



Programming Guide

PS 2000 B Series

2020 TFT models

Elektro-Automatik



Attention! This document is only valid for the 2020 facelifted models of PS 2000 B with color TFT screen

Doc ID: PG2EN
Revision: 8
Date: 06-23-2020

TABLE OF CONTENTS

1. PREAMBLE	3
1.1 Introduction.....	3
1.2 Driver installation.....	3
1.2.1 Windows.....	3
1.2.2 Linux, MacOS and others.....	3
1.3 Terms.....	4
2. COMMUNICATION WITH THE DEVICE IN GENERAL	4
2.1 Structure of the communication.....	4
2.2 Serial communication parameters.....	4
2.3 Translating values	4
2.3.1 Actual values.....	4
2.3.2 Set values	4
2.4 General.....	5
2.5 Effective resolution when programming	5
3. MESSAGE FORMATS	6
3.1 Custom binary format.....	6
3.1.1 Telegram structure	6
3.1.2 The start delimiter in detail	7
3.1.3 The mask byte on object 54	7
3.1.4 Message examples	7
3.1.5 Possible problems when setting device conditions.....	8
3.1.6 Error messages.....	8
3.1.7 Trouble-shooting	9
3.1.8 Object list	9
3.2 ModBus RTU.....	10
3.2.1 Preamble.....	10
3.2.2 General information about ModBus RTU	10
3.2.3 About the register list	10
3.2.4 Message types	11
3.2.5 Functions.....	11
3.2.6 Control messages (write)	11
3.2.7 Query message.....	12
3.2.8 Response message (read).....	12
3.2.9 The ModBus checksum.....	13
3.2.10 Communication errors.....	13
3.2.11 Examples of ModBus RTU messages	14
3.3 SCPI.....	16
3.3.1 Format of set values and actual values.....	16
3.3.2 Syntax	16
3.3.3 Concatenated commands	16
3.3.4 Upper and lower case	16
3.3.5 Long form and short form.....	16
3.3.6 Termination character.....	17
3.3.7 Communication errors.....	17
3.3.8 Standard IEEE commands.....	17
3.3.9 Status registers	18
3.3.10 Output addressing.....	19
3.3.11 Set value commands.....	20
3.3.12 Measuring commands.....	20
3.3.13 Status commands	20
3.3.14 Commands for protective features	21
3.3.15 Commands for adjustment limits.....	21
3.3.16 Commands for the tracking mode	22
3.3.17 Further commands.....	22

1. Preamble

1.1 Introduction

This guide's purpose is to explain the communication protocols for the 2020 redesigned generation of series PS 2000 B models. These devices support three different message formats or command syntax:

- Custom binary EA message format (same as in the previous PS 2000 B model generation)
- SCPI
- ModBus RTU

The device can distinguish the three different protocols automatically upon the first byte of a message. This is possible because of some rules:

- The slave address in byte 0 of ModBus message must always be either 0x00 or 0x01, though it's not used to address the device -> when the first byte is either 0 or 1 the message is considered and processed as ModBus RTU message
- When using the binary protocol, the 2nd byte of the message can only be 0x00 or 0x01 in order to address the output 1 or output 2 (for a single model it would thus always be 0x00) -> if the 2nd byte is either 0x00 or 0x01, the message is considered and processed as custom binary message
- When the first byte contains a value >1 and the 2nd byte as well, the message is considered and processed as SCPI command string
- Any other case is considered as an invalid message

Communication is solely done with the front USB as it the only digital interface. Once installed, the USB driver creates a virtual COM port (VCP) for any new unit of this device type. Using the COM port will reduce the programming effort for the communication port itself to a minimum, because the virtual COM port doesn't require configuration.


1.2 Driver installation

1.2.1 Windows

The USB driver comes on USB stick with the device. In case the stick isn't available, you can find the driver on our website as well. It's compatible to all Windows versions since 7 (except for Embedded).

We recommend to install the driver before connecting the device to the PC the first time, in order to prevent Windows from installing another driver which wouldn't configure the device as PS 2000 in the system. This can become a problem when starting to use LabView and our supplied VI set which requires the device to be correctly named according to our driver.

After successful installation you can safely connect the device to the PC. In order to verify correct device installation it's helpful to open the Windows Device Manager (on Windows 10 you find in the context menu when right-clicking the start menu button). Go to section "Ports" to find a

 **PS 2000 Single (COM8)** or PS 2000 Triple (COM8), depending on the model.

Note: The COM port number 8 above is only an example. Windows assigns a new COM port for every new device of this type that is installed in the system. The port is remembered and used again if the device is connected the next time.

Note: In case there are multiple units connected to the PC, they will all be listed as separate devices in the device manager.

1.2.2 Linux, MacOS and others

We can't offer a proprietary driver for operating systems other than Windows. There are generic drivers on at least Linux and MacOS available. In case the automatic detection fails, the USB hardware is of type CDC (communication device class).

However, the generic driver has a downside: our LabView VI package can't be used or at least the scan VI, which is used to find and list our devices, because it relies on data coming from the Window registry.

1.3 Terms

Telegram or Message or Message format = Chain of hexadecimal bytes, with varying length. It's either sent to a device or received from it. A part of the telegram (data field) represents hexadecimal values or ASCII strings.

SCPI = **S**tandard **C**ommands for **P**rogrammable **I**nstruments, an internationally standardized text/string based command language

ModBus / ModBus RTU = binary, prolific and specified data transmission format which is based on sub formats like RTU

2. Communication with the device in general

2.1 Structure of the communication

The communication with the unit is based on these telegram types:

a) **Send message**: an object or command is sent which shall, for instance, set the output voltage. As long as this action is permitted by the current state of the device, the command is accepted and executed. Depending on the message format used in the last transmission, the device would do following:

- when using the custom binary message format, it would send an answer in form of an error message, but with error code 0. Otherwise, an error code other than 0 is returned.
- when using ModBus it would return an acknowledge message (different formats, depending on the function code)
- when using SCPI it won't return anything

b) **Query message**: a query is sent to the device and an answer is expected, which would contain the requested data. In case of a communication error the device would either return an error message (custom binary / ModBus) or nothing (SCPI).

