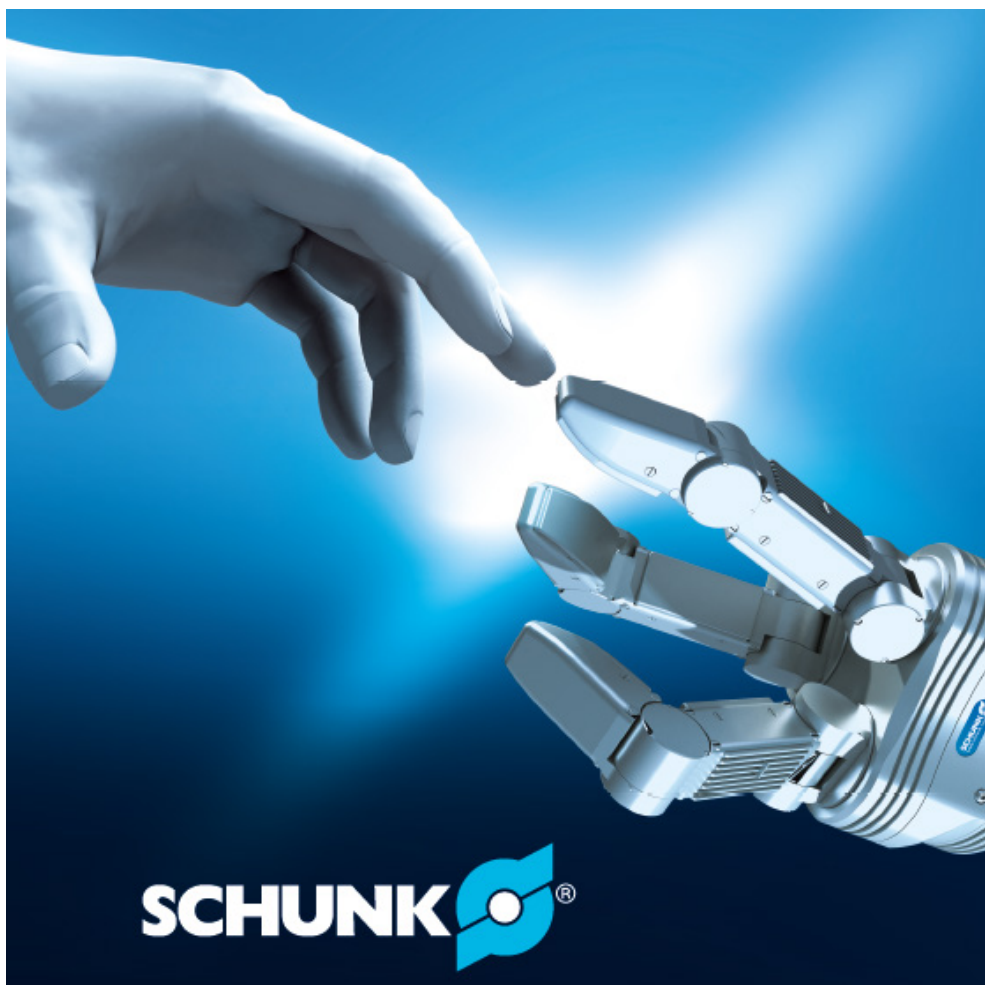


Motion Control Schunk V 1.60

Motion Control Schunk

Software manual



Imprint

Copyright:

This manual remains the copyrighted property of SCHUNK GmbH & Co. KG. It is solely supplied to our customers and operators of our products and forms part of the product. This documentation may not be duplicated or made accessible to third parties, in particular competitive companies, without our prior permission.

Technical changes:

We reserve the right to make alterations for the purpose of technical improvement.

Document number: 0389506

Edition: V 1.00 | 14/09/2015 | en

© SCHUNK GmbH & Co. KG

All rights reserved.

Dear customer,

congratulations on choosing a SCHUNK product. By choosing SCHUNK, you have opted for the highest precision, top quality and best service.

You are going to increase the process reliability of your production and achieve best machining results – to the customer's complete satisfaction.

SCHUNK products are inspiring.

Our detailed assembly and operation manual will support you.

