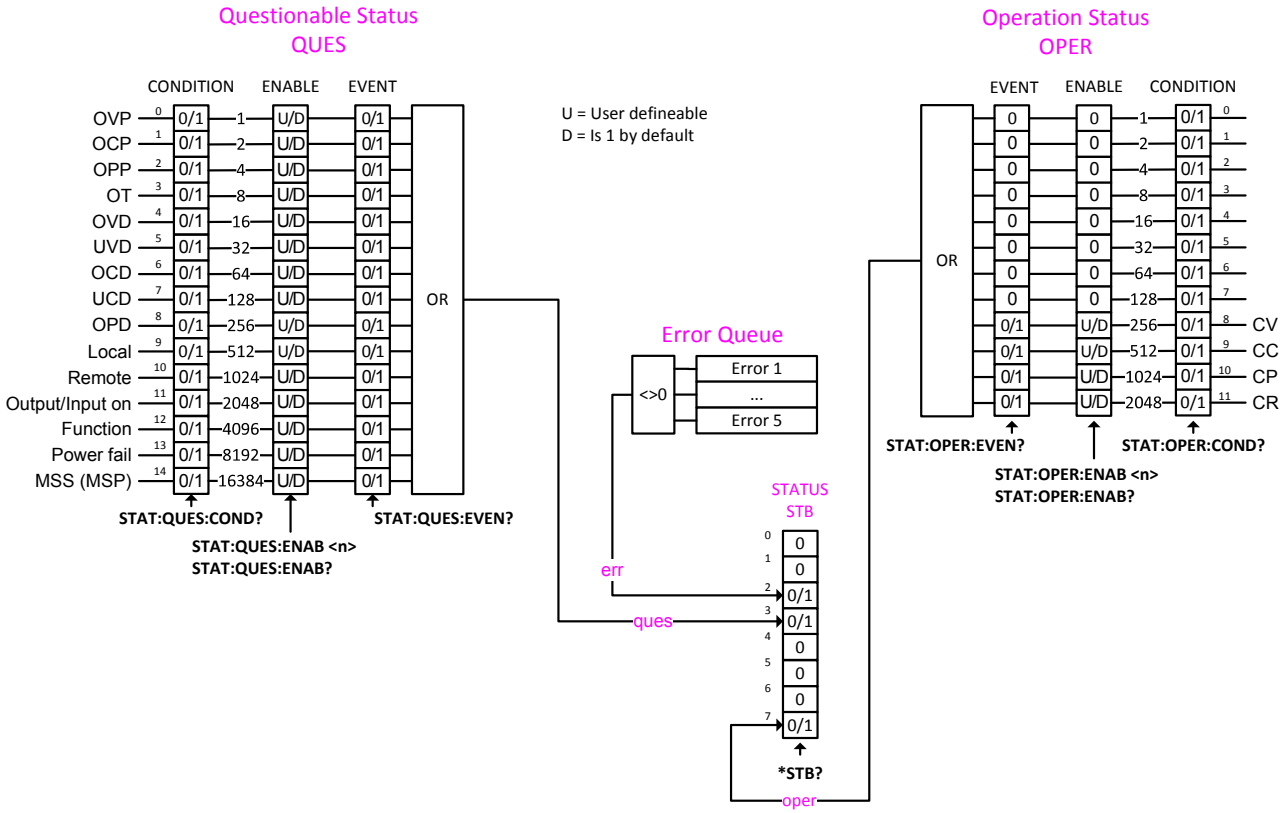
### 5.4.2 Status registers

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Not all device conditions and alarms can be read with dedicated SCPI commands. As an alternative, the remaining device-related information are grouped in status registers. Using regular polling, the status byte (STB) can be a starting point for reading the device status. It tells what status register has recorded at least one event. Apart from that, the other status registers could also be polled directly. The difference then would be, that the user would have to find out, what bits in the register have changed, by comparing the most recent value with an older value. The bits in the status byte register will do that job for you. If they remain 0, nothing has happened.

Once a bit in the STB signalises, that there was an event recorded in QUES, OPER or ESR register, you could read the corresponding event register of OPER and QUES, in order to find out, which bit(s) has changed in the condition register.

Register model:




*The various device series differ in the number of statuses and events they can signalise to the user in the registers. Rule of thumb: if there is no feature "UVD" (undercurrent detection) specified for your device, it won't be present as signal in the status registers, too. Which ones of the status register signals, in relation to certain features, are available for your particular device is specified in the device operating guide.*

*Device alarms, like OVP, have to be acknowledged (see device manual, section about device alarms). They are signalled via CONDITION and EVENT registers. Reading SCPI errors with SYST:ERR? or SYST:ERR:ALL? is also considered as alarm acknowledgement and will clear the alarm bit in CONDITION and in EVENT, but only if the alarm condition is not present anymore. Acknowledged alarms can later only be read from the device as an alarm counter (where featured, only available from a certain KE firmware version). It is recommended to regularly poll alarms from the device and to use the command STAT:QUES? prior to SYST:ERR?.*

#### 5.4.2.1 STATus:QUEStionable[:EVENt]?

Reads the Questionable status EVENT or CONDITIONAL register. The device will return a 16 bit value, which represents device information as defined in the register model in *5.4.2*.

| | |
|---|---|
| Query form 1: | STATus:QUEStionable[:CONDition]? |
| Query form 2: | STATus:QUEStionable:EVENt? |
| Examples: | |
| STAT:QUES? --> 3072 | This value tells, that bits 10 and 11 are set and according to the register model this is interpreted as “remote control = active” and “DC input/output = on”. |
| STAT:QUES:EVEN? | Reads the Event register of the Questionable status register. Any values different from zero signalises, that at least one event has occurred. |

#### 5.4.2.2 STATus:QUEStionable:ENABle <NR1>

This command sets the Enable register of the Questionable status register. The Enable register is a filter that enables all or single bits to signalise an event to the status byte STB. By default, all bits of the Enable register are set. In case you want to ignore certain bits, you just need to add the values of the remaining bits and send the value to the Enable register.

| | |
|---|---|
| Query form: | STATus:QUEStionable:ENABle? |
| Value range: | 0...32767 |
| Example: | |
| STAT:QUES:ENAB␣3072 | Sets the Enable register of the Questionable status registers to 3072 and enables the bits „OVP“, „OT“, „Remote“ and „Input/Output on“ for event reporting to STB. |

#### 5.4.2.3 STATus:OPERation[:EVENt]?

Reads the Operation status EVENT or CONDITIONAL register. The device will return a 16 bit value, which represents device information as defined in the register model in *5.4.2*.

| | |
|---|---|
| Query form 1: | STATus:OPERation[:CONDition]? |
| Query form 2: | STATus:OPERation:EVENt? |
| Examples: | |
| STAT:OPER? --> 256 | Reads the Operation Register. A possible response would be a value of 256, which tells, that bit 8 is set and according to the register model bit 8 signalises, that „CV“ (constant voltage regulation) is active. |
| STAT:OPER:EVEN? | Reads the Event register of the Operation status registers. |

#### 5.4.2.4 STATus:OPERation:ENABle <NR1>

Sets or reads the Enable register of the Questionable status register. The Enable register is a filter. It enables single or all bit of the condition registers to change the corresponding bit in the event register. This also impacts the summary bit in the status byte STB. By default, all bits of the Enable register are set to 1. If you want to use only some specific bits to be left through, just add their bit values (see register model) and send the total to the Enable register.

| | |
|---|---|
| Query form: | STATus:OPERation:ENABle? |
| Value range: | 0...4095 |
| Example: | |
| STAT:OPER:ENAB␣1792 | Sets the Enable register of the Operation register to value 1792 and enables bits „CV“, „CC“ and „CP“ for reporting events to the STB. |

### 5.4.3 Set value commands

#### 5.4.3.1 [SOURce:]VOLTage <NRf>[Unit]

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Sets the input resp. output voltage limit of the device within a certain range, which is either defined by adjustment limits ("Limits", where featured) or is 0...100% nominal value, or reads the last setting. Alternatively, parameters MIN or MAX can be used to instantly set the voltage to the adjustable MINimum or MAXimum.

Query form: [SOURce:]VOLTage?

Value range: <NRf> = 0...nominal voltage, according to technical specs

Examples:

VOLT 12 — Absolute short form. Sets 12 V.

SOUR:VOLTAGE␣24.5V — Mixed form short/long, with unit. Sets 24.5 V, unless the voltage adjustment range has been limited otherwise.

SOURCE:VOLTAGE␣MIN — Sets the voltage to the defined minimum, usually 0 V.

#### 5.4.3.2 [SOURce:]CURRent <NRf>[Unit]

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Sets the input resp. output current limit of the device within a certain range, which is either defined by adjustment limits ("Limits", where featured) or is 0...100% nominal value, or reads the last setting. Alternatively, parameters MIN or MAX can be used to instantly set the current to the adjustable MINimum or MAXimum.

Query form: [SOURce:]CURRent?

Value range: <NRf> = 0...nominal current, according to technical specs

Example:

CURR␣170 — Absolute short form. Sets 170 A.

SOUR:CURRENT␣45.3A — Mixed form short/long, with unit. Sets 45.3 A, unless the adjustment range of the current has been limited otherwise.

SOURCE:CURRENT␣MAX — Sets the current to the defined maximum, the nominal current of the device.

#### 5.4.3.3 [SOURce:]POWer <NRf>[Unit]

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Sets the input resp. output power limit of the device within a certain range, which is either defined by adjustment limits ("Limits", where featured) or is 0...100% nominal value, or reads the last setting. Alternatively, parameters MIN or MAX can be used to instantly set the power to zero (MINimum) or MAXimum.

Query form: [SOURce:]POWer?

Value range: <NRf> = 0...nominal power, according to technical specs

Examples:

POW␣3000 — Absolute short form. Sets 3000 W, unless the power adjustment range has been limited otherwise.

SOUR:POWER␣3.5kW — Mixed form short/long, with unit and magnitude Kilo. Sets 3.5 kW resp. 3500 W, unless the adjustment range of the power has been limited otherwise.

SOURCE:POWER␣MIN — Sets the power to the defined minimum, usually 0 W.

#### 5.4.3.4 [SOURce:]RESistance <NRf>[Unit]

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | – | – | ✓ | – | – | ✓ |

With electronic load devices, this command will set the input resistance value in Ohm within a defined range, as it can be adjusted on the front panel. Power supplies with internal resistance feature use this value to simulate an internal resistor in series to the output, where the output voltage differs from the adjusted value by an amount that calculates from the adjusted resistance value and actual output current. The way of setting the resistance value on both device types is identical. The adjustable range can be limited with an upper adjustment limit. Alternatively, parameters MIN or MAX can be used to instantly set the resistance to the adjustable MINimum or MAXimum.

Query form: [SOURce:]RESistance?

Value range: <NRf> = Min. resistance...max. resistance, according to technical specs

Examples:

RES? Absolute short form. Queries the currently set resistance value.

SOUR:RESISTANCE␣10 Mixed form short/long. Sets 10 Ω.

SOURCE:RES␣MIN Sets the resistance to the minimum defined for the particular device model.

### 5.4.4 Measuring commands

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Actual values, as returned by the measuring commands, are the DC input resp. output values as they are present in the moment they are queried. They are not necessarily identical to the corresponding set values. Your device constantly measures the actual values and returns the last snapshot when queried.

#### 5.4.4.1 MEASure:[SCALar:]VOLTage[:DC]?

Queries the device to return the last measured DC input resp. output voltage value in Volt.

Example:

MEAS:VOLT? Absolute short form. Queries the actual voltage. A response, which should be instantly coming from the device, will return a value between 0% and max. 125% of nominal device voltage, like for example “43.50 V”. The number of decimal places in the returned value will be identical to the value format in the device display and varies from model to model.

#### 5.4.4.2 MEASure:[SCALar:]CURRent[:DC]?

Queries the device to return the last measured DC input resp. output current value in Ampere.

Example:

MEASURE:CURRENT? Queries the actual current only. A response, which should be instantly coming from the device, will return a value between 0% and max. 125% of nominal device current, for example “100.1 A”. The number of decimal places in the returned value will be identical to the value format in the device display and varies from model to model.

#### 5.4.4.3 MEASure:[SCALar:]POWer[:DC]?

Queries the device to return the last calculated DC input resp. output power value in Watts.

Example:

MEAS:POW? Absolute short form. Queries the consumed (e-load) resp. supplied power (PSU). A response, which should be instantly coming from the device, will return a value between 0% and max. 125% of nominal device power, for example “2534 W”. No matter how the actual power format is in the device’s display, here it will always be returned in Watts.

#### 5.4.4.4 MEASure:[SCALar:]ARRay?

Queries the device to return the last measured resp. calculated actual values of voltage, current and power (in that sequence) , separated by commas and with unit and eventually magnitude.

Example:

| | |
|---|---|
| MEAS:ARR? | Absolute short form. A response, which should be instantly coming from the device, will return three values between 0% and max. 125% of nominal device values, for example “12.5 V, 33.3 A, 420 W” |

### 5.4.5 Status commands

Status commands are used to alter the status of the device in terms of activating remote control or switching the DC input/output, or to query the current status.

#### 5.4.5.1 SYSTem:LOCK <B0>

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

This command is used to activate remote control of a device. Basically, remote control has to be activated first before you can send any command that changes device status or value. Once remote control has been activated via one of the digital interfaces, only that interface is in charge.

The activation of remote control can be refused by the device due to several reasons. It is usually replied in form of a SCPI error which is put into the SCPI error buffer. This buffer can be read with the error command (see *„5.4.5.4. SYSTem:ERRor?“*).

| | |
|---|---|
| Query form: | SYSTem:LOCK:OWNer? |
| Value range for set: | {1 \| ON} resp. {0 \| OFF} |
| Value range for query: | {REMOTE, NONE, LOCAL} |
| Examples: | |
| SYST:LOCK␣ON | Absolute short form. Requests the device to switch to remote control. The device then usually indicates activated remote control either by a LED or a status text in the display. |
| SYSTEM:LOCK:OWNER? | Queries the lock owner regarding remote control. This can be used to verify whether the device has accepted the request to switch to remote control or not. It can return three different statuses:<br>REMOTE = Device is in remote control via any of the available interfaces<br>NONE = Device is not in remote control<br>LOCAL = Device is in LOCAL condition, which denies or interrupts remote control. Usually actually manually on the device’s front panel |

#### 5.4.5.2 INPut <B0>

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | ✓ | – | – | – | – | ✓ |

This command is used to switch the DC input of devices with an input, here: electronic loads, on or off. The main key word SOURCE is taken from the SCPI standard and actually does not fit an energy sinking device, but is used for that purpose.

| | |
|---|---|
| Query form: | INPut? |
| Value range: | {1 \| ON} resp. {0 \| OFF} |
| Examples: | |
| INP␣1 | Absolute short form. Switches the DC input on if remote control is active. |
| INPUT? | Queries the condition of the DC input, which will be returned as ON or OFF. Against all expectations, the input might be switched off due to a device alarm. |

#### 5.4.5.3 OUTPut <B0>

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| – | – | ✓ | ✓ | ✓ | ✓ | ✓ |

This command is used to switch the DC output on or off with devices that have an output, like power supplies or other sources.

| | |
|---|---|
| Query form: | OUTPut? |
| Value range: | {1 \| ON} resp. {0 \| OFF} |
| Examples: | |
| OUTP␣1 | Absolute short form. Switches the DC output on if remote control is active. |
| OUTPUT? | Queries the condition of the DC output, which will be returned as ON or OFF. Against all expectations, the output might be switched off due to a device alarm. |

#### 5.4.5.4 SYSTem:ERRor?

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

This commands is used to read a single error or all errors from the device's internal SCPI error queue. This queue only contains errors in relation to commands, i.e. wrong syntax, too high values etc. It can not return any device alarm. Those are usually queried from the device by reading bits of the status registers (see *„5.4.2. Status registers“*). You can chose either to query the next error multiple times until it says “No error” or generally query all at once. After all errors have been read from the buffer, it will be purged.

The queue is of type FIFO (first in, first out). It means, that the first occurred error is put out first when querying them.

| | | |
|---|---|---|
| Query form 1: | SYSTem:ERRor? | Queries the last or next error |
| Query form 2: | SYSTem:ERRor:NEXT? | Queries the last or next error |
| Query form 3: | SYSTem:ERRor:ALL? | Queries all errors in the buffer (up to 5) |
| Example: | | |
| SYST:ERR? | Absolute short form. The device replies to this query with a string that first contains an error code (see error code list) and second an error description, for example: 0,“No error“. This is returned every time no error is present or after all error have been returned. | |
| SYSTEM:ERROR:ALL? | This query will let the device return up to 5 concatenated errors in one string, separated by comma plus space. | |

*Querying errors with SYST:ERR? also clears bits related to device alarms in register QUEStionable (see „5.4.2. Status registers“), but only if the alarm condition is "gone". This is considered as acknowledgement by the user. Alarms that have been acknowledged this way can then not be read from the register anymore.*

### 5.4.6 Commands for protective features

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

Devices from EA, like power supplies or electronic loads, feature a set of device alarms, partly for self-protection, partly for the protection of connected loads resp. sources. There is furthermore a supervision feature which can monitor DC input/output related values like voltage, current or power for exceeding adjustable limits and initiate user-definable actions like an acoustic alarm or shutdown of the DC input/output. The configuration of the supervision can be done manually in the actual user profile or by remote commands.