2.2 Serial communication parameters

Data transfer is done via a virtual COM port (VCP), which is generated by the USB driver. Since the COM port is virtual, the driver ignores the actual serial setting, so any setting is OK, also the default one. It means, depending on the IDE in use, it may not even be required to configure the serial settings.

2.3 Translating values

Two of the above listed message formats, the custom binary and ModBus, require to translate set values and actual values as percentage values for transmission. A value of 0x6400 (custom binary) or 0xCCCC (ModBus) corresponds to 100.00%. The different hex value for 100% with ModBus comes from the requirement of compatibility to our software and other series which also use ModBus.

2.3.1 Actual values

An actual value is queried and read from the device and will be returned as hexadecimal 16 bit value, representing a per cent value.

$$\text{Real actual value} = \frac{\text{Rated value} * \text{Per cent act. value}}{\text{Translation factor}}$$

Translation factors

- Custom binary format: 26500 or 0x6400
ModBus: 52428 or 0xCCCC

Example: The rated voltage of the device is 42 V, the percentage actual value came as 0x2454 = 9300. When translating it from ModBus format it results in an actual value of $42 * 9300 / 25600 = 7.45$ V.

2.3.2 Set values

Set values have to be translated into 16 bit per cent values before transmission. Reading set values back from the device requires to translate them vice versa.

$$\text{Percentage set value} = \frac{\text{Translation factor} * \text{Real set value}}{\text{Rated value}}$$

Translation factors

- Custom binary format: 26500 or 0x6400
ModBus: 52428 or 0xCCCC

Example: the set value of current shall be 13.5 A, the rated current of the device is 20 A. With the formula it results in a per cent value of $25600 * 13.5 / 20 = 17280 = 0x4380$, when translating it for the custom binary message format. After sending 0x4380 to the 20 A model, it should set a current of 13.5 A for the addressed output.

2.4 General

Basic rules:

- Monitoring, i.e. only querying actual values and status, is always possible. The device doesn't require to be in remote mode in this case
- Setting of status and set values (controlling) requires the activation of remote control mode
- With the triple models, outputs 1 and 2 are remotely controllable and have to be addressed separately, except for when tracking mode is activated where it's only possible to address output 1 which is followed by output 2

In order to start **controlling** a device you need to

1. always activate the remote mode first (object 54)

2. and then you can send set values or status.

Remote control should be left if not used any further. As long as it's active, the device or the addressed output can't be operated manually. The mode is indicated on the front display.

2.5 Effective resolution when programming

All values related to voltage and current, as they can be transferred to the device and which are transferred via the power stages to the addressed DC output have the same defined programmable resolution and an effective resolution. The same applies to the actual which are sampled from the DC output using simple measuring circuits. "Simple" means that the device can't be considered as and compared to a multimeter, which measures faster and more precise.

Overview:

Message format	Programmable resolution of set values	Effective resolution
Custom binary format	0 - 0x6400 = 25600	25600 steps
ModBus	0 - 0xCCCC = 52428	25600 steps
SCPI	Theoretically infinite, but the number of effective decimal places is identical to the format of the corresponding value on the display	Model depending, e. g. 5 A = 500 steps, because a 5 A rated model would display it as 5.00

Message format	Achievable resolution of actual values
Custom binary format	≤1024
ModBus	≤1024
SCPI	Model depending, e. g. 5 A = 500 steps (see above)

The **effective resolution** depends on the analog-digital converters used in the hardware. They determine the achievable step width of set values on the DC output. It calculates as step width of voltage or current = rated value ÷ effective resolution. For instance, with model PS 2384-05 B the approximate step width of a voltage set value set with ModBus could then be $84 \text{ V} / 25600 = \approx 3 \text{ mV}$. For the current it would be $50 \text{ A} / 25600 = \approx 0.2 \text{ mA}$. Actual values have a significantly lower resolution.

However, tolerances add to the result when setting a value, shifting the actual result. The PS 2384-05 B from the example above has a voltage tolerance of max. 0.2%, as stated in the user manual. This is up to 168 mV. When setting, for example, 24 V the true output voltage is allowed to be within 23.83 V and 24.17 V. The actual value, as readable from the device, already includes this tolerance (or error). If you would measure the actual output voltage with an external multimeter and it would probably read 24.1 V and you would want to it have closer to the desired 24 V, the software could adjust the set value in approx. 3 mV steps to further narrow the actual value to the set value.

3. Message formats

3.1 Custom binary format

3.1.1 Telegram structure

The telegram consists of a variable number of bytes of this structure:

Byte 0	Byte 1	Byte 2	Variable number of bytes	2 bytes
SD	DN	OBJ	DATA	CS

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

The start delimiter marks the beginning of the telegram and determines if the message is a query or not. Meaning of the bits:

Bits 3-0: Number of bytes -1 in the **DATA** field of the telegram (bytes 3-18), which can be up to 16 Bytes when writing or reading. The length is primarily used for query messages to define the number of bytes to return. The maximum of 16 would then result in the nibble value 0xF in bits 3-0. When only writing, it tells the device the length of **DATA** so it can determine and check the expected length of the entire message. The maximum data length of a specific object is given in column 6 of the communication object list.