Do you have further questions? You may contact us at any time – even after purchase.

Kindest Regards

Yours SCHUNK GmbH & Co. KG

Spann- und Greiftechnik

Bahnhofstr. 106 – 134

D-74348 Lauffen/Neckar

Tel. +49-7133-103-0

Fax +49-7133-103-2399

info@de.schunk.com

www.schunk.com



Reg. No. 003496 QM08



Reg. No. 003496 QM08

Table of contents

1	In general	8
1.1	Warnings.....	8
1.2	Electrical connection	8
1.3	Display elements	10
1.3.1	Factory settings.....	11
1.3.2	Bootting	11
1.4	Protocol	11
1.4.1	Data format	11
1.4.2	Data frame	14
1.4.3	Special features in serial communication	16
1.4.4	Special features with CAN	17
1.4.5	Special features with Profibus	18
1.4.6	Fragmentation.....	20
1.5	Unit system.....	22
1.5.1	Float.....	22
1.5.2	Integers	23
1.6	User administration.....	24
1.6.1	User	24
1.6.2	Diagnosis	24
1.6.3	Profi	26
1.6.4	Advanced.....	26
1.6.5	Root	26
1.7	Pseudo-absolute encoder.....	27
1.7.1	Prerequisites	27
1.7.2	Function.....	28
1.8	Standstill commutation	30
1.8.1	Prerequisites	30
1.8.2	Function.....	31
2	Commands	32
2.1	Axis index.....	32
2.2	Motion	33
2.2.1	CMD REFERENCE (0x92)	33
2.2.2	CMD REFERENCE HAND (0x97)	34
2.2.3	MOVE POS (0xB0).....	36
2.2.4	MOVE POS REL (0xB8)	38
2.2.5	MOVE POS TIME (0xB1)	40

2.2.6	MOVE POS TIME REL (0xB9).....	42
2.2.7	MOVE POS LOOP (0xBA)	44
2.2.8	MOVE CUR (0xB3)	44
2.2.9	MOVE VEL (0xB5)	45
2.2.10	MOVE GRIP (0xB7)	46
2.2.11	SET TARGET VEL (0xA0).....	47
2.2.12	SET TARGET ACC (0xA1)	48
2.2.13	SET TARGET JERK (0xA2)	48
2.2.14	SET TARGET CUR (0xA3)	49
2.2.15	SET TARGET TIME (0xA4)	50
2.2.16	SET TARGET POS (0xA6)	50
2.2.17	SET TARGET POS REL (0xA7).....	51
2.2.18	CMD STOP (0x91)	52
2.2.19	CMD FAST STOP (0x90)	53
2.3	Spontaneous messages	54
2.3.1	CMD INFO (0x8A)	54
2.3.2	CMD MOVE BLOCKED (0x93)	55
2.3.3	CMD POS REACHED (0x94).....	56
2.3.4	CMD ERROR (0x88)	56
2.3.5	GET STATE (0x95)	57
2.3.6	GET STATE AXIS (0x98)	57
2.3.7	CMD TOGGLE IMPULSE MESSAGE (0xE7)	58
2.4	Settings	59
2.4.1	SET CONFIG (0x81)	59
2.4.2	GET CONFIG (0x80)	61
2.4.3	SET CONFIG EXT (0x83)	64
2.4.4	GET CONFIG EXT (0x82).....	68
2.4.5	CMD_DEFAULT (0x8C).....	69
2.5	Commands for internal programming	70
2.5.1	SET PHRASE (0xC0)	71
2.5.2	GET PHRASES (0xC2)	72
2.5.3	PRG EXE (0xCF)	72
2.5.4	EXE PHRASE (0xC1).....	73
2.5.5	EXE PHRASE0 (0xD0)	73
2.5.6	EXE PHRASE1 (0xD1)	74
2.5.7	EXE PHRASE2 (0xD2)	75
2.5.8	EXE PHRASE3 (0xD3)	75
2.5.9	EXE PHRASE4 (0xD4)	76