#### 5.4.6.1 [SOURce:]VOLTage:PROTection[:LEVel] <NRf>[Unit]

This command is connected to the adjustable value “OVP” (overvoltage protection, see SETTINGS menu on the device front panel). The value is adjustable between 0 and 110% nominal device voltage. It defines a threshold where the device switches the DC input/output off, no matter if the device has generated a voltage higher than this threshold or any outside source. When controlling a source, i.e. power supply, this feature usually serves to protect the connected load from overvoltage and thus damage. This can occur if the output voltage is accidentally adjusted to a dangerous level.

A sink, i.e. an electronic load, can not be protected from overvoltage from outside, though it will switch off the DC input at this threshold.

Query form: [SOURce:]VOLTage:PROTection[:LEVel]?

Value range: 0...1.1*nominal voltage of the device

Examples:

VOLT:PROT␣88 Absolute short form. Sets the OVP threshold to 88 V. At a model with 80 V nominal voltage, this is 110% of the maximum voltage and the maximum OVP value.

SOURCE:VOLTAGE:PROTECTION 65V Absolute long form, with unit. Sets this threshold to 65 V.

#### 5.4.6.2 [SOURce:]CURRent:PROTection[:LEVel] <NRf>[Unit]

This command is connected to the adjustable value “OCP” (overcurrent protection, see SETTINGS menu on the device front panel). The value is adjustable between 0 and 110% nominal device current. It defines a threshold where the device switches the DC input/output off. With sinks, like an electronic load is one, it usually serves to protect the source from too high current consumption which might possibly damage the source.

Query form: [SOURce:]CURRent:PROTection[:LEVel]?

Value range: 0...1.1*nominal current of the device

Example:

CURR:PROT␣100 Absolute short form. Sets the OCP threshold to 100 A. Once the input/output current reaches the threshold, the device will instantly switch the DC input/output off. The threshold is only effective if it is adjusted to a lower value then the input/output current, because else the device will just limit the current, but not switch off. If current value and overcurrent protection are adjusted to the same value, the OCP has priority and will switch off rather than limit.

#### 5.4.6.3 [SOURce:]POWer:PROTection[:LEVel] <NRf>[Unit]

This command is connected to the adjustable value “OPP” (overpower protection, see SETTINGS menu on the device front panel). The value is adjustable between 0 and 110% nominal device power. It defines a threshold where the device switches the DC input/output off. This feature shall help to protect a source or load from exceeding a certain power output resp. consumption, regardless of voltage and current.

Query form: [SOURce:]POWer:PROTection[:LEVel]?

Value range: 0...1.1*nominal power of the device

Example:

POW:PROT␣1.5kW Absolute short form. Sets the OPP threshold to 1.5 kW. Once the input/output power reaches the threshold, the device will instantly switch the DC input/output off. The threshold is only effective if it is adjusted to a lower value then the input/output power, because else the device will just limit the power, but not switch off. If power value and overpower protection are adjusted to the same value, the OPP has priority and will switch off rather than limit.

### 5.4.7 Commands for supervision features

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | – | – | ✓ | – | – | ✓ |

The commands below enable the remote configuration of the supervision features (Events) of the device, related to voltage, current or power on the DC input or DC output.

| Command | Beschreibung |
|---|---|
| **SYSTem:CONFig:UVD[?] <NRf>[Unit]** | Identical to event UVD, as configurable at the device |
| **SYSTem:CONFig:UVD:ACTion[?] {NONE \| SIGNAL \| WARNING \| ALARM}** | |
| **SYSTem:CONFig:UCD[?] <NRf>[Unit]** | Identical to event UCD, as configurable at the device |
| **SYSTem:CONFig:UCD:ACTion[?] {NONE \| SIGNAL \| WARNING \| ALARM}** | |
| **SYSTem:CONFig:OVD[?] <NRf>[Unit]** | Identical to event OVD, as configurable at the device |
| **SYSTem:CONFig:OVD:ACTion[?] {NONE \| SIGNAL \| WARNING \| ALARM}** | |
| **SYSTem:CONFig:OCD[?] <NRf>[Unit]** | Identical to event OCD as configurable at the device |
| **SYSTem:CONFig:OCD:ACTion[?] {NONE \| SIGNAL \| WARNING \| ALARM}** | |
| **SYSTem:CONFig:OPD[?] <NRf>[Unit]** | Identical to event OPD, as configurable at the device |
| **SYSTem:CONFig:OPD:ACTion[?] {NONE \| SIGNAL \| WARNING \| ALARM}** | |

The **:ACTion** can have following parameters (also see the device's operation guide):

**NONE** = Event inactive, no supervision

**SIGNAL** = As soon as the event occurs, a status text is presented in the status field of the device display resp. a bit in the Questionable Register (STAT:QUES?) is set (see *„5.4.2. Status registers“*). The bit indicates, that a specific event has occurred. This can be used to record the event.

**WARNING** = As soon as the event occurs, a warning pop-up is presented in the device display resp. a bit in the Questionable Register (STAT:QUES?) is set (see *„5.4.2. Status registers“*). The bit indicates, that a specific event has occurred. This can be used to record the event.

**ALARM** = As soon as the event occurs, a warning pop-up is presented in the device display, as well as an acoustic alarm is initiated and the DC input/output is switched off resp. a bit in the Questionable Register (STAT:QUES?) is set (see *„5.4.2. Status registers“*). The bit indicates, that a specific event has occurred. This can be used to record the event.

*The action ALARM lets the device act similar to when device alarms occur. However, device alarm still have priority. It means, that if, for example, the values OVP and OVD would be equal and the input/output voltage reaches that level, the device would initiate an OV alarm rather than an OVD event.*

### 5.4.8 Commands for adjustment limits

Adjustment limits are additional, globally effective, adjustable limits for the set values U, I, P and R (where featured). The purpose is to narrow the standard 0...100% adjustment range and to prevent, for example, to accidently set a too high voltage for the load. There is also the overvoltage protection (OVP), but it's generally better to block irregular set values in the first place.

In case a set value is sent to the device that would exceed an adjustment limit, no matter if too high or too low, the device will ignore it and put an error into the error queue. At the same time it is impossible to set the lower adjustment limit (:LOW) higher than the related set value or, vice versa, the upper adjustment limit.

These commands are connected to the "Limits" setting as you can adjust them in the setup menu of your device, (were featured). Refer to the device manual for details.

| Command | ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|---|
| **SOURce:VOLTage:LIMit:LOW[?] <NRf>[Unit]**<br>Identical to value U-min, as configurable at the device | ✓ | ✓ | ✓ | ✓ | — | ✓ | ✓ |
| **SOURce:VOLTage:LIMit:HIGH[?] <NRf>[Unit]**<br>Identical to value U-max, as configurable at the device | ✓ | ✓ | ✓ | ✓ | — | ✓ | ✓ |
| **SOURce:CURRent:LIMit:LOW[?] <NRf>[Unit]**<br>Identical to value I-min, as configurable at the device | ✓ | ✓ | ✓ | ✓ | — | ✓ | ✓ |
| **SOURce:CURRent:LIMit:HIGH[?] <NRf>[Unit]**<br>Identical to value I-min, as configurable at the device | ✓ | ✓ | ✓ | ✓ | — | ✓ | ✓ |
| **SOURce:POWer:LIMit:HIGH[?] <NRf>[Unit]**<br>Identical to value P-max, as configurable at the device | ✓ | ✓ | ✓ | ✓ | — | ✓ | ✓ |
| **SOURce:RESistance:LIMit:HIGH[?] <NRf>[Unit]**<br>Identical to value R-max, as configurable at the device | ✓ | ✓ | — | ✓ | — | — | ✓ |

### 5.4.9 Commands for master-slave operation

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | – | – | ✓ | – | ✓ | ✓ |

The commands, as listed below, are used to remotely configure and control the master-slave mode (short: MS). The commands are connected to the related settings in the device setup menu. For details about MS refer to the device's operation manual.

Configuration and control require a certain procedure. It applies generally:

- Configuration should always be first, but can also be done on the device's front panel, so that after activating remote control the master-slave operation can instantly start

The commands in the tables below are listed in the sequence they should be used (top to bottom).

#### 5.4.9.1 Configuration commands

The configuration of MS can be skipped if already done manually at the device's HMI or formerly in remote control and nothing has changed.

| Command | Description |
|---|---|
| **SYSTem:MS:ENABle {ON \| OFF}** | Enables (ON) or disables (OFF) master-slave (MS) mode |
| **SYSTem:MS:ENABle?** | Queries, whether the MS is enabled or not |
| **SYSTem:MS:LINK {MASTER \| SLAVE}**<br>**SYSTem:MS:LINK?** | Defines or queries the role of the device in the MS system:<br>**MASTER** = Device will be master unit<br>**SLAVE** = Device will be slave unit |
| **SYSTem:MS:ADDRess <NR1>**<br>**SYSTem:MS:ADDRess?** | Defines or queries the device's slave address in the MS system. It applies:<br>**0** = Always assigned to master unit, cannot be changed after defining a unit as master with SYST:MS:LINK<br>**1**...**15** = arbitrary, as long as the device is defined as slave |
| **SYSTem:MS:INITialisation** | Starts the MS initialisation with the given settings. Also refer to the devices's operating manual. After a successful initialisation, the MS mode can be controlled with further commands. To test if the init has been successful, the next command can be used: |
| **SYSTem:MS:CONDition?** | Queries the result of a former MS init. Possible return values:<br>**INIT** = Init was successful<br>**NO INIT** = Init was not successful<br>An init is also successful if there is only a master. In order to find out whether you have a complete MS system available or not, you would have to query the number of initialised units from the master with command **SYST:MS:UNITS?** (see below). Any value other than 0 means, the MS system is initialised and available for control. |
| **SYSTem:MS:UNITs?** | Queries the number of units that have been initialised successfully. The number can differ from the expected value, if the master did not initialise one or multiple slaves due to any reason (no connection, wrong parameters, double addresses etc.). If only the master has initialised itself, the command will return a 1. |
| **SYSTem:SHARe:LINK {MASTER \| SLAVE}**<br>**SYSTem:SHARe:LINK?** | This commands only works, if<br>• the device is set to MASTER for master-slave and<br>• master-slave mode is activated and<br>• the device is an electronic load,<br>else the device will generate a "Settings conflict" error.<br>For the correct function of the so-called "two-quadrants operation" (for details refer to the device manual), it is imperative to set an electronic load unit to mode "Share bus slave" with command **SYST:SHAR:LINK SLAVE** |

#### 5.4.9.2 Other MS commands

| Command | Description |
|---|---|
| **SYSTem:MS:NOMinal:VOLTage?** | Queries the total voltage of the initialised MS system. The voltage of a MS system will increase in a series connection, compared to a single unit. Series connection is only allowed for power supplies |
| **SYSTem:MS:NOMinal:CURRent?** | Queries the total current of the initialised MS system. The current of a MS system will increase on a parallel connection, compared to a single unit |
| **SYSTem:MS:NOMinal:POWer?** | Queries the total power of the initialised MS system. The power of a MS system will always increase, compared to a single unit |

*The nominal values, as calculated during a MS initialisation procedure, are different to the nominal values of a single device and thus have extra commands. These values are only used for master-slave purposes.*

### 5.4.10 Commands for general queries

Here are commands listed that can be used to query other information from the device, which is not used so often.

| Command | ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|---|
| **SYSTem:NOMinal:VOLTage?**<br>Queries the nominal, i.e. maximum input/output voltage | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **SYSTem:NOMinal:CURRent?**<br>Queries the nominal, i.e. maximum input/output current | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **SYSTem:NOMinal:POWer?**<br>Queries the nominal, i.e. maximum input/output power | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **SYSTem:NOMinal:RESistance:MINimum?**<br>Queries the minimum internal resistance value. This value is usually not zero with electronic loads. | ✓ | — | — | ✓ | — | — | ✓ |
| **SYSTem:NOMinal:RESistance:MAXimum?**<br>Queries the maximum internal resistance value | ✓ | — | — | ✓ | — | — | ✓ |
| **SYSTem:DEVice:CLAss?**<br>Queries the device class and lets the device return a value which defines to what device series the device belongs to. This is an easy way to distinguish different device types, like an electronic load from a power supply or battery charger. See *„A1. Device classes“* | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

### 5.4.11 Commands for device configuration

The commands as listed below are used to modify settings of the device configuration. The settings can be part of the current user profile (see device's operating manual). Any modification on the configuration requires activated remote control. These settings are automatically stored.

| Command | ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|---|
| **SYSTem:CONFig:USER:TEXT <SRD>**<br>**SYSTem:CONFig:USER:TEXT?**<br>Writes or queries a user-definable text of up to 40 characters permanently to the device. This string can be used to add custom information to the unit, in order to distinguish it from other, identical models, alternatively to the serial number. | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **SYSTem:CONFig:INPut:RESTore {AUTO \| OFF}**<br>**SYSTem:CONFig:OUTPut:RESTore {AUTO \| OFF}**<br>Defines the condition of DC input/output after the device is powered. This is connected to the device setting "**Input after power on**" resp. "**Output after power on**"<br>**AUTO** = DC input/output will be restored to the condition it had when switching the device off the last time<br>**OFF** = DC input/output will always be off | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

| Command | ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|---|
| **SYSTem:CONFig:INPut:RESTore?**<br>**SYSTem:CONFig:OUTPut:RESTore?**<br>Queries the most recent selection of the above setting | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **SYSTem:CONFig:ANAlog:REFerence {5 \| 10}**<br>**SYSTem:CONFig:ANAlog:REFerence?**<br>Select the voltage range for analog inputs and outputs of the analog interface. This has no effect on anything concerning digital remote control.<br>**5** = 0...5 V range<br>**10** = 0...10 V range (factory setting) | ✓ | — | ✓ | ✓ | ✓ | ✓ | ✓ |
| **SYSTem:CONFig:ANAlog:REMSB:LEVel {NORMAL \| INVERTED}**<br>**SYSTem:CONFig:ANAlog:REMSB:LEVel?**<br>Determines how pin REM-SB of the analog interface (see device manual) shall be interpreted by the device:<br>**NORMAL** = level and conditions as described in the manual (factory setting)<br>**INVERTED** = level and conditions are interpreted as inverted | ✓ | — | ✓ | ✓ | ✓ | ✓ | ✓ |
| **SYSTem:CONFig:ANAlog:REMSB:ACTion {OFF \| AUTO}**<br>**SYSTem:CONFig:ANAlog:REMSB:ACTion?**<br>Determines the action that is caused by using pin REM-SB of the analog interface in connection with DC input/output of the device:<br>**OFF** = pin can only be used to switch the DC input/output off<br>**AUTO** = pin can be used to switch off and on again, if the DC input/output was at least switched on once by pushbutton on the control panel or digital command | ✓ | — | ✓ | ✓ | ✓ | ✓ | ✓ |
| **SYSTem:CONFig:MODE {UIP \| UIR}**<br>**SYSTem:CONFig:MODE?**<br>Selects the operation mode between U/I/P and U/I/R. Both modes are available for electronic loads and also on selected power supplies. By selecting U/I/R, the adjustable resistance value (command **[SOURce:]RESistance**) is unlocked. Activated U/I/R mode can only be detected in the display from the resistance value being shown. | ✓ | — | — | ✓ | — | — | ✓ |
| **[SOURce:]VOLTage:CONTrol:SPEed {FAST \| SLOW}**<br>**[SOURce:]VOLTage:CONTrol:SPEed?**<br>This command is used to switch the internal voltage regulator of the power stage(s) between FAST and SLOW (default) mode. Both modes are not clearly defined regarding time and dynamics. The desired effect when switch to FAST or vice versa can be different or even the opposite of what was expected. | ✓ | ✓ | — | — | — | — | — |
| **SYSTem:COMMunicate:PROTocol:MODBus {ENABLE \| DISABLE}**<br>**SYSTem:COMMunicate:PROTocol:MODBus?**<br>Enables or disables ModBus protocol on the device. This setting is stored. After disabling ModBus with this command, further ModBus messages are ignored, so that only SCPI commands are accepted. Only one of both protocols can be deactivated at the same time. | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **SYSTem:COMMunicate:TIMeout {5...65535}**<br>**SYSTem:COMMunicate:TIMeout?**<br>Defines a timeout in milliseconds (factory setting: 5 ms), a max. time that can elapse between two consecutive bytes, before the device considers the message as "completely received". For details refer to section *3.6.*<br>*Note: this only applies for serial interface (USB, RS232)* | ✓ | — | ✓ | ✓ | ✓ | ✓ | ✓ |

#### 5.4.11.1 Commands for Anybus module settings

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | – | – | ✓ | – | ✓ | – |

Most of the Anybus interface modules can also be remotely configured using SCPI commands, either via USB port or even via the interface itself. These settings are always saved automatically.