Bit 4: Direction

0= Telegram from device to control unit

1= Telegram from control unit to device

Bit 5

1= Must always be 1 for sending/querying to the device, though in answers from the device it will be 0

Bits 6+7: Transmission type

00 = Reserved

01 = Query data (PC->device)

10 = Answer to a query (device->PC)

11 = Send data (PC->device)

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

Here we need to distinguish whether a Single or Triple model of PS 2000 B series is going to be controlled. This value is used to address a specific DC output. While Single models only have one output (Output 1), there are two outputs to address with Triple models, Output 1 (left-hand display) and Output 2 (right-hand display). Output 3 can't be remotely controlled. The DN is returned 1:1 in an answer message, in order to know from which output it came.

Rules:

Output 1: DN must be 0 (Single or Triple model)

Output 2: DN must be 1 (only with Triple models)

In regard to the automatic distinction between the three supported message formats, it has to be pointed out that the byte DN must always only be 0 or 1 in order to correctly detect the message as "custom binary format", apart from the correctly addressing a specific output.

Byte 2: **OBJ**

The communication object number is given here. Refer to the communication object list for PS 2000 B series to find the available objects, their object number and function. See section 3.1.8.

Bytes 3 - 18: **DATA**

The **DATA** field can be 0-16 bytes long, hence the length of the telegram varies. If a query is sent to the device, the data field isn't used and the checksum of the telegram directly follows after byte 2. Data are only transmitted when sending something to the device or when receiving an answer from it.

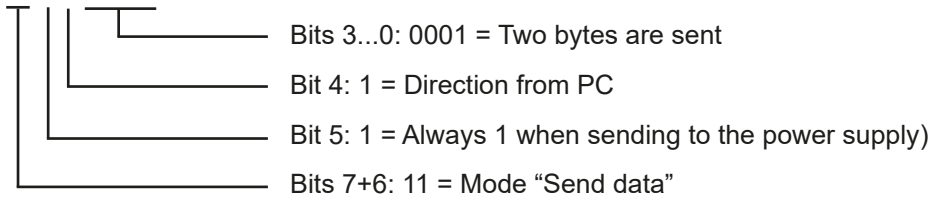
Last two bytes: **CS** (check sum)

The check sum is always located at the end of the telegram. It's built by the simple addition of all preceding bytes of the telegram. It's two bytes long. The high byte is placed before the low byte.

3.1.2 The start delimiter in detail

According to the telegram format (see above), the first byte of a telegram is the start delimiter, which depends on the type and direction of the telegram. For example, the SD can be 0xF1 and looks like this in single bits:

11 11 00 01



This SD determines that data are sent to the device. The content of the data and the object define what is sent and what the device will do in reaction.

Alternatively to the bitwise assembly, this can be simplified by adding hex values:

SD = Message type + Cast type + Direction + Length

whereas the message type is either

0xC0 Send data or

0x40 Query data

0x80 Answer from device

and the cast type is

0x20 Broadcast

and the direction is either

0x10 from PC to the device or

0x00 from device to the PC

and the data length - 1 can be

0x01...0x0F up to 16 bytes of data

By the above example, the SD of 0xF1 is built from 0xC0 + 0x20 + 0x10 + 0x01.

The SD in answers from the device will be different and can be ignored or at least used to read the data length from it.

3.1.3 The mask byte on object 54

Object 54 requires to use a mask byte, which is sent together with the control byte. The possible mask values are given in column 6 of the object list (see separate PDF file). The bits of the control byte have various functions, so it has to be determined which bit is going to be changed. This is defined by the mask with a 1 for the corresponding bit of the control byte.

Example: bit 0 of the control byte shall be changed to 0 or 1. This will result in the control byte being either 0x00 or 0x01 and the mask being 0x01. Also see the object list at object 54.



Though the mask bytes allows to change multiple bits at once, it's strongly recommended to only change one bit per message in order to avoid conflicts.

3.1.4 Message examples

Note: the below listed hex values are in simplified form, without the usually leading 0x.

Example 1: The actual values shall be queried from the device. According to the object list, this can be done with object 71. The telegram to query the actual values has to look like this, according to the above explained telegram structure and construction of the start delimiter:

75 00 47 00 BC for a Single model or output 1 of a triple model or

75 01 47 00 BD for Output 2 of a triple model

The answer from the device could be like this (single model or output 1 of a triple model):

85 00 47 01 01 64 00 1E 00 01 50 ⁽¹⁾

This will translate to 42 V (green value = actual voltage) and 1.8 A (blue value = actual current) for a PS 2042-06B with 42 V rated voltage and 6 A rated current. The status is returned as 0x0101 (pink value) and translates to "remote control on", "DC output on" and "regulation mode CV".

1) For the decoding of the bytes see the object list, object 71

Example 2: Activate remote control. This can only be done when the addressed output and its dedicated display isn't menu mode or when Output 2 of a Triple model is going to be addressed, tracking mode can't be active.

F1 00 36 10 10 01 47 for a Single model or output 1 of a triple model or

F1 01 36 10 10 01 48 for Output 2 of a triple model

A successful execution of the command is responded by either

80 00 FF 00 01 7F for a Single model or output 1 of a triple model or

80 01 FF 00 01 80 for Output 2 of a triple model, containing error code 0x00, which means "OK".

In case the command can not be executed, the error code would change:

80 00 FF 05 01 80 returned error 0x05 ("Wrong device node"), which means it was tried to address Output 2 on Single model.



Hex values must be transferred as binary bytes, not as ASCII string!

3.1.5 Possible problems when setting device conditions

Object 54 is used to either activate/deactivate remote control operation or switch the addressed output of a device on or off. The object can be used to activate both states at once, but it's strongly not recommended to do so, because setting the output requires remote control already being active and else would generate an error message. The best way is to activate remote control first, via the corresponding bit the control byte, and then control the DC output by sending object 54 a second time with a different control byte and mask. When deactivating remote control it goes vice versa.