| Command | Description |
|---|---|
| **SYSTem:COMMunicate:INTerface:CODE?** | Returns a value, representing a model code for the installed Anybus interface module:<br>5 = Profibus<br>9 = RS232<br>16 = CANopen<br>18 = ModBus TCP 1P<br>19 = Profinet/IO 1P<br>20 = Ethernet 1P<br>21 = Ethernet 2P<br>22 = ModBus TCP 2P<br>23 = Profinet/IO 2P<br>25 = CAN<br>26 = EtherCAT |
| **SYSTem:COMMunicate:INTerface:TYPE?** | Queries the name of the installed Anybus interface module. |
| **SYSTem:COMMunicate:INTerface:SERial?** | Queries the serial number of the installed Anybus interface module. |
| **SYSTem:COMMunicate:INTerface:ADDRess <NR1>**<br>**SYSTem:COMMunicate:INTerface:ADDRess?** | Sets the Profibus address of the Profibus module IF-AB-PBUS or queries it. Allowed range: 0...125 |
| **SYSTem:COMMunicate:PROFibus:ID?** | Queries the Profibus ID of the device manufacturer. |
| **SYSTem:COMMunicate:PROFibus:FTAG <SRD>**<br>**SYSTem:COMMunicate:PROFibus:FTAG?** | Sets or queries the Profibus/Profinet specific „function tag“, a string of up to 32 characters |
| **SYSTem:COMMunicate:PROFibus:LTAG <SRD>**<br>**SYSTem:COMMunicate:PROFibus:LTAG?** | Sets or queries the Profibus/Profinet specific „location tag“, a string of up to 22 characters |
| **SYSTem:COMMunicate:PROFibus:DATE <SRD>**<br>**SYSTem:COMMunicate:PROFibus:DATE?** | Sets or queries the Profibus/Profinet specific „date tag“, a date/time string of up to 40 characters |
| **SYSTem:COMMunicate:PROFibus:DESCription <SRD>**<br>**SYSTem:COMMunicate:PROFibus:DESCription?** | Sets or queries the Profibus/Profinet specific „description“ tag, a string of up to 54 characters |
| **SYSTem:COMMunicate:PROFibus:NAME <SRD>**<br>**SYSTem:COMMunicate:PROFibus:NAME?** | Sets or queries the Profinet specific „station name”, a string of up to 200 characters |
| **SYSTem:COMMunicate:INTerface:BAUD <NR1>**<br>**SYSTem:COMMunicate:INTerface:BAUD?** | Queries or sets the bus speed, i.e. baud rate of a CANopen or RS232 interface module. The device will only save the value. This means, with value 3 being saved and CANopen installed, it will run at 100 kbps and with RS232 installed, with 19200 Baud. |
| **SYSTem:COMMunicate:CAN:BROadcast <NR1>**<br>**SYSTem:COMMunicate:CAN:BROadcast?** | Sets the CAN broadcast ID for normal CAN communication. Allowed range:<br>**0...2047** (11 bit) resp. **0...536870911** (29 bit) |

| Value | CANopen | CAN | RS232 |
|---|---|---|---|
| 0 | 10 kbps | 10 kbps | 2400 Bd |
| 1 | 20 kbps | 20 kbps | 4800 Bd |
| 2 | 50 kbps | 50 kbps | 9600 Bd |
| 3 | 100 kbps | 100 kbps | 19200 Bd |
| 4 | 125 kbps | 125 kbps | 38400 Bd |
| 5 | 250 kbps | 250 kbps | 57600 Bd |
| 6 | 500 kbps | 500 kbps | 115200 Bd |
| 7 | 800 kbps | 1 Mbps | - |
| 8 | 1 Mbps | - | - |
| 9 | Auto | - | - |

| Command | Description |
|---|---|
| **SYSTem:COMMunicate:CAN:DLC {AUTO \| FILL}**<br>**SYSTem:COMMunicate:CAN:DLC?** | CAN data length setting for response messages from the device.<br>**AUTO** = the number of data bytes in a CAN message from the device (response) varies according to the used command/register (default)<br>**FILL** = the number of data bytes in a CAN message is always 8, filled with zeros |
| **SYSTem:COMMunicate:CAN:FORMat {BASE \| EXT}**<br>**SYSTem:COMMunicate:CAN:FORMat?** | Selects the CAN address format.<br>**BASE** = 11 Bit (CAN 2.0A) (default)<br>**EXT** = 29 Bit (CAN 2.0B) |
| **SYSTem:COMMunicate:CAN:NODe <NR1>**<br>**SYSTem:COMMunicate:CAN:NODe?** | Sets the CAN base ID for normal CAN communication. Allowed range:<br>**0...2047** (11 bit) resp. **0...536870911** (29 bit) |
| **SYSTem:COMMunicate:CAN:READ:NODe <NR1>**<br>**SYSTem:COMMunicate:CAN:READ:NODe?** | Sets the CAN base ID for cyclic read over CAN. Also see section *8.3.5*. Allowed range:<br>**0...2047** (11 bit) resp. **0...536870911** (29 bit) |
| **SYSTem:COMMunicate:CAN:READ:ACTual <NR1>**<br>**SYSTem:COMMunicate:CAN:READ:ACTual?** | Defines the interval (in milliseconds) for the cyclic read of the device's actual values (U, I, P) over CAN interface IF-AB-CAN. Also see section *8.3.5*. Allowed parameter range: **0** or **20...5000**<br>(0 = cyclic read for this object is deactivated) |
| **SYSTem:COMMunicate:CAN:READ:ALIMits <NR1>**<br>**SYSTem:COMMunicate:CAN:READ:ALIMits?** | Defines the interval (in milliseconds) for the cyclic read of the device's adjustment limits for U and I over CAN interface IF-AB-CAN. Also see section *8.3.5*. Allowed parameter range: **0** or **20...5000**<br>(0 = cyclic read for this object is deactivated) |
| **SYSTem:COMMunicate:CAN:READ:BLIMits <NR1>**<br>**SYSTem:COMMunicate:CAN:READ:BLIMits?** | Defines the interval (in milliseconds) for the cyclic read of the device's adjustment limits for P and R over CAN interface IF-AB-CAN. Also see section *8.3.5*. Allowed parameter range: **0** or **20...5000**<br>(0 = cyclic read for this object is deactivated) |
| **SYSTem:COMMunicate:CAN:READ:SETS <NR1>**<br>**SYSTem:COMMunicate:CAN:READ:SETS?** | Defines the interval (in milliseconds) for the cyclic read of the device's set values (U, I, P, R) over CAN interface IF-AB-CAN. Also see section *8.3.5*. Allowed parameter range: **0** or **20...5000**<br>(0 = cyclic read for this object is deactivated) |
| **SYSTem:COMMunicate:CAN:READ:STAT <NR1>**<br>**SYSTem:COMMunicate:CAN:READ:STAT?** | Defines the interval (in milliseconds) for the cyclic read of the device's status over CAN interface IF-AB-CAN. Also see section *8.3.5*. Allowed parameter range: **0** or **20...5000**<br>(0 = cyclic read for this object is deactivated) |
| **SYSTem:COMMunicate:CAN:TERMination {ON \| OFF}**<br>**SYSTem:COMMunicate:CAN:TERMination?** | Switches the integrated CAN bus termination resistor ON or OFF |

#### 5.4.11.2 Commands for Ethernet interface settings

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

The commands below are related to any Ethernet interface port, no matter if built-in or pluggable module. The only exceptions are SYST:COMM:LAN:1SPEED and SYS:COMM:LAN:2SPEED, which are only supported for Anybus Ethernet modules while with built-in ports these parameters are permanently set to AUTO.

| Command | Description |
|---|---|
| **SYSTem:COMMunicate:INTerface:CODE?** | Returns a value, representing a model code for the installed Anybus interface module: |
| **SYSTem:COMMunicate:LAN:DHCP[?] {ON \| OFF}** | Activates (=ON) or deactivates (=OFF) the DHCP functionality of Ethernet capable devices. Default is OFF, so the IP, as set with :ADDR command above, is used |
| **SYSTem:COMMunicate:LAN:ADDRess[?] <SRD>** | Queries or sets the IP address of the network port. This IP is used by every Ethernet module. When setting the IP, the string has to in the typical IP format like this: 192.168.0.2 |
| **SYSTem:COMMunicate:LAN:SMASk[?] <SRD>** | Queries or sets the subnet mask of the network port. This subnet mask is used by every Ethernet module. Format is the same as with the IP. |
| **SYSTem:COMMunicate:LAN:GATEway[?] <SRD>** | Queries or sets the gateway address of the network port. This gateway mask is used by every Ethernet module. Format is the same as with the IP. This address is often not used and can thus be left at the default IP. |
| **SYSTem:COMMunicate:LAN:MAC?** | Queries the MAC of the currently installed Ethernet interface. If no Ethernet interface is installed, an error will be generated |
| **SYSTem:COMMunicate:LAN:CONTrol[?] {0...65535}** | Queries or sets the current TCP port, which is used by all Ethernet module. This port is default 5025 and used for ModBus or SCPI communication. The Ethernet interface may also feature further ports which are not modifiable. |
| **SYSTem:COMMunicate:LAN:HOSTname[?] <SRD>** | Queries or set the host name (refer to network terminology for details). This is a simple ASCII string of up to 54 characters. |
| **SYSTem:COMMunicate:LAN:DOMain[?] <SRD>** | Queries or sets the domain name (refer to network terminology for details). This is a simple ASCII string of up to 54 characters.<br>The domain can be used to select and access a particular device in the network without knowing the IP address. |
| **SYSTem:COMMunicate:LAN:DNS1[?] <SRD>**<br><br>**SYSTem:COMMunicate:LAN:DNS2[?] <SRD>** | Queries or sets the network addresses of the first (DNS1, any Ethernet port) and second DNS (DNS2, only with 2-port Anybus) server, if required. |
| **SYSTem:COMMunicate:LAN:1SPEed[?] {AUTO \| 10HALF \| 10FULL \| 100HALF \| 100FULL**<br><br>**SYSTem:COMMunicate:LAN:2SPEed[?] {AUTO \| 10HALF \| 10FULL \| 100HALF \| 100FULL** | Only for IF-AB Ethernet modules (ETH, PNET, MBUS)<br>Sets the communication speed of the network port(s) of Anybus Ethernet interfaces with one port (P1) or two ports (P1, P2):<br>**AUTO** = Auto negotiation<br>**10HALF** = 10MBit, half duplex<br>**10FULL** = 10MBit, full duplex<br>**100HALF** = 100MBit, half duplex<br>**100FULL** = 100MBit, full duplex |

| Command | Description |
|---|---|
| **SYSTem:COMMunicate:LAN:TIMeout[?] {5...65535}** | Defines a socket connection timeout for all Ethernet based interfaces. Default value: 5 seconds, range: 5...65535 seconds. Also see *„3.7. Connection timeout“* |
| **SYSTem:COMMunicate:LAN:KEEPalive[?] {ON \| OFF}** | Enables / disables the so-called TCP keep-alive timeout.<br>Default: OFF |

### 5.4.12 Commands for remote control of the function generator

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | – | – | ✓ | – | – | ✓ |

*Sequence data or table data, which you can write via SCPI commands, is **not stored** in the device.*

The function generator is a complex part of the whole control options of the device. It can be remotely configured and controlled by a set of SCPI commands. When operating the function manager on the control panel of the device, it requires a certain procedure of setup before getting to the actual starting point. The single commands can not enforce that procedure, so it's up to the user to use them in the correct sequence. What to do:

**1)** You need to configure the function generator. One of the first things is to select the type of function generator you want to use. Further steps depend on your selection. There are two types available: XY and arbitrary. The XY function generator is usually only used for the functions UI and IU, the arbitrary generator is used for the other functions like sine wave, square wave etc.

**2)** If you are going to use the arbitrary generator, there is another setup part to do where you first select to which DC input/output value the function, voltage (U) or current (I), will be applied to. The power (P) is not available for wave generation. As a third part of the setup, you should set up the number of sequences to use. This does not happen automatically when filling a certain number of sequences with data, which is done later on.

If you are going to use the XY generator, there is another setup part to do where you need to select whether it shall generator an U-I or I-U curve. Depending on that setting, values you will write to the table, are interpreted. Without this selection the device won't accept any table value.

**3)** Now the sequence(s) to use are filled with data. The arbitrary generator can use X out of 100 possible sequences, which are written with three commands each. The number of sequences to use is variable, but at least 1.

The XY generator, on the other hand, should be filled with 4096 values which are then interpreted as either voltage, for UI function, or current for IU function, depending on the selection in step 2.

After this, the function generator is completely configured and can be started.

*When submitting curve data, it is loaded into an internal buffer which takes some time to wait before the command is accepted. We recommend to wait at least 2 seconds after submitting the data and before the next command.*

#### 5.4.12.1 Configuration of the function generator

| Command | Description |
|---|---|
| **[SOURce:]FUNCtion:GENerator:SELect {VOLTAGE \| CURRENT \| UI \| IU \| PV \| NONE}**<br>**[SOURce:]FUNCtion:GENerator:SELect?** | Select the type of function generator:<br>**VOLTage** = Arbitrary generator for U<br>**CURRent** = Arbitrary generator for I<br>**UI** = XY generator for U-I curve<br>(not with EL 9000 DT and PSI 9000 DT)<br>**IU** = XY generator for I-U curve<br>(not with EL 9000 DT and PSI 9000 DT)<br>**PV** = XY generator for PV curve<br>(only with PSI 9000 2U / 3U / 15U / 24U)<br>**FC** = XY generator for FC curve<br>(only with PSI 9000 2U / 3U / 15U / 24U)<br>**NONE** = Exit function generator |
| **[SOURce:]FUNCtion:GENerator:WAVE:STARt {1..100}**<br>**[SOURce:]FUNCtion:GENerator:WAVE:STARt?** | **Arbitrary generator only**<br>Defines the start, i.e. first sequence (1...100) or queries the last setting. If only one sequence is used, then it must be **:STARt** = **:END**. Sequences going to be used should also be written with data before submitting them. |
| **[SOURce:]FUNCtion:GENerator:WAVE:END {1..100}**<br>**[SOURce:]FUNCtion:GENerator:WAVE:END?** | **Arbitrary generator only**<br>Defines the end, i.e. last sequence (1...100) or queries the last setting. If only one sequence is used, then it must be **:STARt** = **:END**. |

| Command | Description |
|---|---|
| **[SOURce:]FUNCtion:GENerator:WAVE:NUMber {0..999}**<br>**[SOURce:]FUNCtion:GENerator:WAVE:NUMber?** | **Arbitrary generator only**<br>Defines, how often the sequence block from **:STARt** to **:END** is cycled through, or queries the last setting.<br>0 = infinite cycles<br>1...999 = number of cycles |

#### 5.4.12.2 Load sequence data (arbitrary generator)

Sequence data should only be sent to the device after it was switched to function generator mode, which also sets the assignment of the arbitrary generator to U or I.

A function can consist of 1 to 100 sequences, so one sequence is either a complete function or just a part of it. When started, the function generator will execute the sequences from start sequence to end sequence, as defined by the user. With every sequence being variable, the resulting function can be quite complex. The sequence data is loaded into the device with three commands and in a specific order like this:

| Command | Description |
|---|---|
| **[SOURce:]FUNCtion:GENerator:WAVE:LEVel {1...100}**<br>**[SOURce:]FUNCtion:GENerator:WAVE:LEVel?** | **1.** Selects a sequence (similar to HMI access) to write or queries the currently selected sequence number |
| **[SOURce:]FUNCtion:GENerator:WAVE:INDex {0...7}**<br>**[SOURce:]FUNCtion:GENerator:WAVE:INDex?** | **2.** For the selected sequence, a set of parameters can be configured. This command selects the parameter between 0 and 7 with value **INDex**. The next command (**:DATA)**, is then used to write a value. The indexes are explained below. Can also be used to query the current index. |
| **[SOURce:]FUNCtion:GENerator:WAVE:DATA <value>**<br>**[SOURce:]FUNCtion:GENerator:WAVE:DATA?** | **3.** This will write a value, for example a frequency, to the previously selected parameter, as part of the sequence. Can also be used to query the last value. |
| **[SOURce:]FUNCtion:GENerator:WAVE:SUBmit** | 4. Submits all data. Without sending this command, the FG can be started, but will run with all values being zero |

*The AC and DC start and end values have a dependency from each other. Rule of thumb: use of the AC part requires to set DC values first, else the AC values are not accepted by the device and an error is put into the SCPI error buffer. The DC values (start, end) must not be smaller than the related AC values (start, end).*
*It can be useful to read back a value that was just written to the device, in order to verify wether it has been accepted or not. Alternatively, you may read the error queue.*

When adjusting parameters for the arbitrary function generator manually on the device's control panel, they are limited to each so the resulting signal will work as expected. But here, in remote control, there will be no plausibility check and so it's up to the user to write correct values.

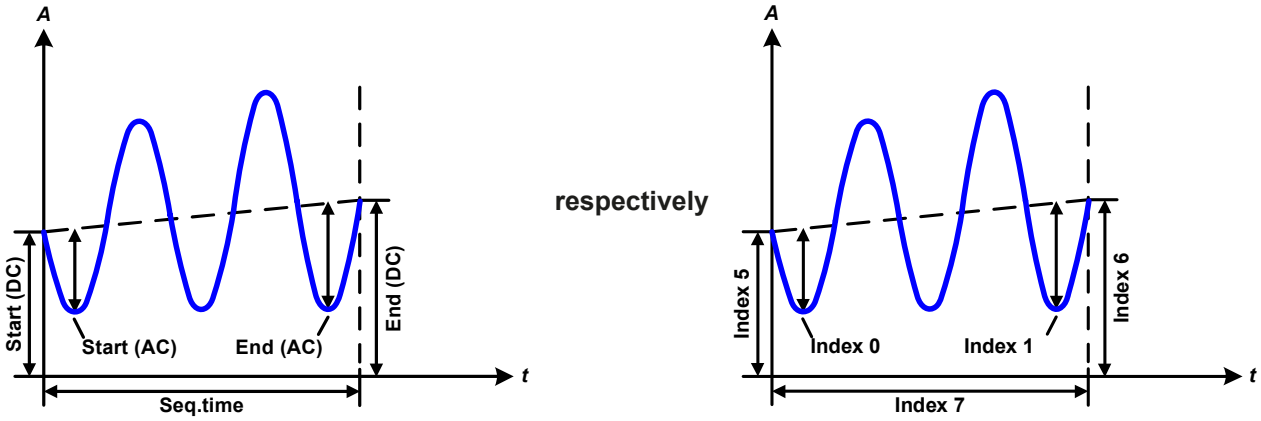
For example, index 0 is connected to index 5, as the DC value is the base line of the AC amplitude. It means, if you, for instance, want to achieve a sine wave with 5 A amplitude on the DC input current of an electronic load, the base line of the resulting sine wave has to be at minimum 5 A, else the negative wave will be clipped at 0. Indexes 5 and 6 are adjustable DC offsets which move the AC wave's base line on the Y axis. So the values in indexes 5 and 6 should at least be as high as index 0 or 1 (whichever is bigger), but they can also be higher. See figures below.

In relation to the adjustments for a function as they can be done on the device's front panel, following indexes are selectable and readable/writable with sub command **:DATA**.

| Index | Parameter | Value range | Note |
|---|---|---|---|
| **0** | Start value(AC part) in A or V | 0...Nominal value of U or I | For AC part only |
| **1** | End value (AC part) in A or V | 0...Nominal value of U or I | For AC part only |
| **2** | Start frequency in Hz (integer) | 0...1000 | For AC part only |
| **3** | End frequency in Hz (integer) | 0...1000 | For AC part only |
| **4** | Start angle in ° (integer) | 0...359 | For AC part only |
| **5** | Start level (offset) in A or V | 0...Nominal value of U or I | For AC and DC part |
| **6** | End level (offset) in A or V | 0...Nominal value of U or I | For AC and DC part |
| **7** | Sequence time in seconds | 0.0001...36000 | |

*In case start and end value (indexes 0+1 and indexes 5+6) are not equal, the device expects a certain minimum change of ±0.058%/s resp. of ±9.3 Hz/s for the start and end frequency (indexes 2+3) over the sequence time. It is thus, for example, not possible to let the input current rise by 1 A over 1 h (ramp), because this exceeds the internal set value resolution by far. Another example: with the sequence time being set to 2 s, a start frequency of 1 Hz and end frequency of 10 Hz would not be accepted, because difference only 9 Hz/s, but start frequency of 30 Hz and end frequency of 5 Hz would.*

Parameter assignment illustrated by an example curve:




### 5.4.12.3 Load table data (XY generator)

The XY generator is based on a table with 4096 values for 0...125% of the referenced physical value. It is expected to write the full table. Refer to the operating manual of your device about further details of the XY function generator.

It is important to decide whether the table is going to be used for a UI or for a IU function, because the device will interpret those 4096 values either as voltage or current values. For example, the table might contain values up to 170 A for a device ELR 9080-170 with 80 V/170 A, which would be acceptable for a IU function, but not for UI function, because then the table would be interpreted as voltage values where 80 V is maximum and will lead to wrong results.

| Command | Description |
|---|---|
| **[SOURce:]FUNCtion:GENerator:XY:LEVel {0...4095}** | **1.** Select one out of 4096 table entries for writing or returns the currently selected entry number. |
| **[SOURce:]FUNCtion:GENerator:XY:DATA <value>**<br>**[SOURce:]FUNCtion:GENerator:XY:DATA?** | **2.** Writes a value, for example a voltage value, to the previously with **:LEVel** selected table entry or returns the value. |

### 5.4.12.4 Control of the XY function generator

After the configuration of the XY generator and after all necessary table data has been loaded it can be started by simply switching the DC output/input of the device on. This is the actual function run and is only stops due to device alarms or abortion by the user.

### 5.4.12.5 Control of the arbitrary function generator

Contrary to the XY generator where the curve is immediately active when switching on the DC output/input, the arbitrary generator requires run control by command. The run cannot be paused. It means, once the function is stopped, no matter by what reason, the next start will run from the beginning, the first sequence in use.

| Command | Description |
|---|---|
| **[SOURce:]FUNCtion:GENerator:WAVE:STATe {RUN \| STOP}**<br>**[SOURce:]FUNCtion:GENerator:WAVE:STATe?** | Starts/stops the arbitrary function generator or queries the STATe |

#### 5.4.12.6 Special function: PV (photovoltaics)

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| – | – | – | ✓ | – | – | – |

The photovoltaics function (PV), as available in the function generator of series PSI 9000 2U/3U uses the standard XY generator to calculate a complete table of 4096 values for the XY function generator from a set of only four values (open circuit voltage Uoc, short-circuit current Isc, voltage and current of the maximum power point Umpp/Impp). During function run, the irradiation value, which simulates different light situations, can be adjusted:

| Command | Description |
|---|---|
| **[SOURce:]IRRadiation <NR1>**<br>**[SOURce:]IRRadiation?** | Adjust or queries the irradiation value during the solar panel simulation in a range of {0...100} per cent, which affects the DC current and shifts the MPP on the Y axis |

With SCPI or ModBus remote control, the PV function is partially supported. Following restrictions apply:

- There are no commands to generate the resulting curve from the four parameters
- There are no commands to enter those four parameters remotely

You have two options to calculate the XY table for use in remote control:

1. You do it externally and then load all data into the device.

2. You use the HMI, enter the four parameters, let it calculate the table and transfer the data to the internal regulation circuit, then leave function generator mode again and start remote control. With this option you could also read the entire table from the device by commands and store it in a file for later use or directly store the function into a file from the HMI using the USB port.

Given that you already switched the device to remote control and that the XY table for the PV function is already calculated and the data is available, following procedure can be used:

**► How to setup the device for PV function**

| No. | Command | Description |
|---|---|---|
| **1** | **FUNC:GEN:SEL PV** | Select the PV function for the XY generator. By sending this command the device will switch to function generator mode |
| **2** | **FUNC:GEN:XY:LEVEL 0** | Write all XY table, ie. PV function data to the device |
| **3** | **FUNC:GEN:XY:DATA 150** | |
| **...** | | |
| **8192** | **FUNC:GEN:XY:LEVEL 4095** | |
| **8193** | **FUNC:GEN:XY:DATA 0** | |
| **8194** | **FUNC:GEN:XY:SUB** | Submit all data |

After submitting all data, it is necessary to wait some time (~1 s) before starting to run the function. Optionally, if not already done, set additional global limits like voltage and power, either to maximum or any other value that does not interfere the PV function run:

| No. | Command | Description |
|---|---|---|
| **8195** | **VOLT MAX** | Set voltage to max, independent from the model |
| **8196** | **POW MAX** | Set power to max, independent from the model |

After all is set, you can run the function and control the simulation and irradiation. The irradiation then acts as a factor that is multiplied to the current value that is read from the table, so changing this value moves the power point vertically on the Y axis. For an illustration of the PV curve, refer to your PSI 9000 series device manual.

**► How to control the device during the PV function run**

| No. | Command | Description |
|---|---|---|
| **8197** | **OUTP ON** | Switch the DC output of your device on to make the function start |
| **8198** | **IRR 85** | Set irradiation to 85% (example) or any other value between 0 and 100 |
| **8199** | **OUTP OFF** | Switch the DC output of your device off to make the function stop |
| **8200** | **FUNC:GEN:SEL NONE** | Parameter **NONE** selects no function generator type and leaves the function generator mode |

#### 5.4.12.7 Special function: FC (fuel cell)

| ELR9 | EL9 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| – | – | – | ✓ | – | – | – |

The fuel cell function (FC), as available with the function generator of series PSI 9000 2U/3U uses the standard XY generator to calculate a complete table of 4096 values for the XY function generator from a set of four supporting points (P1-P4) on a fuel cell simulation curve. During function run nothing related to the FC simulation can be adjusted. With SCPI or ModBus remote control, the FC function is only supported by using the general XY generator. Following restrictions apply:

- There are no commands to generate the resulting curve from the four points
- There are no commands to enter those four parameters remotely

The user has two options to calculate the XY table for use in remote control:

1. You do it externally and then load all data into the device. The procedure to set upt the XY generator is described in *„5.4.13.2. Command sequence for the XY generator“* can be used. For more details about FC function refer to the PSI 9000 series device manual.

2. You use the HMI, enter the four points, let the device calculate the table and transfer the data to the internal regulation circuit, then leave function generator mode again and start remote control. With this option you could also read the entire table from the device by command and store it in a file for later use or directly store the function into a file from the HMI using the USB port.

### 5.4.13 Programming examples for the function generator

#### 5.4.13.1 General command sequence for the arbitrary generator

Let's say you want to apply a sine wave with 30 A amplitude and 10 Hz frequency for 60 s to the DC input current of an electronic load. This can be achieved by setting up just one sequence. Let's use sequence number 12. Because this is about DC current, the amplitude also requires an offset. The amplitude is usually understood as the difference between the base line, which is here defined by Start(DC) and End(DC) values, and the top value of the sine wave. In this example, the offset then has to be at least 30 A, so let's say 50 A. This will result in a DC input current varying sinusoidally between 20 A and 80 A.

The sine wave, when applied to DC voltage or current, emulates AC characteristics and thus requires to set at least indexes 0, 1, 2, 3, 5, 6 and 7, according to the table above. As long as no specific start angle is required, index 4 can be skipped, because the default value is 0°.

Given that the device is already in remote control and the DC input resp. DC output is off, following command sequence would be necessary:

| No. | Command | Description |
|---|---|---|
| 1 | **FUNC:GEN:SEL CURRENT** | Selects arbitrary generator for current. By sending this command the device will switch to function generator mode |
| 2 | **FUNC:GEN:WAVE:LEV 12** | Selects the 12th sequence for writing values |
| 3 | **FUNC:GEN:WAVE:IND 5** | Select index 5: Start value of DC part or AC offset |
| 4 | **FUNC:GEN:WAVE:DATA 50** | Set wave offset to 50 A |
| 5 | **FUNC:GEN:WAVE:IND 6** | Select index 6: End value of DC part or AC offset |
| 6 | **FUNC:GEN:WAVE:DATA 50** | Set wave offset to 50 A. If the offset shall not change during the function run, end and start value have to be identical. |
| 7 | **FUNC:GEN:WAVE:IND 2** | Select index 2: Start frequency of sine wave |
| 8 | **FUNC:GEN:WAVE:DATA 10** | Set start frequency to 10 Hz |
| 9 | **FUNC:GEN:WAVE:IND 3** | Select index 3: End frequency of sine wave |
| 10 | **FUNC:GEN:WAVE:DATA 10** | Set end frequency to 10 Hz. If the frequency shall not change during the function run, end and start value have to be identical. |
| 11 | **FUNC:GEN:WAVE:IND 0** | Select index 0: Start value of sine wave amplitude |
| 12 | **FUNC:GEN:WAVE:DATA 30** | Set amplitude to 30 A |
| 13 | **FUNC:GEN:WAVE:IND 1** | Select index 1: End value of sine wave amplitude |
| 14 | **FUNC:GEN:WAVE:DATA 30** | Set amplitude to 30 A. If the amplitude shall not change during the function run, end and start value have to be identical. |
| 15 | **FUNC:GEN:WAVE:IND 7** | Select index 7: Sequence time |
| 16 | **FUNC:GEN:WAVE:DATA 60** | Set sequence time to 60 s |
| 17 | **FUNC:GEN:WAVE:START 12** | Set starting sequence to 12 |
| 18 | **FUNC:GEN:WAVE:END 12** | Set end sequence to 12 too, because only one sequence is used. |
| 19 | **FUNC:GEN:WAVE:NUM 1** | Set number of sequence cycles to 1, because the sequence will already for 60 s. Alternatively, it is possible to define 1 s for the sequence time and let the sequence run through 60 cycles. |
| 20 | **FUNC:GEN:WAVE:SUBMIT** | Load the parameters from above into the function generator |

Now you also need to set the three global set values "U/I/P Limits", as you would set them in manual control. This is required, because in function generator mode, the set values of U, I and P from normal operation are not used. In this example with a current sink (i.e. electronic load), it is recommended to set the voltage to 0 V, the power to maximum and the current to 105% or higher of the peak that would result from the sine wave current.

| No. | Command | Description |
|---|---|---|
| 21 | **VOLT 0** | Set voltage to 0 V, so the device can clearly operate in current control mode |
| 22 | **CURR 88** | The current peak of this example is calculated as 80 A, so we set 110%, means 88 A |
| 23 | **POW MAX** | Power to max, independent from the model |

The function generator is now configured and sequence 12 is set up. You may start to function generator now and control it remotely:

| No. | Command | Description |
|---|---|---|
| 24 | **INP ON**<br>**OUTP ON** | Switch the DC input resp. DC output of your device on |
| 25 | **FUNC:GEN:WAVE:STAT RUN** | Start the function with RUN. After 60 s, the function will stop. |
| 26 | **FUNC:GEN:WAVE:STAT STOP** | Stop/abort function run. The DC input/output will remain on at first and can be switched off with the dedicated command, if necessary |
| 27 | **FUNC:GEN:SEL NONE** | Parameter **NONE** will leave function generator mode |

#### 5.4.13.2 Command sequence for the XY generator

Configuration and loading of table data for XY generator is very similar to the procedure of the arbitrary generator. Let's say you want to make the DC input current of an electronic load react to the input voltage. This is were the XY generator suits well with its IU function.

The IU table with its data determines the current to draw from the source for the entire input voltage range (0...125% $U_{Nom}$), which is resolved in 4096 values. With this you can define everything you want, like for example the current to remain 0 A below a certain input voltage threshold. The desired current curve could be created in Excel or similar tools and exported as CSV file. Because the measurement range for the reference value is defined as 0...125%, but for the depending value it is only 0...100%, the 100% for the depending value is already at table entry 4096/1.25 = 3276.

There is furthermore a global power limitation, so the device can not make 100% voltage at 100% current. When creating the table in Excel or similar, it might help to add another two columns (which are later not exported to CSV, of course). One column, where the referenced value is distributed between 0....125% nominal value over the 4096 entries, and another where the power is calculated for every entry (P = U * I), to find out which of the entries could not be physically realised by the device.

*Caution! It is not advisable to have big value differences between two or a group of consecutive table entries. Rather use values that make voltage/current change "softly".*

| No. | Command | Description |
|---|---|---|
| 1 | **FUNC:GEN:SEL IU** | Select the IU function for the XY generator use: I = f(U). By sending this command the device will switch to function generator mode |
| 2 | **FUNC:GEN:XY:LEVEL 0** | Select table entry 0 for writing |
| 3 | **FUNC:GEN:XY:DATA 0** | Write a current value to the table entry, here: 0 A (random value) |
| ... | | |
| 8192 | **FUNC:GEN:XY:LEVEL 4095** | Select table entry 4095 for writing |
| 8193 | **FUNC:GEN:XY:DATA 120** | Write a current value to the table entry, here: 120 A (example) |
| 8194 | **FUNC:GEN:XY:SUBMIT** | Submit all data and send to the internal controller |

Now it is advised to also define the set values which are not affected by the table, else the function would run without any effect. It means, if you load an UI table, the voltage is set from values in the table, but current and power are static and the values you can adjust for U, I and P in normal operation are not effective here.

For an IU table, voltage and power are static. You can set the static values to any value you like, but in order for the static set values not to interfere in the UI or IU function run, it is recommended to set both to maximum:

| No. | Command | Description |
|---|---|---|
| 8195 | **VOLT MAX resp. CURR MAX** | Set voltage, for an IU function, respectively current, for an UI function, to maximum |
| 8196 | **POW MAX** | Power to max, independent from the model |

After this, the function generator is configured and the IU table is loaded. Now the function can be started by remotely controlling the generator:

| No. | Command | Description |
|---|---|---|
| **8197** | **INP ON**<br>**OUTP ON** | Switch the DC input resp. DC output of your device on |
| **8198** | **INP OFF**<br>**OUTP OFF** | Switch the DC input resp. DC output of your device off to make the function stop |
| **8199** | **FUNC:GEN:SEL NONE** | Parameter **NONE** selects no function generator type and leaves the function generator mode |

#### 5.4.13.3 Command sequence to generate a rising ramp

Before you can configure the arbritary generator for a ramp it is necessary to think about the best way to achieve the ramp generation. It is important to keep in mind that the arbitrary generator stops at the end of the function run, unless you set the repetition to infinite. After a stop, the DC input/output remains switched on. In case of a ramp, this is wanted, because the end value shall usually remain set for time x. However, the device will go to static mode again, setting the static set values of U, I and P. The static values also apply for the period before the function run and when the DC output/input is already switched on.

The stop action and the static values are thus a little problematic for the ramp function. Why? Supposed that you wanted to have a power supply generate a ramp starting from 0 V. The static value for U (voltage) would then be set to 0. But after the function stop, the device would also set 0 V and the voltage would drop from whatever value has been set during the function run. Conclusion: the static value of voltage has to part of the function.

In order to achieve this, the function has to consist of two parts: one for the rising or falling ramp and the other for the static value. This can be done using two sequences of the arbitrary generator.

Assumption: the ramp shall start from 0 V and rise to 50 V within 6 seconds. The end voltage shall remain constant for 3 minutes (the time can be varied at will). We are going to use sequences 1 and 2. Remote control is already active, we only need to configure. Since the ramp will make the voltage rise linearly, using only the DC part of a sequence, the parameters for the AC part (indexes 0 - 4) should be set to zero, in order to avoid remainders of wrongly set AC parameters to disturb the correct wave generation. The commands need for this are not listed for this example.