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

3.1.6 Error messages

When sending values or conditions, the device will return an acknowledging message in form of an error message using object number **0xFF** and containing error code **0**. This indicates that the last command was received and executed correctly. Otherwise, if the device can't execute the last command for some reason or the telegram was bad, an error message containing an error code other than 0 is returned. Error code list:

Error code		
Hex.	Dec.	Description
00	0	No error
03	3	Check sum incorrect
04	4	Start delimiter incorrect
05	5	Wrong address for output
07	7	Object not defined
08	8	Object length incorrect
09	9	Read/Write permissions violated, no access
0F	15	Device is in "Lock" state
30	48	Upper limit of object exceeded
31	49	Lower limit of object exceeded

Legend

	Communication error
	User error

Example: if you try to set the output voltage with object 50 while the device isn't in remote control, it would return the error message **80 00 FF 09 01 88**. The error code **0x09** indicates, that the device is in a state where it's not able to accept object 50.

Explanation of some error codes:

Code 0x7: the object number used in the telegram is unknown to the device. Note, that the object numbers are not subsequent.

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

Code 0x9: additionally to the above example, this error code can also mean that there was an attempt to write to an object which is, according to the object list, read only (ro).

Code 0xF: there was an attempt to switch the addressed output to remote control while the related display was in menu mode.

3.1.7 Trouble-shooting

Possible problem: Multiple queries have been sent, but not all of them have been answered

Most likely cause: The queries have been sent too quickly after each other. Depending on the not exactly defined execution time of commands, you need to include a certain latency between two transmissions. The minimum time between two commands is recommended as at least **50 ms** for a device of PS 2000 B series.

Possible problem: Set values and status are not set

Possible causes

- The device or addressed output isn't in remote control mode. Should result in error code 0x09.
- A value sent to the (addressed) output is wrong (too high, too low). Then an error message would be returned. Or the value is accepted, but can't be transferred to the output because of the current device condition, like when sending a voltage set value while the device is in current limitation. In this case it wouldn't return an error.

3.1.8 Object list

See separate PDF file named object_list_ps2000b_de_en.pdf.

3.2 ModBus RTU

3.2.1 Preamble

Important! Read this for further understanding:



This series has been update in 2020 to support the ModBus RTU message format for the first time. Despite ModBus having a specification, our devices didn't fully comply to that specification in the past. In the beginning of 2020 all series supporting ModBus thus received an update due to which the compliance to the specification can be switched between limited and full, whereas "limited" represents the former way of our implementation and in order to keep a certain compatibility to earlier firmware releases, the limited compliance mode is the default setting after production or after a factory reset. It means that when starting using ModBus with a PS 2000 B device it's required to activate the full compliance at least once. It will be stored.

This system also has been submitted to the PS 2000 B, to have it compatible to other series. The compliance can be switched (register 10013) between "Full" and "Limited" (default). Differences:

- "Full" only supports the slave ID / address 1 and returns READ COILS functions correctly
- "Limited" only supports slave ID / address 0, so activating mode "Full" requires to send the message to address 0 and it would always return 16 coils to a READ COILS query

From here on it's assumed that the device you are going to program is set to mode "Full".

3.2.2 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 so-called slave ID, **must always be 1** because our devices don't need an adjustable address and so it's defined to be 1. Reason: The first byte of a telegram is also used to detect the message format of the telegram. Also see „3. Message formats“ about this topic.

Format and length of a telegram are defined as detailed below.

3.2.3 About the register list

Along with this programming guide, there is a so-called register list included as PDF file. This list gives an overview about the remote programming features that are available for a PS 2000 B device when accessing it with ModBus.

The list explains in compact format how the data in a binary ModBus message has to be interpreted or how a register 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 don't need this list. Later in this document, the SCPI commands are referenced in a separate chapter. When using the custom binary format, however, there is a separate list, the PS 2000 B object list.

3.2.3.1 Columns "ModBus address"

This number, given in decimal and hexadecimal form, is the so-called ModBus register address or register number. It's used in hexadecimal form in ModBus messages.

3.2.3.2 Columns "Function"

The heads of the 5 columns next to the ModBus address column contain the names and codes of the supported ModBus functions. An "x" in these columns mark the assignment of a register to any of the functions. For example, the so-called coil registers are usually writable and readable, so they're assigned to functions "Read Coils (0x01)" and "Write Single Coil (0x05)".

3.2.3.3 Column "Data type"

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

3.2.3.4 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 or wouldn't return a reasonable value when read from

RW = Register can be read or written



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

3.2.3.5 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 value is always the half of the value in column "Data length in bytes".

3.2.3.6 Column "Data"

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

3.2.4 Message types

Basically, the message system distinguishes between **query messages**, **control messages** and **response messages**. Query messages will cause the device to send a response message, while control messages only cause it to reply with a 1:1 echo, in order to confirm reception.

3.2.5 Functions

The second byte of a message contains a ModBus **function** code (**FC**, marked in blue below), which determines whether the message is a READ or WRITE message. It also determines, whether one or multiple registers are accessed. The protocol as described below supports following ModBus functions :

Function		Function name		Description	Example of use
Hex	Dec	Long	Short		
0x01	1	READ COILS	RC	Only allows to read 1 coil, because the coils are not organized incrementally.	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 the coil (TRUE/FALSE) of a boolean register	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. Can't 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

3.2.6 Control messages (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 „3.2.11. Examples of ModBus RTU messages“.



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

WRITE Single Register

Byte 0	Byte 1	Bytes 2+3	Bytes 4+5	Last 2 Bytes
ID	FC	Start reg.	Data word	CRC
0x01	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
ID	FC	Start reg.	Number	Count	Data bytes	CRC
0x01	0x10	0...65535	0...123	Number*2	Data	Checksum ModBus-CRC16 ⁽¹⁾

WRITE Single Coil

Byte 0	Byte 1	Bytes 2+3	Bytes 4+5	Bytes 6+7
ID	FC	Register	Data word	CRC
0x01	0x05	0...65535	0x0000 (FALSE) or 0xFF00 (TRUE)	Checksum ModBus-CRC16 ⁽¹⁾

3.2.7 Query message

When querying something from the device, the response is expected to be immediate 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 or 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. With READ COILS, the response will be one byte (=1 coil) or two bytes (=16 coils, former response in earlier firmwares).