Sequence 1, rising ramp

| No. | Command | Description |
|---|---|---|
| **1** | **FUNC:GEN:SEL VOLTAGE** | Selects arbitrary generator for voltage. By sending this command the device will switch to function generator mode |
| **2** | **FUNC:GEN:WAVE:LEVEL 1** | Select sequence 1 for writing |
| **3** | **FUNC:GEN:WAVE:IND 5** | Select index 5: Start voltage of the ramp |
| **4** | **FUNC:GEN:WAVE:DATA 0** | Set start voltage to 0 V |
| **5** | **FUNC:GEN:WAVE:IND 6** | Select index 6: End voltage of the ramp |
| **6** | **FUNC:GEN:WAVE:DATA 50** | Set end voltage to 50 V |
| **7** | **FUNC:GEN:WAVE:IND 7** | Select index 7: Ramp duration |
| **8** | **FUNC:GEN:WAVE:DATA 6** | Set 6 seconds |

Sequence 2, the static voltage at the ramp end

| No. | Command | Description |
|---|---|---|
| **9** | **FUNC:GEN:WAVE:LEVEL 2** | Select sequence 2 for writing |
| **10** | **FUNC:GEN:WAVE:IND 5** | Select index 5: Start value of the static voltage (ramp without slope) |
| **11** | **FUNC:GEN:WAVE:DATA 50** | Set start voltage to 50 V |
| **12** | **FUNC:GEN:WAVE:IND 6** | Select index 6: End value of the static voltage (ramp without slope) |
| **13** | **FUNC:GEN:WAVE:DATA 50** | Set end voltage to 50 V |
| **14** | **FUNC:GEN:WAVE:IND 7** | Select index 7: Duration |
| **15** | **FUNC:GEN:WAVE:DATA 180** | Set to 3 minutes (180 seconds) |

Configure the arbitrary generator

| No. | Command | Description |
|---|---|---|
| 16 | **FUNC:GEN:WAVE:START 1** | Set sequence 1 as start sequence |
| 17 | **FUNC:GEN:WAVE:END 2** | Set sequence 2 as end sequence |
| 18 | **FUNC:GEN:WAVE:NUM 1** | Number of cycles |
| 19 | **FUNC:GEN:WAVE:SUBMIT** | Submit all data |

The number of cycles is set to 1, so the function runs once and then stops. The value can be changed at will, but with every repetition after 3 m 6 s, the voltage of the ramp would have to drop from 50 V to 0 V, where the ramp shall start. But it cannot drop in no time. How long it takes primarily depends on the connected load (supposed you are using a power supply). The resulting ramp could be malformed more or less. In order to avoid that a third sequence could be configured which just gives the voltage some time to drop.

After this the ramp function is fully configured and can be started. If the DC output/input is not yet switched on, it will be automatically switched on when running the function. Alternatively, switching it on can also be done separately with the corresponding commands. Here it is not required, because the function starts from 0 V, but in case a function shall not start at 0, it would be necessary to switch on the DC output/input first.

| No. | Command | Description |
|---|---|---|
| 20 | **FUNC:GEN:WAVE:STAT RUN** | Run the function generator |

Without repetition the function would stop after one run and after the time defined in sequence 2 (ignoring the duration of sequence 1 here, because only 6 s) and the voltage would drop to zero. If you wanted to have the static value remain set for much longer, you would probably need to include the remaining 98 sequences. With one sequence, you can achieve a duration of 10 h, so the static value could remain set for a maximum of 99 x 10 h = 990 h.

### 5.4.14 Commands for remote control of the sequence generator

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| – | ✓ | – | – | – | – | – |

*Sequence data written with SCPI commands is permanently stored inside the device and is also connected to the sequence data which you can edit on the front panel (HMI).*

The sequence generator of the ELR/ELM 5000 series is a simplified version of the arbitrary function generator of other series, but the handling is almost identical.

When operating the sequence generator on the control panel of the device, it enforces a certain procedure of setup before getting to the actual starting point. The single commands can not enforce that procedure, so it's up to the user to use them in the correct sequence. What to do:

**1)** Configure the sequence generator. There are two items. One is to load the data for the sequence points, at least if different settings than stored are going to be used. The other settings, like start sequence point etc., are recommended to always be configured.

**2)** Run and control the sequence generator.

#### 5.4.14.1 Configuration of the sequence generator

| Command | Description |
|---|---|
| **[SOURce:]FUNCtion:GENerator:WAVE:STARt {1..100}**<br>**[SOURce:]FUNCtion:GENerator:WAVE:STARt?** | Defines the sequence point (1...100) to start from or queries the last setting. If only one sequence point is used, then it must be **:STARt** = **:END**. Sequence points going to be used should also be written with data before using them. |
| **[SOURce:]FUNCtion:GENerator:WAVE:END {1..100}**<br>**[SOURce:]FUNCtion:GENerator:WAVE:END?** | Defines the sequence point (1...100) to stop at or queries the last setting. If only one sequence point is used, then it must be **:STARt** = **:END**. |
| **[SOURce:]FUNCtion:GENerator:WAVE:NUMber {0..999}**<br>**[SOURce:]FUNCtion:GENerator:WAVE:NUMber?** | Defines, how often the sequence block from **:STARt** to **:END** is cycled through or queries the last setting.<br>0 = infinite cycles<br>1...999 = number of cycles |

#### 5.4.14.2 Control the arbitrary function generator

| Command | Description |
|---|---|
| **[SOURce:]FUNCtion:GENerator:WAVE:STATe[?] {RUN \| STOP}**<br>**[SOURce:]FUNCtion:GENerator:WAVE:STATe?** | Starts/stops the sequence generator or queries the state. With the start of the generator in remote control, the display of the device will automatically switch to sequence generator mode. |

#### 5.4.14.3 Load sequence point data

Sequence data should can always be sent to the device while the DC input is switched off. There is no special mode to activate before.

Which ones of the 100 sequence points are going to be filled with data is determined by the user, as well as the points to start from and to stop at and the number of cycles of the entire sequence. The block of sequence points can be of arbitrary size. When processing a sequence point during the sequence run which has not been filled with data, it would result in an interrupt of the minimum sequence point time of 300 ms, along with values U and I being zero for that moment.

Commands to load sequence point data:

| Command | Description |
|---|---|
| **[SOURce:]FUNCtion:GENerator:WAVE:LEVel {1...100}**<br>**[SOURce:]FUNCtion:GENerator:WAVE:LEVel?** | Selects a sequence point (similar to HMI access) for write actions or queries the number of the currently selected sequence point |

| Command | Description |
|---|---|
| **[SOURce:]FUNCtion:GENerator:WAVE:INDex {0...3}**<br>**[SOURce:]FUNCtion:GENerator:WAVE:INDex?** | For the selected sequence point, a set of 4 parameters has to be configured. This command selects the parameter between 0 and 3 with **:INDex**. The next command (**:DATA)** is then used to write a value. The indexes are explained below. Can also be used to query the current index. |
| **[SOURce:]FUNCtion:GENerator:WAVE:DATA <value>**<br>**[SOURce:]FUNCtion:GENerator:WAVE:DATA?** | This will write a value to the previously selected parameter, as part of the sequence point definition. Can also be used to query the last value. |
| **[SOURce:]FUNCtion:GENerator:WAVE:SUBmit** | Submits all data. Without sending this command, the sequence generator would use formerly stored data. |

These commands would be repeated for every sequence point to load. When using the full set of 100 sequence points it would require to send 900 commands, see steps 1-9 in the example below.

In relation to the adjustments for a function as they can be done on the device's front panel, following parameters are selectable with **:INDex** and readable/writable with **:DATA**.

| Index | Parameter | Value range |
|---|---|---|
| **0** | Sequence point voltage set value in Volt | 0...Nominal value of U |
| **1** | Sequence point current set value in Ampere | 0...Nominal value of I |
| **2** | Sequence point power set value in Watt | 0...Nominal value of P |
| **3** | Sequence point time in seconds | 0.300...36000.000 (i. e. 300 ms ... 10 h), step width 1 ms |

*In case any of the three set values is not written, it keeps the last stored value. It is recommended to always set all three value to ensure the result of the sequence is as expected.*

*The adjustment limits on the HMI ("Limits") do no apply here.*

#### 5.4.14.4 Example command sequence

Assumption: the source shall be loaded with 20 A for 60 seconds. Because the ELM 5000 load module can only take up to 320 W of power, the input voltage must not reach 16 V for this run, else the power would be limited by decreasing the input current below the programmed value. The power is set to maximum.

| No. | Command | Description |
|---|---|---|
| **1** | **FUNC:GEN:WAVE:LEV 12** | Selects the 12th sequence point for writing values, for this example only this particular point is going to be written and used for the sequence run |
| **2** | **FUNC:GEN:WAVE:IND 0** | Select index 0: Voltage set value. |
| **3** | **FUNC:GEN:WAVE:DATA 0** | Set voltage for sequence point 12 to 0 V. Could be 0...80 for an 80 V model. With an electronic load, the voltage is secondary and when running it in CC mode is intended, the voltage would usually be set to 0 |
| **4** | **FUNC:GEN:WAVE:IND 1** | Select index 1: Current set value |
| **5** | **FUNC:GEN:WAVE:DATA 20** | Set current for the sequence point 12 to 20 A. Could be 0...25 for an 80 V model. |
| **6** | **FUNC:GEN:WAVE:IND 2** | Select index 2: Power set value |
| **7** | **FUNC:GEN:WAVE:DATA 320** | Set power of sequence point 12 to 320 W. If the total power shall not be limited to less than nominal for the entire sequence, it is required to set every sequence point's power to maximum, because there is no global power value for the sequence. |
| **8** | **FUNC:GEN:WAVE:IND 3** | Select index 3: Time |
| **9** | **FUNC:GEN:WAVE:DATA 60** | Sets the time to elapse while the set values of sequence point 12 are effective to 60 s or 1 minute. |
| **10** | **FUNC:GEN:WAVE:SUBMIT** | Submit all sequence data |
| **11** | **FUNC:GEN:WAVE:START 12** | Sets the first sequence point of the sequence to 12 |
| **12** | **FUNC:GEN:WAVE:END 12** | Sets the last sequence point of the sequence also to 12 |
| **13** | **FUNC:GEN:WAVE:NUM 1** | Sets 1 cycle, i. e. no repetition |

Result of this setup: when running the sequence generator, no matter if the DC input is already switched on or not, the electronic load will set the values of sequence point 12 (0 V, 20 A, 320 W) and switch on the DC input. The load will then take 20 A for a period of 60 seconds. After the 60 seconds have elapsed, the sequence generator would automatically stop after 1 cycle. With the DC input remaining on after the sequence stop, the load would continue to draw 20 A. This could be unwanted.

In order for the current to be zero after the end of the sequence run, another sequence point could be configured. The chain of commands, as shown above, would change after step 9 like this:

| No. | Command | Description |
|---|---|---|
| **10** | **FUNC:GEN:WAVE:LEV 13** | Selects the 13th sequence point for writing values |
| **11** | **FUNC:GEN:WAVE:IND 1** | Select index 1: Current set value |
| **12** | **FUNC:GEN:WAVE:DATA 0** | Set current for the sequence point 13 to 0 A. |
| **13** | **FUNC:GEN:WAVE:SUBMIT** | Submit all sequence data |
| **14** | **FUNC:GEN:WAVE:START 12** | Sets the first sequence point of the sequence to 12 |
| **15** | **FUNC:GEN:WAVE:END 13** | Sets the last sequence point of the sequence to now 13 |
| **16** | **FUNC:GEN:WAVE:NUM 1** | Sets 1 cycle, i. e. no repetition |

The voltage value and time of sequence point 13 don't matter. The sequence generator would then stop after point 13 with 0 A and DC input switched on. Since no current is flowing, it could be considered as "input off".

### 5.4.15 Commands for remote control of the MPP tracking feature

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| – | ✓ | – | – | – | – | – |

Maximum Power Point (MPP) tracking is something inverters for solar panels use. The MPP tracking emulates the tracking behaviour of such inverters. The feature itself and its modes and settings are described in the device manual. Here only the corresponding commands for remote setup and control are explained.

#### 5.4.15.1 Configuration of the MPP tracking

The configuration is done with 14 indexes and specific :DATA commands:

| Command | Description |
|---|---|
| **[SOURce:]FUNCtion:GENerator:MPP:INDex 0**<br>**[SOURce:]FUNCtion:GENerator:MPP:DATA[?] {MPP1 \| MPP2 \| MPP3 \| MPP4}** | Index 0: MPP tracking mode selection<br>MPP1: MPP tracking mode 1 (Find MPP)<br>MPP2: MPP tracking mode 2 (Track)<br>MPP3: MPP tracking mode 3 (Fast track)<br>MPP4: MPP tracking mode 4 (User curve) |
| **[SOURce:]FUNCtion:GENerator:MPP:INDex 1**<br>**[SOURce:]FUNCtion:GENerator:MPP:DATA[?] <value>** | Index 1: Set the open circuit voltage<br>Uoc (0-Unom) |
| **[SOURce:]FUNCtion:GENerator:MPP:INDex 2**<br>**[SOURce:]FUNCtion:GENerator:MPP:DATA[?] <value>** | Index 2: Set the short-circuit current<br>Isc (0-Inom) |
| **[SOURce:]FUNCtion:GENerator:MPP:INDex 3**<br>**[SOURce:]FUNCtion:GENerator:MPP:DATA[?] <value>** | Index 3: Set the voltage limit for fast track mode 3<br>Umpp (0-Unom) |
| **[SOURce:]FUNCtion:GENerator:MPP:INDex 4**<br>**[SOURce:]FUNCtion:GENerator:MPP:DATA[?] <value>** | Index 4: Set the current limit for fast track mode 3<br>Impp (0-Inom) |
| **[SOURce:]FUNCtion:GENerator:MPP:INDex 5**<br>**[SOURce:]FUNCtion:GENerator:MPP:DATA[?] <value>** | Index 5: Set the MPP for fast track mode 3<br>Pmpp (0-Pnom) |
| **[SOURce:]FUNCtion:GENerator:MPP:INDex 6**<br><br>**[SOURce:]FUNCtion:GENerator:MPP:DATA[?] <value>** | Index 6: Set ΔP (in Watts), a difference to the MPP above which the tracker starts to find the MPP again<br>0-50 W |
| **[SOURce:]FUNCtion:GENerator:MPP:INDex 7**<br>**[SOURce:]FUNCtion:GENerator:MPP:DATA?** | Index 7: resulting MPP<br>Reads three values (Uact, Iact, Pact) which define the MPP (modes 1, 2 and 4) |
| **[SOURce:]FUNCtion:GENerator:MPP:INDex 8**<br>**[SOURce:]FUNCtion:GENerator:MPP:LEVel[?] {1-100}**<br>**[SOURce:]FUNCtion:GENerator:MPP:DATA[?] <value>** | Index 8: Set voltage values for mode 4<br>1-100: Select the value to set<br>Set value (0-Unom) |

| Command | Description |
|---|---|
| **[SOURce:]FUNCtion:GENerator:MPP:INDex 9**<br>**[SOURce:]FUNCtion:GENerator:MPP:LEVel[?] {1-100}**<br>**[SOURce:]FUNCtion:GENerator:MPP:DATA?** | Index 9: Measured results of mode 4<br>Select measured data to read<br>Read data (three values: Uact, Iact, Pact) |
| **[SOURce:]FUNCtion:GENerator:MPP:INDex 10**<br><br><br><br>**[SOURce:]FUNCtion:GENerator:MPP:DATA[?] {5-60000}** | Index 10: Regulation interval for mode 4 stepping or for next tracking action in the other mode (this parameter is only available in remote control; for manual control it is set to minimum)<br>5-60000 ms |
| **[SOURce:]FUNCtion:GENerator:MPP:INDex 11**<br><br>**[SOURce:]FUNCtion:GENerator:MPP:DATA[?] {1-100}** | Index 11: Start number for mode 4 of the voltage values set with index 8<br>Set start number |
| **[SOURce:]FUNCtion:GENerator:MPP:INDex 12**<br><br>**[SOURce:]FUNCtion:GENerator:MPP:DATA[?] {1-100}** | Index 12: End number for mode 4 of the voltage values set with index 8<br>Set end number |
| **[SOURce:]FUNCtion:GENerator:MPP:INDex 13**<br>**[SOURce:]FUNCtion:GENerator:MPP:DATA[?] {0-65535}** | Index 13: Number of repetitions for mode 4<br>Repetitions of the scan |

#### 5.4.15.2 Control of the MPP tracking

The MPP tracking is started or stopped with a separate command. Independent from the DC input condition of the device it would automatically switch the DC input on when sending RUN.

| Command | Description |
|---|---|
| **[SOURce:]FUNCtion:GENerator:MPP:STATe[?] {RUN \| STOP}** | **RUN** = Runs the MPP tracking in the configured mode<br>**STOP** = Stops MPP tracking anytime, nom matter if there is a positive result or not<br>**?** = Read the tracking status |

The tracking status, as it can be read with command FUNC:GEN:MPP:STAT?, returns the current status as RUN (running) or STOP (stopped). The meaning of STOP slightly differs, depending on the chosen mode:

Mode 1: STOP means, the MPP has been found (positive result) and the tracking has been finished

Modes 2 and 3: These modes don't stop automatically, so STOP only returns the status as set by :STAT command

Mode 4: STOP means, the user curve has been processed the defined number of cycles

### 5.4.16 Commands for alarm management

In remote control operation it is also important to manage alarms correctly. This can be done the same way as in manual control. When using SCPI command language, device alarms are indicated via status register which can be polled. Furthermore, most alarms have to be acknowledged.

#### 5.4.16.1 Reading device alarms

Reading device alarms should happen in certain intervals, by querying the Questionable status register by either the subregister CONDITION or EVENT. The command STAT:QUES? resp. STAT:QUES:COND? or STAT:QUES:EVEN? return a value that represents certain bits (see *„5.4.2 Status registers“ on page 32*), indicating various statuses. When a bit is set, it means a certain alarm is present. Refer to the device's operating manual for details about device alarms.