For message examples see „3.2.11. Examples of ModBus RTU messages“.

READ HOLDING REGISTERS

Byte 0	Byte 1	Bytes 2+3	Bytes 4+5	Last 2 Bytes
ID	FC	Start reg.	Number	CRC
0x01	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	Last 2 Bytes
ID	FC	Start reg.	Number	CRC
0x01	0x01	0...65535	Must always be 1	Checksum ModBus-CRC16 ⁽¹⁾

3.2.8 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	Last 2 Bytes
ID	FC	Start reg.	Data	CRC
0x01	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	Last 2 Bytes
ID	FC	Start reg.	Data	CRC
0x01	0x05	0...65535	Written value echoed	Checksum ModBus-CRC16 ⁽¹⁾

⁽¹⁾ See „3.2.9. The ModBus checksum“

Expected response for WRITE MULTIPLE REGISTERS:

Byte 0	Byte 1	Bytes 2+3	Bytes 4+5	Last 2 Bytes
ID	FC	Start reg.	Data	CRC
0x01	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	Last 2 Bytes
ID	FC	Data length in bytes	Data	CRC
0x01	0x03	2...250	Queried registers content	Checksum ModBus-CRC16 ⁽¹⁾

Expected response for READ COILS:

Byte 0	Byte 1	Byte 2	Byte 3	Last 2 Bytes
Head	FC	Data length in bytes	Data	CRC
0x01	0x01	1	0x00 or 0x01	Checksum ModBus-CRC16 ⁽¹⁾

Unexpected response (communication error):

Byte 0	Byte 1	Byte 2	Last 2 Bytes
Head	FC		CRC
0x01	0x80 + Function code	Error code	Checksum ModBus-CRC16 ⁽¹⁾

3.2.9 The ModBus checksum

The checksum at the end of ModBus RTU messages is a 16 bit checksum, but it isn't calculated as the usual CRC16 checksum. Furthermore, **the byte order** of the checksum in the message is **reversed**. Information about ModBus CRC16 and source code for implementation and calculation are available on the Internet, for example here: http://www.modbus.org/docs/Modbus_over_serial_line_V1_02.pdf, section 2.5.1.2.

3.2.10 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 isn't in remote control. Then the message won't be accepted and the device will return an error message instead of a confirmation message. The message format itself 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 added to 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 the 2nd byte of the ModBus message isn't supported. See „3.2.5. Functions“ for supported codes. The error also occurs when trying to read or write a register with a function code for which the register isn't defined.

⁽¹⁾ See „3.2.9. The ModBus checksum“

Code	Error	Explanation
0x02	2	Invalid address The register address you were trying to access with read or write isn't 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 isn't 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 isn't 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 the "local" condition, so only read access is possible. "Local" means that remote control isn't allowed.

An example: You attempted to switch the device to remote control in order to control it from PC, but instead of an echo of your message it returns something like this: 01 85 07 03 52. 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 0x07 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.

3.2.11 Examples of ModBus RTU messages

3.2.11.1 Writing a set value



Set values are adjustable control value for the regulation of the physical values current and voltage. They can only be written to a device, if it has been switched to remote control before.

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:	ID	FC	Start	Data	CRC	Expected response:	ID	FC	Start	Data	CRC
	0x01	0x06	0x01F5	0x6666	0x338E		0x01	0x06	0x01F5	0x6666	0x338E

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 10 A nominal current it should show 5.00 A or for a model with 5 A current rating it should show 85.00 A.

3.2.11.2 Query all actual values at once

The device holds three readable actual values of voltage, current and power. These 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:	ID	FC	Start	Data	CRC	
	0x01	0x03	0x01FB	0x0003	0x75C6	

Possible response:	Head	FC	Len	Data	CRC
	0x01	0x03	0x06	0x2620 0x0C9B 0x091B	0x9350

3.2.11.3 Read the nominal voltage of a device

The nominal or rated voltage, like the other nominal values of current or power, is an important value to read from a device. They're all referenced for translating set values and actual values. It's 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:	ID	FC	Start	No.	CRC	Possible response:	ID	FC	Len	Data	CRC
	0x01	0x03	0x0079	0x0002	0x15D2		0x01	0x03	0x04	0x42A00000	0xEE69

Also see 3.2.8. The response contains a float value according to IEEE754 format, which translates to 80.0.

3.2.11.4 Read device status

All device report their device status in register 505.

Query message:	ID	FC	Start	No.	CRC	Possible response:	ID	FC	Len	Data	CRC
	0x01	0x03	0x01F9	0x0002	0x15C6		0x01	0x03	0x04	0x00000483	0xB952

Also see 3.2.8. The response contains the value 0x483 which states that the device is in remote control via the USB port, that the DC output is switched on and that CC (constant current) mode is active.

3.2.11.5 Switch to remote control or back to manual control

Before you can control a device from remote, it's 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 from all readable registers is always possible.