#### 5.4.16.2 Acknowledging device alarms

In order to make the user take notice of device alarms, they have to be acknowledged after they occurred and vanished again. This will delete those alarms from the status register and should be only be done after they have been recorded. To delete/acknowledge an alarm, the command SYST:ERR? resp. SYST:ERR:ALL? is used, which also serves to query other errors.

In case one or multiple alarms are still present, they won't be cleared from the register.

There is one exception in handling, the OT (overtemperature) error. This doesn't require extra acknowledgement and thus won't be indicated anymore in CONDITION once it is gone.

#### 5.4.16.3 Alarm counters

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

These counters count alarm occurrences since the last time the device was powered. They can be read by command anytime, are not stored when the device is switched off and are purged by reading.

| Command | Description |
|---|---|
| **SYSTem:ALArm:COUNt:OV?** | Counts overvoltage alarms (OVP, adjustable threshold) |
| **SYSTem:ALArm:COUNt:OT?** | Counts overtemperature alarms (OT, not adjustable) |
| **SYSTem:ALArm:COUNt:OP?** | Counts overpower alarms (OPP, adjustable threshold) |
| **SYSTem:ALArm:COUNt:OC?** | Counts overcurrent alarms (OCP, adjustable threshold) |
| **SYSTem:ALArm:COUNt:PF?** | Counts power fail alarms (PF, not adjustable) |

#### 5.4.16.4 Example

You are running the device in remote control and poll the alarm status with STAT:QUES? command in a certain interval and you always receive value 3072. This is the sum of the bit values of bits 10 (remote) and 11 (output/input on). It tells you that remote control is active and the DC output/input is switched on. Then a device alarm occurs caused by the unit overheating. When reading the questionable register the next time, bit 3 should indicate the OT alarm for you to take notice. Additionally, the DC output/input might be indicated as switched off, which does not happen with every device series. Thus the returned value could be 1032 or 3080. Both contain bit's 3 value of 8.

### 5.4.17 Commands for presets (Recall)

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| – | – | – | – | ✓ | – | – |

The Recall feature, as integrated with power supply series PSI 5000 A, can also be configured and used remotely. Here, standard SCPI commands are used and thus it is a little different compared to ModBus. Overview of commands used only for access to the presets:

| Command | Description |
|---|---|
| ***RCL {1...9}** | Recalls, i.e. loads one out of nine presets from the internal storage. The preset consists of four values (voltage, current, OVP, OCP) and overwrites the currently active values on the DC output. The command can only be executed while the DC output is switched off. |
| ***SAV {1...9}** | Save the four currently active values (set value of voltage, set value of current, values of OVP and OCP) on the DC output to the selected preset for future use, when they are recalled using the *RCL {1...9}. The command can only be executed while the DC output is switched off. |
| **MEMory:STATe:DELete {1...9}** | Deletes the selected preset by writing zero to all values. The preset can later be written with custom values anytime. |
| **MEMory:STATe:VALid? {1...9}** | Triggers a memory check to determine if a previously written preset has been written correctly to the internal storage and the values are OK and valid. Will return a 1 of all is OK and a 0, if not OK. |

Similar to manually operating the recall feature on the control panel of the device, these commands are only accepted while the DC output is switched off.

In order to set the four value that can be stored as a preset, common commands like [SOURce:]VOLTage are used. Also see examples below.

#### 5.4.17.1 Example sequence for setting up and saving a preset

For example, you have a power supply PSI 5040-20 A with 49 V nominal voltage and 20 A nominal current. Now you want to set up and save preset number 5 for later recall. Let's say the value to set up are U = 10 V, I = 5 A and overvoltage protection (OVP) = 12 V. The protection is used to prevent the load from high voltage overshoots which can result from a typical regulation characteristics when setting the current to zero while voltage is put out and then releasing the current again, so the voltage "jumps".

The overcurrent protection is of now priority for you, but it is recommended to set it as well, so it won't interfere. There is always a setting which you might not know, so let's set it to maximum (here: 22 A). With all being prepared, following sequence of commands would have to be sent to the device:

| Nr. | Command | Description |
|---|---|---|
| **1** | **OUTP OFF** | Switch the DC output off first (no matter if already off), because saving the preset requires the output to be off and changing the output values with the output being on and with load attached could result in unwanted behaviour. |
| **2** | **VOLT 10** | Set 10 V output voltage (normal set value) |
| **3** | **CURR 5** | Set 5 A output current (normal set value) |
| **4** | **VOLT:PROT 12** | Set OVP threshold to 12 V |
| **5** | **CURR:PROT 22** | Set OCP threshold to 22 A |
| **6** | ***SAV 5** | Store preset 5 |
| **7** | **MEM:STAT:VAL? 5** | (optional) Query, if preset 5 has been stored correctly and values are valid |

#### 5.4.17.2 Example sequence for recalling a preset

During remote control, the four values that are stored in a preset could also be set with four small commands, so that recalling a preset in remote control is something that will be used quite rarely. The big advantage of setting the values directly is that it doesn't matter whether the DC output is switched on or off, they can be sent anytime.

In case you still want to recall a preset, for example number 3, make following steps:

| Nr. | Command | Description |
|---|---|---|
| 1 | **OUTP OFF** | Switch the DC output off first (optional, if already off), so the preset can be recalled |
| 2 | ***RCL 3** | This will recall preset 3 and overwrite the former output values U, I, OVP and OCP with the ones stored in the preset |
| 3 | **OUTP ON** | (optional) Switch DC output on again |

In order to also read what values are stored in a preset, you would have to recall first and anyway, but then you could query the four output values directly to know what a specific preset has stored:

| Nr. | Command | Description |
|---|---|---|
| 1 | **VOLT?** | Queries the output voltage set value, as previously recalled from a preset |
| 2 | **CURR?** | Queries the output current set value, as previously recalled from a preset |
| 3 | **VOLT:PROT?** | Queries the OVP threshold value, as previously recalled from a preset |
| 4 | **CURR:PROT?** | Queries the OCP threshold value, as previously recalled from a preset |

## 5.5 Example applications

### 5.5.1 Configure and control master-slave with SCPI

| ELR9 | ELR5 | PS9 | PSI9 | PSI5 | PSE | DT |
|---|---|---|---|---|---|---|
| ✓ | – | – | ✓ | – | ✓ | – |

Certain device series which feature true master-slave (short: MS) with totals formation via a dedicated master-slave bus also support the full remote configuration and control of the system. In a MS system, usually only the master unit is remotely controlled, while the slaves are not connected to the PC and so cannot be configured remotely. It is thus recommended to configure the MS system on the control panels of the units and only put the master into remote control via any software. Even if you would configure all units manually on the control panel, the remote control software could later read the status of the MS init from the master. The initialisation of the MS system is done automatically every time the master is powered, but can be triggered and repeated by command.

Let's assume following example configuration: five power supplies PSI 9080-510 3U (80 V, 510 A, 15 kW) in parallel.

The master has to display itself as an 80V, 2550 A and 75 kW unit after successful configuration and initialisation. These values are also the nominal values of the MS system, used to limit the values you can send by SCPI command. The same way as with manual control, the "Limits" and set values can be adjusted in 0...102% of nominal, while protection values allow for 0..110%.

The step-by-step guide below is separated into several parts, because some are optional.

**Part 1a: Configure the master**

1. Activate remote control: **SYST:LOCK ON**
2. Activate master-slave mode: **SYST:MS:ENABLE ON**
3. Define the unit as master: **SYST:MS:LINK MASTER**
4. (when running two-quadrants operation and the master in an electronic load):
   Set the master as Share bus slave: **SYST:SHAR:LINK SLAVE**

**Part 1b: Configure the slave (in case it is connected to the PC)**

5. Activate remote control: **SYST:LOCK ON**
6. Activate master-slave mode: **SYST:MS:ENABLE ON**
7. Define the unit as slave: **SYST:MS:LINK SLAVE**
8. Set slave address, for example 5: **SYST:MS:ADDR 5**

If there is more than one slave, repeat steps 4-7 for the other slave(s) with their own addresses.

**Part 2: Initialise the MS system**

**9.** Activate remote contol, in steps 1-7 were not processed, because system was already configure: **SYST:LOCK ON**

**10.** Trigger initialisation, then wait a few seconds: **SYST:MS:INIT**

**Part 3: Further, optional steps**

**11.** Query the initialisation status from the master, in order to analyse it: **SYST:MS:COND?**

**12.** Query the number of units initialised for the MS system (should be 5 with this example): **SYST:MS:UNIT?**

**13.** Query nominal current of the MS system, for later use: **SYST:MS:NOM:CURR?**

**14.** Query nominal power of the MS system, for later use: **SYST:MS:NOM:POW?**

After every successful MS initialisation, all set values, limits and protection values are reset to defaults, so they should be configured to meet your requirements.

*The protection values OVP, OCP and OPP, as well as the events UCD/OCD, UVD/OVD and OPD (refer to the device manual for details) are NOT adapted to the resulting nominal values of the initialised MS system and remain like when remotely controlling a single unit. However, they are transferred to the slave unit(s) by the master and work as expected. You only have to translate the value you actually would like to send to the device by dividing it by the number of units in the MS system. This number can be read from the master with SYST:MS:UNIT?. Clarification: The MS system in this example has a total current of 2550 A. Let's say you wanted the OCP value to be 2000 A. Sending this value to the master would result in an data out of range error. The translated value would have to be sent as 400 A instead (CURR:PROT 400), because there are five units.*

**15.** Configure protection values, for example OCP: **CURR:PROT 400**

**16.** Configure events, for example,

- set OCD to 2100 A, translated for the master it is then 420 A: **SYST:CONF:OCD 420**
- then define the alarm type for OCD to "warning": **SYST:CONF:OCD:ACT WARNING**

The adjustment limits ("Limits") require extra treatment, because they are tied to the set values. Means, with the set values being reset to defaults during the MS init, for example the set value of current would be at maximum and thus the related adjustment limit $I_{Max}$ can't be set lower than this without prior changing the set value.

**17.** Narrow the adjustable range of values, for example limit the max. current set value to 2200 A

- First, set the current value down to anything lower than the desired limit, like the minimum: **CURR MIN**
- Second, set the adjustment limit to the value translated for the master unit (here: 440 A): **CURR:LIM:HIGH 440**

With these settings applied, the current should be at 0, because the lower adjust limit has not yet been changed. The current will me monitored for the threshold of 2100 A by the event system and since it is adjustable up to 2200 A, the true current might exceed the threshold and cause an OCD event, which would only generate a warning on screen, but not switch off the DC output.

**18.** To start working with your MS system, switch the DC output: **OUTP ON**

A system configure like this will remain configured and keep the settings when power-cycling it. The master unit has to initialise the MS and the slaves at least after power-up. This works best if the slaves are powered first and the master at last. In case there is a main switch (e.g. contactor) powering all units at once, the first initialisation attempt might not work correctly and thus the initialisation should be repeated. The status of the first automatic initialisation can be read from the master by custom software and depending on the result, the software could trigger further steps like the ones above, probably from at least step 9 or, if required, even from step 1.

## 5.6 Errors

Errors in terms of SCPI are only communication errors. According to the standard, devices using SCPI do not return errors immediately. They have to be queried from the device. The query can occur directly with the error command (see *5.4.5.4*) or by first reading the signal bit "err" from the STB register (see *„5.4.2. Status registers“*).

The error format is defined by the standard and is made of a string containing a number (the actual error code) and an explanatory text. Following errors strings can be generated by the device:

| Error code / error text | Description |
|---|---|
| 0,"No error" | No error |
| -100,"Command error" | Command unknown |
| -102,"Syntax error" | Command syntax wrong |
| -108,"Parameter not allowed" | A command was sent with a parameter though the command doesn’t use parameters |
| -200,"Execution error" | Command could not be executed |
| -201,"Invalid while in local" | Control command could not be executed, because device is in LOCAL mode |
| -221,"Settings conflict" | Command could not be executed because of the condition of the device (being in MENU etc.) |
| -222,"Data out of range" | Parameter could not be set because it exceeded a limit |
| -223,"Too much data" | Too many parameters per command or too many commands at once |
| -224,"Illegal parameter value" | A parameter not specified for the command has been sent |

# 6. Profibus & Profinet

## 6.1 General

The Anybus interface modules IF-AB-PBUS (Profibus) or IF-AB-PNET (Profinet, 1 or 2 ports) simplify use and implementation of the device to the absolute necessary. The user just has to select the slave address (0...125) for both and for Profinet a few additional network parameters. All other parameters like tags, which can be defined in the device's setup menu or via command, are optional.

This part of the document will only explain how to use the register list for your particular device. The list should be along with this document as PDF file. The list is the reference for the remote access to the device.

The Anybus interface modules define a **DP-V1 slave** for cyclic and acyclic data transmission.

## 6.2 Preparation

For the implementation of the device into a Profibus or Profinet and the enumeration at the master (PLC or similar), a fully configured and wired unit is presumed. The next thing you will usually need is a device description file called GSD (Generic Station Default) for Profibus or a GSDML for Profinet/IO, which is either delivered with the device on the included CD or is available as download from the manufacturer's web site or can be obtained upon request.

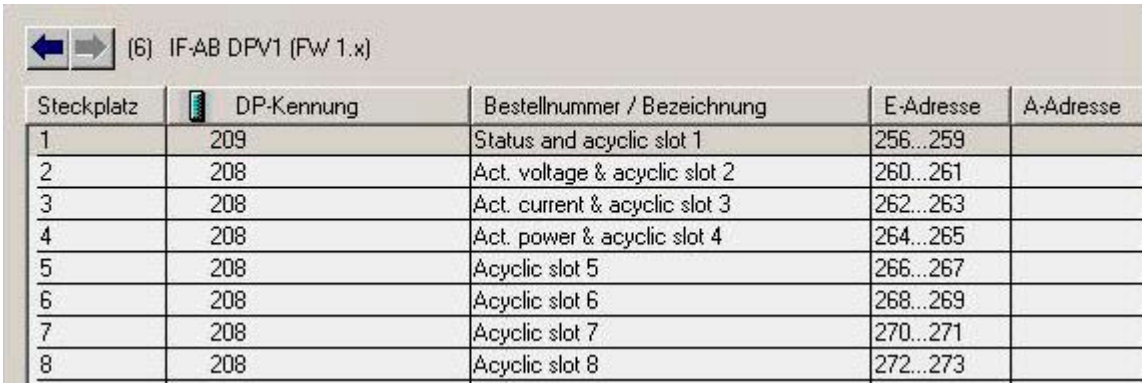
This GSD file describes a certain number of slots to the master application and configures cyclic process data, such as actual values or status. Those slots are also used to access other data objects of the device via acyclic read/write.

## 6.3 Slot configuration for Profibus

The slot configuration for users of the interface module IF-AB-PBUS is typically done by loading the GSD/GSE file in the configuration dialogue (with Siemens STEP7: HWCONFIG). Default slot configuration:

| Slot | Slot name | Description |
|---|---|---|
| 1 | Device status & acyclic slot 1 | Cyclic: Device status (see register list)<br>Acyclic: all registers (indexes) assigned to slot 1 |
| 2 | Act. voltage & acyclic slot 2 | Cyclic: Actual voltage of DC input/output<br>Acyclic: all registers (indexes) assigned to slot 2 |
| 3 | Act. current & acyclic slot 3 | Cyclic: Actual current of DC input/output<br>Acyclic: all registers (indexes) assigned to slot 3 |
| 4 | Act. power & acyclic slot 4 | Cyclic: Actual power of DC input/output<br>Acyclic: all registers (indexes) assigned to slot 4 |
| 5 | Acyclic slot 5 | Acyclic: all registers (indexes) assigned to slot 5 |
| 6 | Acyclic slot 6 | Acyclic: all registers (indexes) assigned to slot 6 |
| 7 | Acyclic slot 7 | Acyclic: all registers (indexes) assigned to slot 7 |
| 8 | Acyclic slot 8 | Acyclic: all registers (indexes) assigned to slot 8 |

Transfer of the above table to the HW CONFIG of Siemens Simatic:


(6) IF-AB DPV1 (FW 1.x)

| Steckplatz | DP-Kennung | Bestellnummer / Bezeichnung | E-Adresse | A-Adresse |
|---|---|---|---|---|
| 1 | 209 | Status and acyclic slot 1 | 256...259 | |
| 2 | 208 | Act. voltage & acyclic slot 2 | 260...261 | |
| 3 | 208 | Act. current & acyclic slot 3 | 262...263 | |
| 4 | 208 | Act. power & acyclic slot 4 | 264...265 | |
| 5 | 208 | Acyclic slot 5 | 266...267 | |
| 6 | 208 | Acyclic slot 6 | 268...269 | |
| 7 | 208 | Acyclic slot 7 | 270...271 | |
| 8 | 208 | Acyclic slot 8 | 272...273 | |

## 6.4 Slot configuration for Profinet

The GSDML/XML (available from the included CD or as download) does not offer automatic slot configuration and thus the module placement has to be done by the user like this:

| Slot | Slot name | Description |
|---|---|---|
| 1 | Input 2 words | Cyclic: Device status (register 505, see register list)<br>Acyclic: all registers (indexes) assigned to slot 1 |
| 2 | Input 1 word | Cyclic: Actual voltage of DC input/output (register 507, see register list)<br>Acyclic: all registers (indexes) assigned to slot 2 |
| 3 | Input 1 word | Cyclic: Actual current of DC input/output (register 508, see register list)<br>Acyclic: all registers (indexes) assigned to slot 3 |
| 4 | Input 1 word | Cyclic: Actual power of DC input/output (register 509, see register list)<br>Acyclic: all registers (indexes) assigned to slot 4 |
| 5 | Input 1 word | Acyclic: all registers (indexes) assigned to slot 5 |
| 6 | Input 1 word | Acyclic: all registers (indexes) assigned to slot 6 |
| 7 | Input 1 word | Acyclic: all registers (indexes) assigned to slot 7 |
| 8 | Input 1 word | Acyclic: all registers (indexes) assigned to slot 8 |

Transfer of the above table to the HW CONFIG of Siemens Simatic:


## 6.5 Cyclic communication via Profibus/Profinet

The Profibus / Profinet slave cyclically transfers process data to certain input addresses of the master, as defined in the GSD for Profibus. For Profinet, this is similar. See *„6.4. Slot configuration for Profinet“*.