The device will never exit remote control automatically, unless it's 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 at least one circumstance and is usually indicated by an error message:

- Condition „**Lock**“ is active (check the display on the front of your device or read the device status), which can mean that the display belonging to the addressed output is currently in menu mode

► How to switch a device to remote control:

1. Create and send a message according to the description above, for example 01 05 01 92 FF 00 2C 2B for output 1.
2. Once the switchover to remote control has been successful, the device will usually indicate the new condition in the display, 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 01 85 17 02 9E. 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. Create and send a message according to the description above, for example 01 05 01 92 00 00 6D DB for output 1.

3.3 SCPI

SCPI is an international standard for a clear text based command language. Details about the standard itself can be found on the internet.

3.3.1 Format of set values and actual values

In the SCPI command language **real values** are used, with or without unit. It means, if you wanted to set a current of 17.5 A you would use the simple command **CURR_17.5** or, with unit, **CURR_17.5A**. Below you will find more detailed information about the available commands and their syntax. The space, as required to be put between the command and a parameter is below replaced by symbol “_”.

3.3.2 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: - The value must be sent after the command and separated by a space - 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: MAX for a set value like U or I = adjustment limit (e. g. U-max) MAX for a protection (OVP, OCP) = 110% of the rated value
<NR1>	Numeric values without decimal place	
<NR2>	Numeric values with decimal place (floating point), includes NR1	
<NRf>	<NR1> or <NR2> or <NR3>	
Unit	V (Volt), A (Ampere), W (Watt)	
<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	
	REMOte: device is in remote control	
<ERR>	Error with number and description	
<SRD>	String data, various formats	
;	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's required to wait for the response of the query before the next control message can be sent.	
->	Response from device	

3.3.3 Concatenated commands

It's possible to couple, i.e. concatenate up to 5 commands in one message. The commands must then be separated by a semicolon (;). Example:

VOLT 20;CURR 10;MEAS:ARR?

The command in the string are processed from left to right, 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.

3.3.4 Upper and lower case

SCPI uses upper case commands by default, though the device also accept lower case form.

3.3.5 Long form and short form

SCPI commands have a long form 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 the additional part of the long form).

3.3.6 Termination character

Some interfaces require to attach a termination character to the message, while others don't, such as USB. There the termination character is optional and used in order to maintain compatibility between several different interfaces in control softwares which use SCPI.

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

3.3.7 Communication errors

Errors in terms of SCPI are only communication errors. According to the standard, devices using SCPI don't return errors immediately. They have to be queried from the device. The query can occur directly with the error command (see 3.3.17) or by first reading the signal bit "err" from the Status Byte register (see „3.3.9. 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 or incomplete
-102,"Syntax error"	Command syntax wrong -> example: SYST:LOKc (partially correct)
-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
-220,"Parameter error"	Wrong parameter used
-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
-225,"Out of memory"	The expected answer could not be sent because it would exceed the internal buffer (can only occur with 5x *IDN? in one query message)

3.3.8 Standard IEEE commands

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.

3.3.8.1 *CLS

Clears the error queue and the status byte (STB).

3.3.8.2 *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)

3.3.8.3 *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)

3.3.8.4 *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 2: *err*, Error Queue --> one or several error in the error buffer. By reading the error buffer or sending *CLS it's

flushed and the bit *err* is reset

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

Bit 5: *esr*, the Event Status Register (ESR) is active

Bit 6: not used

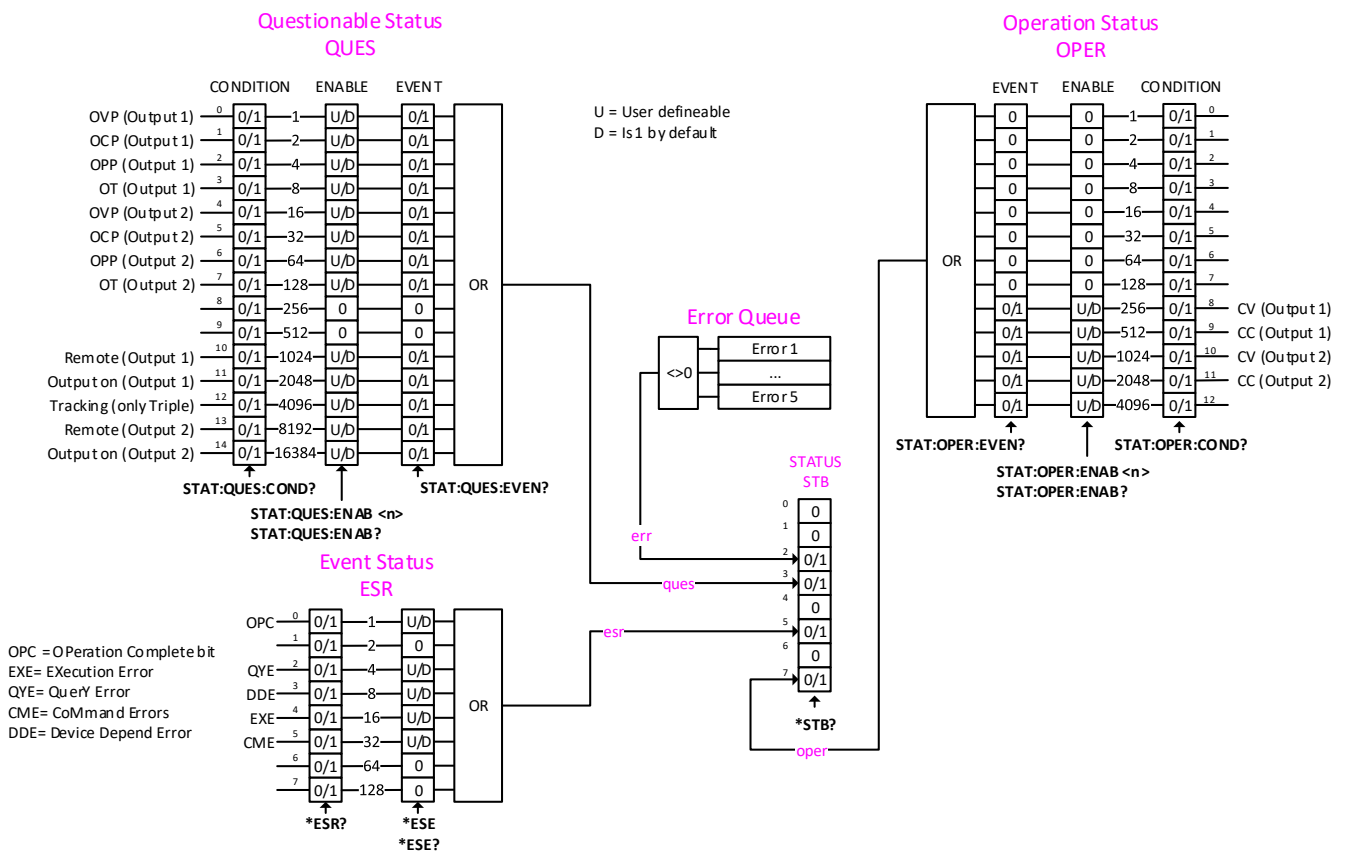
Bit 7: *oper*, Operation Status Register is active (one or several events have occurred)

3.3.9 Status registers

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 which 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 signalizes, that there was an event recorded in QUES or OPER register, you could read the corresponding event register of OPER and QUES, in order to find out which bits have changed in the COND register.

Register model:



Events recorded in the event registers STAT:QUES:EVENT and STAT:OPER:EVENT only record PTRs (positive transition), i. e. the changeover from 0 to 1.



Device alarms like OVP are signaled in the subregisters CONDITION and EVENT. The have to be cleared separately by using either SYST:ERR? or SYST:ERR:ALL?, which is considered as alarm acknowledgement and will clear the corresponding bit in CONDITION, but only if the alarm condition isn't present anymore.

Command	Description
STATus:QUESTionable? STATus:QUESTionable:CONDition?	Reads the inputs of the Questionable Status register and returns a value representing the bit signals on CONDITION.
STATus:QUESTionable:EVENT?	Reads the Event sub register of the Questionable Status register and returns a value representing the bit status.

Command	Description
STATus:QUEStionable:ENABle_<NR1> STATus:QUEStionable:ENABle?	Sets a filter/mask for CONDITION before it's passed on to the Event sub register of the Questionable Status register or reads the current mask. This filter can be used to suppress events of particular signals. By default, all bits are enabled for which a signal is given (see register scheme). The value NR1 must not be higher than the sum of all active signal bits.
STATus:OPERation? STATus:OPERation:CONDition?	Reads the inputs of the Operation Status register and returns a value representing the bit signals on CONDITION.
STATus:OPERation:EVENT?	Reads the Event subregister of the Operation Status register and returns a value representing the bit status.
STATus:OPERation:ENABle_<NR1> STATus:OPERation:ENABle?	Sets a filter/mask for CONDITION before it's passed on to the Event sub register of the Operation Status register or reads the current mask. This filter can be used to suppress events of particular signals. By default, all bits are enabled for which a signal is given (see register scheme). The value NR1 must not be higher than the sum of all active signal bits.

3.3.10 Output addressing

The so-called Single models only have one output, so addressing isn't required there. The Triple models, however, have to separately accessible and controllable outputs, so addressing becomes necessary. With SCPI this is done with a suffix added to the actual command for either set or query commands. Following rules apply:

- The suffix @1 is dedicated to Output 1 (featured with Single and Triple models)
- The suffix @2 is dedicated to Output 2 (only featured with Triple models)
- Using @1 with a Single model isn't required, but supported
- With Triple models it's possible to omit the suffix when addressing Output 1. Without any suffix the command would always directed to Output 1
- With Triple models it's possible to address both outputs at once, to achieve a synchronous setting of values or query of actual values

Format of output addressing for set commands using an example:

CURR 12, (@1,2) addresses outputs 1 and 2 of a Triple model and sets 12 A. This command wouldn't be executed with a Single model and cause an error, because a Single model has no Output 2 to address. The command and the space after the command are required. Alternative forms: **CURR 12, (@1-2)** or **CURR 12, (@1:2)**

Format of output addressing for query commands using an example:

MEAS:ARR? (@2) addresses Output 2 of a Triple model to query the actual values. The space after the actual command, which ends with the question mark, is mandatory. Extended to query both outputs the command would be **MEAS:ARR? (@1,2)** and would return the actual values of both outputs at once as comma separated grouped string, such as 19.80 V, 3.25 A, 64.4 W, 0.00 V, 0.00 A, 0.0 W in which the first group belongs to Output 1 etc. The values indicate that output 2 is either set to 0 V or switched off.

3.3.11 Set value commands



All values which have dedicated commands are always by adjustment limits, as definable in the setup menu or by additional commands in remote control.



Also in remote control it applies that setting a voltage or current set value will impact the other one so the maximum power won't be exceeded. If you wanted, for instance, to define the current at the desired level as given by the CURR command, you would have to set the voltage first. The resulting opposite set value can only be determined by reading it back with CURR? or VOLT?.

Command	Description
[SOURce:]VOLTage_<NRf>[Unit] [SOURce:]VOLTage?	Writes or reads the voltage set value. The unit is optional when writing. When querying the value it always comes with unit. Examples: VOLT_10 -> absolute short form, set 10 V SOURCE:VOLTAGE_5.35V -> absolute long form, sets 5.35 V
[SOURce:]CURRent_<NRf>[Unit] [SOURce:]CURRent?	Writes or reads the set value of current. The unit is optional when writing. When querying the value it always comes with unit. Examples: CURR_8 -> absolute short form, set 8 A SOURCE:CURRENT_18.7A -> absolute long form, sets 18.7 A

3.3.12 Measuring commands

Measuring commands return the last actual values which have been acquired by the device by either measurement (U, I) or calculation (P). These represent the last situation on the (addressed) DC output. Actual values are acquired asynchronously to the measuring commands. It means, the values are not measured in the moment of query.