Actual value have to be translated according to the procedure as described in *„4.4. Translating set values and actual values“*, while any other data are referenced in those so-called register lists (in relation to the internally used ModBus protocol), which usually should be with this document. The slot names are partially connected to corresponding registers in the lists. For instance, a slot might be named “Actual current”. The name can be found at register 508 in the register list for an ELR 9000 device, for instance. This is also where the register is specified for Profibus/Profinet use.

According to *6.3* and *6.4* there are eight slots for acyclic access to the device, which are assigned a varying number of indexes (see register lists). Using appropriate SFBs, the user can acyclically access the IDs (slot addresses) and indexes by write and read. The additional four slots are only defined to reserve slot address space for acyclic data transfers.

*Set values and settable status are not be transferred cyclically for several reasons. One is the high number ot available registers which cannot be covered by only 16 available slots and the max. data size per slot.*

## 6.6 Acyclic communication via Profibus/Profinet

Acyclic communication with the target device is done by using **slots**, precisely their addresses (ID), and **indexes**, which are accessed by system function blocks for read or write. The SFBs to use here are usually SFB52 and SFB53 when using Siemens software. Other PLC control software offer similar options.

The SFBs require an ID, an index and a parameter as input. The parameter can be a status or a set value, translated to a hexadecimal value according to what's described in *„4.4. Translating set values and actual values“*.

For beginners we offer example projects, one each for Profibus and Profinet, which can be opened with Siemens STEP7 and which shall demonstrate the access to the device with preconfigured data blocks.

The register list for your device or device series has two columns, related to Profibus/Profinet use only. These define slot and index number for a particular command. The necessary parameter is defined in the register lists respectively in *„4.4. Translating set values and actual values“*. Rule of thumb:

- **Commands, where no slot/index is given, are not supported via Profibus or Profinet**

The general procedure to control a device remotely is like this:

1. Activate remote control with the appropriate command (can be denied by the slave, see *„3.2. Control locations“*)
2. Control and monitor your device remotely as long as required
3. Deactivate, i.e. leave remote control

If you just want to record data by reading values from the device, activation of remote control is not necessary. You can send query commands to the device at any time and the device will respond immediately, if the current device situation allows the device to respond at all.

When querying something from the device, the function block will put out the data returned from the device to an output buffer. That data can then be processed.

The field bus ensures that the command is transmitted to the slave unit, otherwise it will generate an error. But it can not verify that the device really accepts the command or already has set the desired value. This can only be verified by reading the value from the device and comparing. Whether a value has been transferred to the device’s DC input / output can not be determined definitely.

In order to send a command from with a typical Profibus/Profinet software, following should be applied:

1. Select the command to use from the register list and read its assigned slot/index values.
2. Determine the I/Q address which is assigned to the particular slot in HWCONFIG and which is basically used to get an ID value. The use of ID, index, slot and subslot are not just different between Profibus and Profinet, but also between the different PLC systems and softwares. The examples below demonstrate the

1. Set ID, index and parameter (set value, status or something else) in decimal or hexadecimal form in the SFB and execute.
2. Process the data returned from the device, if the last command was a query.

## 6.7 Examples for acyclic access

### 6.7.1 Activate/deactivate remote control

Remote control is a device state and not the default one. It has to be activated, i.e. requested by the user before the device can be controlled remotely. Depending the settings and on the state the device is currently in when trying to switch to remote control, the device can deny the request.

**► How to activate or deactivate remote control of your device via Profibus**

1. Use the register list and find the proper command, here: Register 402 - Remote mode.
2. Find the slot and index values for this command in the dedicated columns, here slot 2 and index 1.
3. From the slot configuration read the I/Q address for slot 2 to have the value for parameter "ID", for example 260 (like in the example configs in *6.3* and *6.4*) or DW#16#104
4. The value "Index" from the register list is submitted to the parameter INDEX like this:
   Profibus: INDEX = Index = 1
   Profinet: INDEX = Slot number * 255 + 1 + Index = 510 + 1 + 1 = 512
5. Use a suitable function block in your automation software, for example SFB53.
6. Define the control value to use for this command, as described in the columns “Data” and “Example”:
   0xFF00 = Activate remote control
   0x0000 = Deactivate remote control
7. Configure the function block with ID, INDEX and control value and execute the block. If not somehow inhibited by the device, it should either switch to remote control or back to manual control.

### 6.7.2 Send a set value

Any command that sets something in the device, no matter if value or status, requires activated remote control status. Also see *„6.7.1. Activate/deactivate remote control“* and *„3.2. Control locations“*.

Before you send a value, you first need to select which one you want to set and you also might need to translate it, because via Profibus/Profinet set values are transferred as per cent of the nominal values. Read sections *„4.3. Format of set values and resolution“* and *„4.4. Translating set values and actual values“* for more information.

**► How to set the DC input/output current value**

1. Use the register list and find the proper command, here: Register 501 - Set current value.
2. Find the slot and index values for this command in the dedicated columns, here slot 2 and index 24.
3. From the slot configuration read the I/Q address for slot 2 to have the value for parameter "ID", for example 260 (like in the example configs in *6.3* and *6.4*) or DW#16#104
4. The value "Index" from the register list is submitted to the parameter INDEX like this:
   Profibus: INDEX = Index = 25
   Profinet: INDEX = Slot number * 255 + 1 + Index = 510 + 1 + 24 = 535
5. Use a suitable function block in your automation software, for example SFB53.
6. Define the control value to use for this command, as described in the columns “Data” and “Example”:
   0x0000...0xCCCC (decimal: 52428) = Current 0...100%.
   For a model with, for example, 170 A nominal current and a desired current of 10 A, this would be 1/17 of nominal, thus 52428/17 = 3084 --> 0x0C0C.
7. Put the conrol value 0x0C0C together with ID and INDEX into the function block and execute the block. The device should instantly set 10 A as current limit. This can be verified in the display of the device where it shows the set value of current.

### 6.7.3 Read something

Reading something from the device is always possible, it means that no remote control is required. Apart from the cyclically transferred data, any other available information can be read via acyclic transfer.

**► How to read the actuals values of voltage and current**

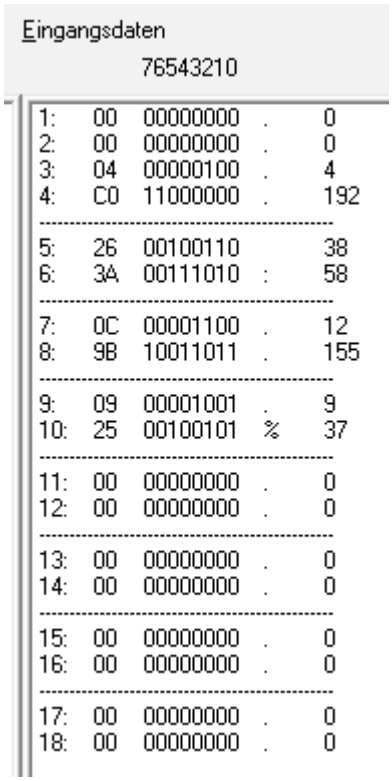
1. Use the register list and find the proper register. The registers of voltage and current are next to each other, the one of voltage is the lower number, thus it will be: Register 507 - Actual voltage
2. Find the slot and index values for this command in the dedicated columns, here slot 2 and index 28
3. From the slot configuration read the I/Q address for slot 2 to have the value for parameter "ID", for example 260 (like in the example configs in *6.3* and *6.4*) or DW#16#104.
4. The value "Index" from the register list is submitted to the parameter INDEX like this:
   Profibus: INDEX = Index = 28
   Profinet: INDEX = Slot number * 255 + 1 + Index = 510 + 1 + 28 = 539
5. Read the length of bytes from the column “Data length in bytes” to determine how many bytes to read. In this case there are two registers with length 2 bytes to read, so it's 4 bytes.
6. Use a suitable function block in your automation software, for example SFB52.
7. Configure the function block with ID, INDEX and data length (4 bytes or 2 word, depending in the way the software defines the input).
8. Execute the function block. The data buffer of the block should return the requested data in form of 20 bytes.

The returned 4 bytes will contain the actual voltage value in the first two bytes and is represented as per cent value (for translation see ). The actual current values will be in the last two bytes.

By varying the data length to 6 you could also include the actual power value. Alternatively, you can query each actual value separately. To do this, you need to use the corresponding register number for parameter INDEX and a data length of 2.

## 6.8 Data interpretation

Data returned from queries, but cyclically transferred data in the first place, have to be interpreted. Let’s use an example from a Profibus master simulator where the cyclic data is comfortably displayed. Also see section *„4.4. Translating set values and actual values“.*




This is the cyclically transferred data of 8 slots, as defined in the GSD/GSDML. Only slots 1-4 are used, so the rest remains empty.

Slot 1: Device status (connected to register 505). The value 0x000004C0 says that bits 6, 7 and 10 are set. It means, the device is configured as master (for master-slave), the input/output is on and regulation mode is CC.

Slot 2: Actual voltage (connected to register 507). With a 250 V model, for instance, the value 0x263A translates to 250 V*0x263A/52428=46.7 V.

Slot 3: Actual current (connected to register 508). With a 510 A model, for instance, the value 0x0C9B translates to 510 A*0xC9B/52428=31.4 A.

Slot 4: Actual power (connected to register 509). For a 5 kW power supply, for instance, the value 0x0925 translates to 5000 W*0x925/52428=223 W or 0.22 kW.

Slot 5: not used for cyclic data

Slot 6: not used for cyclic data

Slot 7: not used for cyclic data

Slot 8: not used for cyclic data

# 7. CANopen

For CANopen available communication objects or registers, indexes are defined in an Electronic Data Sheet file (EDS/XDD), which is delivered with your device on a CD or available as download from the manufacturer's website or upon request. This EDS can be integrated in special CANopen related software. The CANopen indexes are not separately explained, because their definition and use is identical to herein described ModBus protocol and the related, external register list files.

The difference is only that the ModBus register addresses count up from 0 and with CANopen, according to the standard and the interface specification of the manufacturer, the user indexes are located from 0x2001. It means, the register addresses as listed in decimal form in those register list are shifted by the value 0x2001 (8193). Examples from the ModBus part of this document can be used and applied for CANopen as well, but reduced to the core data, because CANopen are not confronted with checksums and function codes as with ModBus.

*The CANopen module IF-AB-CANO does not feature an internal termination resistor. Thus the required bus termination resistor has to be applied by the user according to the CAN bus requirements.*

## 7.1 Preparation

For the communication with the device via CANopen, a few things are required:

1. A suitable CAN cable, preferably with switchable termination resistor, which has to be activated always if the device is at the end of the bus, like when directly connecting the PC to a single ELR 9000 unit
2. EDS/XDD file for a particular device model or for an entire device series (usually provided along with this document)
3. CANopen software for the PC (not included, any available software for CANopen should suffice)
4. Documentation about how to use the supported indexes. See sections *1. - 4.*, *7.2* and *9.*, as well as the included register list(s).

## 7.2 User objects (indexes)

The message format used for CANopen communication is related to ModBus. A specific index reflects a specific ModBus register. The CANopen module manufacturer defines that user objects are enumerated from index 2001. With ModBus, the index are called registers and start from 0. The register addresses of ModBus are used 1:1 for CANopen, but they're shifted by the value of 0x2001. It means, that index 2001 corresponds to register 0 or index 21F5 corresponds to register 500 etc. Along with this document there usually are so-called register lists for primary ModBus use, but these can also be used for CANopen, as they also define data type and value range of the indexes. Examples in other sections of this documents can be applied for CANopen as well.

### 7.2.1 Translation ADI -> register

The translation of an CANopen index, as listed in the EDS file, to a register address is quite easy due to the fixed offset 0x2001. For example, if you pick the index "207A Nominal voltage" from the EDS, it translates like this:

Index number - Offset = register address --> 0x207A - 0x2001 = 0x79 (hex) = 121 (dez). According to the register list for an ELR 9000 device, this represents the nominal device voltage as a FLOAT value. Because CANopen does not support the data type FLOAT, the EDS uses UNSIGNED32 here. The user just has to translate the 32 bit value according to IEEE 754 specification.

### 7.2.2 Specific examples

#### 7.2.2.1 Switching to remote control

As described in *„4.8.9.5. Switch to remote control or back to manual control"*, it is required to switch the device to remote control before you can control it. In order to do this, you first need to find the proper command, i.e. register in the register list resp. the dedicated index in the EDS. In this case, it is register 402 resp. index 0x2193. The register list defines that the value 0xFF00 has to be sent to switch to remote or value 0x0000 to leave remote control.

#### 7.2.2.2 Setting a set value

After remote control has been accepted by the device, you are allowed to send set values. Those values usually represent a per cent value. From the definition in the register list, the hexadecimal value 0xCCCC translates to 100% and 0x0000 to 0%. It means, there are 52428 possible values between 0% and 100%. It has to be pointed out here, that this is not the resolution a device value like voltage or current can have at the DC input/output. The effective resolution of output/input values is 26214 steps. An example for set value translation is in *„4.8.9.1. Writing a set value"*.

# 8. CAN

This section is solely dedicated to the communication with a device via the CAN interface IF-AB-CAN. Configuration of the interface itself is done on the control panel (HMI) of the device.

## 8.1 Preparation

Communication with the device via CAN module IF-AB-CAN requires a few things:

1. A suitable CAN cable. It is not required to have one with integrated bus termination switch and resistor, because the interface module has an electronic switch and resistor for bus termination. In case the cable also has one, it is important to take care to activate only one of both, else there can be bus errors.
2. When using Vector™ or similar software which can make use of so-called database files (DBC), a dedicated DBC for the particular device model. If not available, it can be requested from the manufacturer or created by the user.
3. CAN software for the PC (not included, any available software for CAN should suffice).
4. Documentation about how to use the supported CAN objects. See below and sections *1. - 4.*, as well as the included register list(s).

## 8.2 Introduction

The data format is derived from the previously in this document described ModBus RTU. In relation to a database file (DBC) a mux value (Vector terminology) represents a specific ModBus register or object/command. Objects in the database are thus selected by the muxer and when programming CAN directly, the first two bytes of data in a CAN message define the object to access. The selection between writing and reading objects is done by the CAN ID.

Each device will be assigned three CAN IDs, which are adjusted with the so-called base ID on the device's CAN settings. The base ID is used write to objects (message: Send_Object), while querying objects (message: Query_Object) is done with base ID +1 and responses (message: Read_Object) coming from the device use base ID + 2.

Responses are either expected after a query, but can also be unexpected in case of communication or access error. When adjusting the base ID of a device, the other IDs will shift automatically.

There is another adjustable ID, the broadcast ID. It is separate from the others and can be used to access multiple devices at once with one command when adjusting the broadcast ID to the same value on all units. This ID is for write access (Send_Object) only. Querys to multiple devices at once with one message are not possible.

## 8.3 Message formats

*Below explanations are, besides the selection of IDs to switch between write and read actions, also related to the ModBus functions, as listed in the register lists in columns 2-6.*

### 8.3.1 Normal sending (writing)

Writing to the device always used the base ID or the broadcast ID. It requires to define the first register/object to write to in the CAN data, as well as the number of registers to write and a specific number of parameter bytes which can represent different data types.

**Access:** Base ID, broadcast ID

**ModBus function:** Write Single Coil (WSC), Write Single Register (WSR)

| Bytes 0+1 | Byte 2 | Bytes 3+4 |
|---|---|---|
| Register | Nr. of regs to write | Data |
| 0...65534 | Always 1 | Value (16 bit) |

**Access:** Base ID, broadcast ID

**ModBus function:** Write Multiple Registers (WMR)

| Bytes 0+1 | Byte 2 | Byte 3 | Bytes 4-7 |
|---|---|---|---|
| Start reg. | Nr. of regs to write | Marker | Data bytes |
| 0...65534 | 2...123 | 0xFF, 0xFE... | Four bytes or two 16 bit values or one 32 bit value |

**Start register:** always the register number from the register list, i. e. start register, even for WMR.

**Nr. of regs to write:** refer to the register list. Every object has a starting register and a certain number of total registers which form the object. An object defined with 40 bytes occupies 20 registers, so when writing to such an object the value here would have to be 20.

**Marker:** is used to distinguish single messages from split messages and to detect the correct sequence of data. For example, a string like the user text can be up to 40 characters long and when writing it has to be split across multiple messages. Every message can transport 4 bytes of register data. The marker always starts with 0xFF and is counted downwards (0xFF, 0xFE...) with every next split message belonging to a transmission. The marker is required, because on CAN bus it is not guaranteed that messages are received in the same order they were sent.

**Data bytes:** the number of bytes in this type of message is always 4, no matter if all bytes are filled with information from the actual data to transmit or are 0. An example: an user text with a length of 15 characters would require to send at least 4 messages. The object for the user text is defined to have 20 registers, means 10 messages. You can choose to either write the full 40 bytes while the rest of bytes in the transmission would be zero or to reduce the number of message to the minimum of 4, i. e. a value of 8 for **Nr. of regs to write**.

### 8.3.2 Cyclic sending (writing)

Cyclic sending or cyclic writing is very similar to normal sending, but it's more time effective and intended for often used objects like set values. It offers the possibility to send all four set values to the device at once. It requires two extra CAN IDs to be reserved. The user defines the interval of send actions with the CAN software, no matter if normal or cyclic sending. But there are also limits. The timing as described in section *3.3.3* also applies here.

In order to use this feature, the user only has to define the separately adjustable "Base ID Cyclic Send" and can then send two different messages with following format:

**Access:** Base ID (Control)

| Bytes 0-1 |
|---|
| Control word |

Control word definition:

| Bit | Name | Related register | Meaning |
|---|---|---|---|
| 0 | Remote control | 402 | Activates remote control of the device with 1 or deactivates it with 0 |
| 1 | Eingang/Ausgang | 405 | Switches the DC input/output of the device on with 1 or off with 0 |
| 2 | UIP / UIR | 409 | Activates resistance control mode (UIR) with 1, while with 0 mode UIP will be active |
| 3 | Spannungsregler | 422 | Only available with electronic loads: switches the speed of the internal voltage controller between „fast“ (1) „slow/normal“ (0) |
| 4 | Alarme | 411 | A 1 acknowledges all currently acknowledgeable alarms |

*This control word requires special attention, as the 5 bits can trigger several actions at once which don't have a certain priority of processing. It means, if you would try to activate remote control together with switching on the DC input/output (bits 0 and 1 both TRUE), you may receive a settings conflict error, because the device would possibly process bit 1 before bit 0.*

**Access:** Base ID Cyclic Send + 1 (Set values)

| Bytes 0-1 | Bytes 2-3 | Bytes 4-5 | Bytes 6-7 |
|---|---|---|---|
| **Register 500** | **Register 501** | **Register 502** | **Register 503** |
| Set value of voltage | Set value of current | Set value of power | Set value of resistance |

### 8.3.3 Querying

Querying an object is the first part of a read action. It is always done via base ID + 1. The device should then respond via base ID + 2 (Read_Object) and with the expected data. Only after reading the response, the read action is finished. In order to query an object via the query ID (base ID +1) it is sufficient to just sent the start register number.

**Access**: Base ID + 1

**ModBus functions**: Read Coils (RC), Read Holding Registers (RHR)

| Bytes 0+1 |
|---|
| Start reg. |
| 0...65534 |

### 8.3.4 Normal reading

Data coming from the device can be one message (expected or error) or split messages, forming a response. The information is either in a buffer or, when using Vector software, automatically sorted into signals. The data of split messages has to be combined again according to the marker. Even the Vector database cannot do this automatically. But there are only a few objects like the user text which require this treatment and these are usually not accessed very often.

Depending on the length of expected data, a response could be split into multiple messages. Those split messages use an extra marker.

**Access:** Base ID + 2

**Response with one message (number of queried registers 1-3):**

| Bytes 0+1 | Bytes 2-7 |
|---|---|
| Register | Data |
| 0...65534 | 1-3 registers |

**Response with multiple messages (number of queried registers >3):**

| Bytes 0+1 | Byte 2 | Bytes 3-7 |
|---|---|---|
| Register | Marker | Data |
| 0...65534 | 0xFF, 0xFE... | 5 bytes |

**Response as error message:**

| Bytes 0+1 | Byte 2 |
|---|---|
| 65535 | Error code |

The error codes used here are the same as with ModBus. See *„4.8.8. Communication errors“*.

### 8.3.5 Cyclic reading

The cyclic read feature is an extended function where the device can automatically send specific objects to specific IDs and in a specific interval. Cyclic read messages differ from those of normal read actions.

In order to activate and use cyclic read, the user has to

1. set the separate base ID for cyclic read on the device (HMI, CAN settings).
2. define which of the 5 available objects for cyclic are going to be used and activate them by setting the interval time to a value other than zero.

The interval times for the 5 objects can be separately and arbitrarily. In case they match or overlap, the device will send the corresponding messages subsequently and as fast as possible.

*The minimum interval is 20 ms. When using very low CAN bus speed, for example 10-50 kbps, CAN bus error may occur because of too much traffic.*

Once cyclic read is activated by setting the interval time of at least available object to other than 0 and as soon as a CAN connection is established, the device will start to automatically and permanently send messages to the defined IDs. While the CAN connection is open, the cyclic read feature can be turned off or on at will using the CAN settings on the HMI or the corresponding commands.

There are 5 IDs to reserve for cyclic read. Starting at the adjustable "Base ID Cyclic Read" (see HMI of the device) the data in the messages is defined as follows:

**Access:** Base ID Cyclic Read (Status)

| Bytes 0-3 |
|---|
| Device status (32 Bit) |

Bit layout of the device status value:

| Bit | Name | Bedeutung |
|---|---|---|
| 31 | Alarm OCP | 1 = alarm active |
| 30 | Alarm OCD | 1 = alarm active |
| 29 | Alarm MSS | 1 = alarm active |
| 28 | Alarme | 1 = at least 1 alarm active |
| 27 | UIP/UIR | 1 = UIR (register 409) |
| 26 | Volt. reg. speed | 1 = fast (register 422) |
| 25 | Input / output | 1 = on (req., register 405) |
| 24 | Remote control | 1 = on |
| 23 | Alarm OT | 1 = alarm active |
| 22 | Alarm OPP | 1 = alarm active |
| 21 | Alarm OPD | 1 = alarm active |
| 20 | Interface in access | register 505, bits 4-0 |
| 19 | | |
| 18 | | |
| 17 | | |
| 16 | | |

| Bit | Name | Bedeutung |
|---|---|---|
| 15 | Alarm UCD | 1 = alarm active |
| 14 | REM-SB | 1 = on (register 505, bit 30) |
| 13 | Alarm PF | 1 = alarm active |
| 12 | | |
| 11 | | |
| 10 | Alarm OVP | 1 = alarm active |
| 9 | Alarm OVD | 1 = alarm active |
| 8 | Warning OTpre | 1 = warning active |
| 7 | - | |
| 6 | Reg. mode | register 505, bits 10-9 |
| 5 | | |
| 4 | Input / output | 1 = on (register 505, bit 7) |
| 3 | MS-Typ | 1 = master, 0 = slave |
| 2 | Function gen. | 1 = FG active |
| 1 | Remote sensing | 1 = external, 0 = internal |
| 0 | Alarm UVD | 1 = alarm active |

**Access:** Base ID Cyclic Read + 1 (Actual values)

| Bytes 0-1 | Bytes 2-3 | Bytes 4-5 |
|---|---|---|
| **Register 507** | **Register 508** | **Register 509** |
| Actual voltage | Actual current | Actual power |

**Access:** Base ID Cyclic Read + 2 (Set values)

| Bytes 0-1 | Bytes 2-3 | Bytes 4-5 | Bytes 6-7 |
|---|---|---|---|
| **Register 500** | **Register 501** | **Register 502** | **Register 503** |
| Set value of voltage | Set value of current | Set value of power | Set value of resistance |

**Access:** Base ID Cyclic Read + 3 (Adjustment limits 1)

| Bytes 0-1 | Bytes 2-3 | Bytes 4-5 | Bytes 6-7 |
|---|---|---|---|
| **Register 9000** | **Register 9001** | **Register 9002** | **Register 9003** |
| U-min | U-max | I-min | I-max |

**Access:** Base ID Cyclic Read + 4 (Adjustment limits 2)

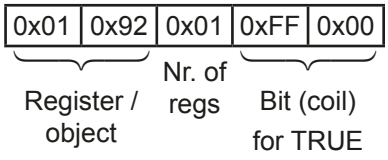
| Bytes 0-1 | Bytes 2-3 |
|---|---|
| **Register 9004** | **Register 9006** |
| P-max | R-max |

### 8.3.6 Message examples

#### 8.3.6.1 Switching to remote control

As described in *„4.8.9.5. Switch to remote control or back to manual control“*, it is required to switch the device to remote control before you can control it. In order to do this, you first need to find the proper command, i.e. register in the register list resp. the dedicated index in the EDS. In this case, it is register 402 (hex: 0x192). The register list defines that the value 0xFF00 has to be sent to switch to remote or value 0x0000 to leave remote control.

Assuming the device would have been set to base ID **0x100**, the data to be sent according to *8.3.1* would be:

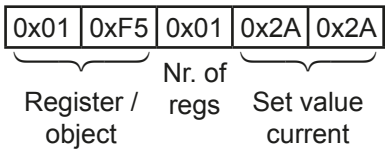



The device should then switch to remote control, if not inhibited somehow. The status of remote control can be read from the display or by reading another object.

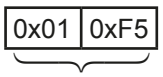
#### 8.3.6.2 Write and read back a set value

After remote control has been accepted by the device, you are allowed to send set values. Those values usually represent a per cent value. From the definition in the register list, the hexadecimal value 0xCCCC translates to 100% and 0x0000 to 0%. It means, there are 52428 possible values between 0% and 100%. It has to be pointed out here, that this is not the resolution a device value like voltage or current can have at the DC input/output. The effective resolution of output/input values is 26214 steps. An example for set value translation is in *„4.8.9.1. Writing a set value“.*

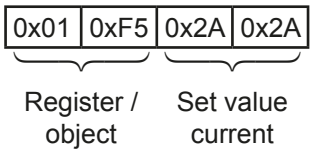
Power supply model PSI 9080-170 3U has a nominal current of 170 A. If you wanted to set it to 35 A, the set value according to the formula in *4.4* calculates as: 35 A * 52428 / 170 A = 10794 = 0x2A2A. The current is set with register 501. Assuming the device would have been set to base ID **0x88**, the data to be sent to ID 0x88 according to *8.3.1* would be:



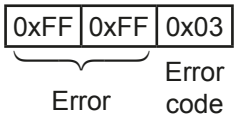

As soon as the device accepts the value, it is set and could be read from the display or by reading it back using the same object. With the same base ID, the query message would be




and would have to be sent to the query ID of the device, here **0x89**. Short after this, the device should respond the requested value on the read ID 0x8A:




In case the values has not been accepted when sending it, for example because the adjustment limit for current (I-max) has been set to 30 A, the device may have responded with an error message (see *8.3.4*) instead of the expected one:




The ModBus error code 0x3 tells "wrong data". In this case, the set value was too high.

# 9. EtherCAT

## 9.1 Preamble

Those device series supporting the Anybus interface modules (see *„2.2. Anybus module support“*) will also support the new EtherCAT module IF-AB-ECT by installing a firmware update. The update will be available from August 2016 as download or upon request.

Per definition, the EtherCAT data communication is based on CANopen protocol, here called CANopen over Ethernet. All documentation for EtherCAT and CANopen is provided by the Beckhoff company resp. the CiA organisation.

Below we will refer anything related to software to Beckhoff's TwinCAT.

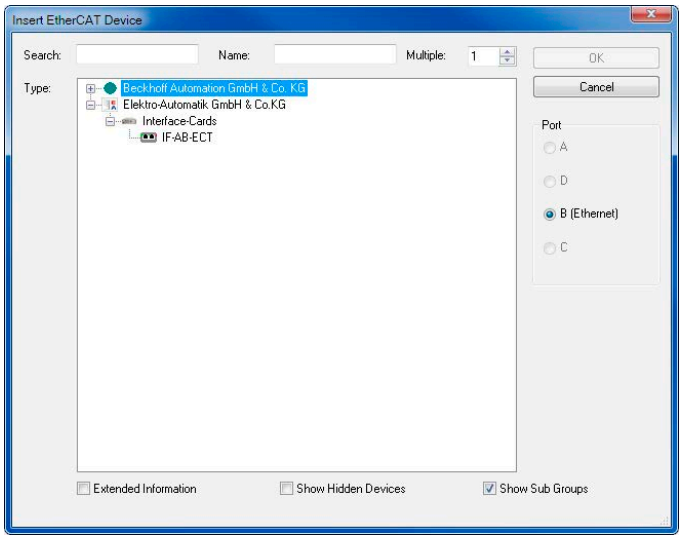
## 9.2 Integrating your device in TwinCAT

Newer devices are shipped with an USB stick that holds an ESI file, an EtherCAT device description in XML format. Alternatively, that file can be obtained upon request when updating an older device for EtherCAT support.

The file is simply put into a dedicated folder in the TwinCAT installation. Default path:

c:\TwinCAT\<twincat_version>\Config\Io\EtherCAT\

After installing that file and restarting the TwinCAT IDE, our EtherCAT slaves can be integrated into the setup with the "**Insert EtherCAT Device**" dialog and by selecting the device name "IF-AB-ECT":




Further slaves can be added with the same method.

## 9.3 Data objects

The devices internally use ModBus protocol and for CANopen over Ethernet communication in both directions the messages are translated. This is why the reference for all cyclic data (PDOs) and acyclic data (SDOs) are those ModBus register lists. They are included with the device on USB stick (or are available as download) as part of the programming documentation. The acyclic objects are downloaded from the device when accessing an online EtherCAT slave in tab "CoE" in TwinCAT. Offline objects in form of an EDS file are not available.

Together with the PDOs defined in the ESI file the complete list of indexes then becomes accessible and enable the user to completely control the device.

There is a connection between the CoE indexes and the ModBus register numbers in the lists. You can translate both back and forth.

- **Translating ModBus register ► CANopen index**

**ModBus register number in decimal + 8193 ► convert to hexadecimal = index**

Example: you want to set the device into remote control mode and want to find the corresponding CoE index. In the register list you have register number 402 for this task. Calculation: 402 + 8193 = 8595 ► converted to hexadecimal it is 0x2193, hence index 2193.

- **Translating CANopen index ► ModBus register**

**CANopen index in hexadecimal - 0x2001 ► convert to decimal = register**

Example: you need know the meaning of the bits in the PDO "Status". Find the corresponding CoE index in the index list. Here it is 21FA. Calculation: 0x21FA - 0x2001 = 0x1F9 ► converted to decimal it is 505. In the register list you will find register number 505 and the layout of the 32 bit value.

### 9.3.1 PDOs

The device description file defines for our EtherCAT slaves the same set of PDOs:

| Name | EtherCAT data type | Length in bytes | ModBus register | Short description |
|---|---|---|---|---|
| Status | UDINT | 4 | 505 | Device status |
| Voltage Monitor | UINT | 2 | 507 | Actual voltage on DC input/output (in per cent) |
| Current Monitor | UINT | 2 | 508 | Actual current on DC input/output (in per cent) |
| Voltage select | UINT | 2 | 500 | Set value of voltage (in per cent) |
| Current select | UINT | 2 | 501 | Set value of current (in per cent) |
| Power select | UINT | 2 | 502 | Set value of power (in per cent) |
| Resistance select | UINT | 2 | 503 | Set value of resistance (in per cent) |

### 9.3.2 SDOs

The acyclic data objects for use in the EtherCAT system are defined in your device and can be downloaded from it. It requires the device to be online with the EtherCAT system. There is no separate documentation for the downloadable data objects. Like with CANopen (see *„7. CANopen“*), the register lists which are part of the programming documentation are the reference for the SDOs to explain data content and function.

### 9.3.3 Use of the data objects

Please refer to *„7.2. User objects (indexes)“*.

# A. Appendix

## A1. Device classes

One or several device series can be assigned to a device class. The device class is a value that can be read from the device (register 0 or SYSTEM:DEVICE:CLASS?) and be used to easily distinguish a power supply from an electronic load when scanning a network for units of our make.

| Class | Assigned to series |
|---|---|
| **1** | PSI 9000 (old, until 2012) |
| **2** | EL 3000 / EL 9000 |
| **3** | PS 8000 T |
| **4** | PSI 800 R |
| **5** | BCI 800 R |
| **10** | PSI 8000 T / DT / 2U / 3U |
| **13** | PS 8000 2U / 3U |
| **15** | PS 8000 DT |
| **16** | PS 2000 B Single |
| **20** | ELR 9000 |
| **21** | PSI 9000 2U/3U (models from 2014) |
| **23** | PS 5000 |
| **24** | PS 2000 B Triple |
| **28** | PS 9000 2U/3U (models from 2014) |
| **29** | PSI 5000 |
| **30** | PS 9000 1U |
| **32** | ELR 9000 TFT |
| **33** | PSI 9000 2U/3U TFT |
| **34** | ELR 9000 TFT 3W |
| **35** | PSI 9000 2U/3U TFT 3W |
| **38** | PS 9000 2U/3U 3W |
| **39** | EL 9000 B |
| **41** | ELR 5000 |
| **42** | PSI 9000 DT |
| **43** | PSE 9000 3U |
| **44** | EL 9000 DT |
| **45** | PSI 9000 Slave |
| **46** | EL 9000 Slave |

Legend:

TFT = Model/series with TFT touch panel (older series had LCD touch panel)

3W = Model/series with option 3W installed

1U / 2U / 3U = Enclosure type is 19" with 1U or 2U or 3U height

T = Enclosure type: Tower

DT = Enclosure type: Desktop

R = Enclosure type: wall mount


**EA-Elektro-Automatik GmbH & Co. KG**
Entwicklung - Produktion - Vertrieb

Helmholtzstraße 31-33
**41747 Viersen**

Telefon: 02162 / 37 85-0
Telefax: 02162 / 16 230
ea1974@elektroautomatik.de
www.elektroautomatik.de