Actual values are not necessarily identical to the corresponding set values. The device acquires the actual values periodically.

Command	Description
MEASure:[SCALar:]VOLTage[:DC]?	Reads the last acquired actual value of DC output voltage, which is immediately returned. Example: 3.45V
MEASure:[SCALar:]CURRent[:DC]?	Reads the last acquired actual value of DC output current, which is immediately returned. Example: 10.12A
MEASure:[SCALar:]POWer[:DC]?	Reads the last acquired actual value of DC output power, which is immediately returned. Example: 34.9W
MEASure:[SCALar:]ARRay?	Reads all three actual values of the DC output and returns them in a combined form in the order U, I, P. Example: 3.45V, 10.12A, 34.9W

3.3.13 Status commands

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

Command	Description
SYSTem:LOCK_<B0>	Activates remote control with ON , if permissible for the current situation of the device, or leaves it with OFF . The state of remote control can be queried anytime, before or after this command, with SYST:LOCK:OWN?
SYSTem:LOCK:OWNer?	Queries the state of remote control. Following possible returns: REMOTE = Remote control is active for (addressed) output NONE = Remote control isn't active for (addressed) output
OUTPut_<B0> OUTPut?	Switches the (addressed) DC output ON , if the (addressed) output is already in remote control, or switches it OFF . The query form would return the actual output status, same as it can be queried via bit 11 and 14 or the Questionable Status register.
SYSTem:ERRor? SYSTem:ERRor:NEXT?	Queries the last error from the error queue. This would then be deleted from the queue so that possible further errors will shift to the top. Repeating the command would query all errors in the queue one by one until the buffer is empty. Return example: -223, "Too much data"

Command	Description
SYSTem:ERRor:ALL?	Queries all error which are currently in the queue and returns them in a combined string in which the last error comes first. Return example: -221,"Settings conflict;@1" , -100,"Command error" , -223,"Too much data" The use of @1 in the first error points to a conflict which occurred when trying to set something for Output 1 while it wasn't in remote control

3.3.14 Commands for protective features

The devices of series PS 2000 B feature a set of device alarms with adjustable thresholds (OVP, OCP) which serve to protect connected loads. The same thresholds can also be manually adjusted on the control (HMI) of the devices. Other alarms like OT have no adjustable parameter.

Command	Description
[SOURce:]VOLTage:PROTection[:LEVel]_<NRf>[Unit]	Defines the so-called OVP threshold for the (addressed) output. Example: VOLT:PROT 46 -> absolute short form, set the threshold to 46 V, which also suits for a 42 V model.
[SOURce:]CURRent:PROTection[:LEVel]_<NRf>[Unit]	Defines the so-called OCP threshold for the (addressed) output. Example: CURR:PROT 11 -> absolute short form, set the threshold to 11 A, which would only suit for models rated 10 A or higher.

3.3.15 Commands for adjustment limits

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

In case a set value is sent to the device that would exceed an adjustment limit, the device will ignore the value and put an error into the error queue. At the same time it's impossible to set the adjustment limit lower than the related set value. These commands are connected to the "Limits" settings as you can adjust them in the setup menu of the device. Also refer to the device manuals for details.

Command	Description
[SOURce:]VOLTage:LIMit:HIGH[?]_<NRf>[Unit]	Defines the upper limit of the voltage set value adjustment range. This is only executed if the limit value is equal to or higher than the set value. Example: VOLT:LIM:HIGH 30 -> absolute short form, sets the adjustment limit to 30 V, but only if the actual set value is 30 V or lower. This could be verified before with the VOLT? command.
[SOURce:]CURRent:LIMit:HIGH[?]_<NRf>[Unit]	Defines the upper limit of the adjustment range for the set value of current. This is only executed if the limit value is equal to or higher than the set value. Example: CURR:LIM:HIGH 6 -> absolute short form, sets the adjustment limit to 6 A, but only if the set value is 6 A or lower. This could be verified before with the CURR? command.

3.3.16 Commands for the tracking mode

The so-called tracking mode is only available with the Triple models. When activated, it couples the two addressable outputs so that Output 2 follows Output 1 in all regards of set values, limits, protection and DC on/off. Once tracking has been activated, Output 2 isn't available anymore for separate addressing. The status of tracking mode is indicated in the displays by "Tracking". This mode is furthermore stored permanently.

Command	Description
SYSTem:CONFig:TRACking_{ON OFF} SYSTem:CONFig:TRACking?	Activates tracking mode with ON or deactivates it with OFF or reads the current status. This status is the actual mode status as also signaled in the Questionable status register.

3.3.17 Further commands

Here are commands listed that can be used to query other information from the device.

Command	Description
SYSTem:NOMinal:VOLTage?	Reads the rated output voltage of device, respectively of the addressed output
SYSTem:NOMinal:CURREnt?	Reads the rated output current of the addressed output
SYSTem:NOMinal:POWer?	Reads the rated output power of the addressed output
SYSTem:DEVice:CLASs?	Reads the device class number, which can be used to distinguish between different series and sub series. For Single models of this series this number would always be 16, while it's 24 for the Triple models.



Elektro-Automatik

EA Elektro-Automatik GmbH & Co. KG

Development - Production - Sales

Helmholtzstraße 31-33

41747 Viersen

Germany

Fon: 02162 / 37 85-0

Fax: 02162 / 16 230

Mail: ea1974@elektroautomatik.de

Web: www.elektroautomatik.de