2.5.10	EXE PHRASE5 (0xD5)	77
2.5.11	EXE PHRASE6 (0xD6)	77
2.5.12	EXE PHRASE7 (0xD7)	78
2.5.13	EXE PHRASE8 (0xD8)	78
2.5.14	EXE PHRASE9 (0xD9)	79
2.5.15	EXE PHRASE10 (0xDA)	80
2.5.16	EXE PHRASE11 (0xDB)	80
2.5.17	EXE PHRASE12 (0xDC)	81
2.5.18	EXE PHRASE13 (0xDD)	81
2.5.19	EXE PHRASE14 (0xDE)	82
2.5.20	EXE PHRASE15 (0xDF)	82
2.5.21	PRG GOTO (0xC3)	83
2.5.22	PRG WAIT (0xC4)	83
2.6	Other	84
2.6.1	GET STATE (0x95)	84
2.6.2	GET STATE AXIS (0x98)	87
2.6.3	CMD REBOOT (0xE0)	90
2.6.4	CMD DIO (0xE1)	91
2.6.5	CMD DIO AXIS (0xEB)	92
2.6.6	CMD GET DIO (0xE9)	93
2.6.7	CMD SET DIO (0xEA)	94
2.6.8	FLASH MODE (0xE2)	96
2.6.9	CMD DISCONNECT (0xE6)	97
2.6.10	CHANGE USER (0xE3)	98
2.6.11	CHECK MC PC COMMUNICATION (0xE4)	98
2.6.12	CHECK PC MC COMMUNICATION (0xE5)	100
2.7	Fragmentation	102
2.7.1	FRAG ACK (0x87)	102
2.7.2	FRAG START (0x84)	102
2.7.3	FRAG MIDDLE (0x85)	102
2.7.4	FRAG END (0x86)	104
2.8	Error messages	104
2.8.1	Error commands	105
2.8.2	Information codes and error codes	107
3	Configuration parameters	117
3.1	In general	117
3.2	EEPROM	117
3.2.1	Motor	117

3.2.2	Gears	123
3.2.3	Referencing	123
3.2.4	Controllers.....	132
3.2.5	Device	135
3.2.6	Positioning.....	146
3.2.7	Brake	150
4	Schunk Motion Tool	152
4.1	Prerequisites.....	152
4.2	First steps.....	152
4.3	Main window	155
4.3.1	Toolbar	156
4.3.2	Menu	156
4.3.3	Output window	158
4.3.4	Basic settings	159
4.3.5	Tools	161
4.4	Module window.....	163
4.4.1	Module states	165
4.4.2	Switch elements	166
4.4.3	Module parameters	167
4.4.4	Menu	169
4.4.5	Manual referencing.....	175
4.4.6	Start-up assistant	175
4.5	Tips.....	176
4.5.1	Supported languages	176
4.5.2	Vector CAN driver	176
4.5.3	Peak CAN driver	176
4.5.4	Softing CAN driver	176
4.5.5	ESD CAN interface	177
4.5.6	Siemens Profibus interface	177
4.5.7	Have the module status displayed automatically.....	177
4.5.8	Open the communication interface when starting	177
4.5.9	Data throughput with CAN	178
4.5.10	Configured modules under Profibus.....	178
4.5.11	Frequent timeouts in serial communication	178
4.5.12	Change individual EEPROM parameters	179
4.5.13	Do not maximize the main window	179
4.5.14	"Auto" communication type for the module.....	179

4.5.15 No automatic initialization.....	179
5 Remarks	180
5.1 Modules.....	180
5.2 Protocol	180
5.3 Serial communication.....	180
5.4 CAN	181
5.5 Profibus.....	181
6 Appendix.....	182
6.1 Selected examples.....	182
6.1.1 Serial communication	182
6.1.2 CAN.....	185
6.1.3 Profibus	188
6.2 CRC16 calculation for serial communication	192
6.3 Command	194
6.4 Information codes and error codes.....	197
6.5 Supported hardware	199
7 Release notes	200

1 In general

1.1 Warnings

To make risks clear, the following signal words and symbols are used for safety notes.

	 DANGER Danger for persons. Non-compliance will inevitably cause irreversible injury or death.
	 WARNING Dangers for persons. Ignoring a safety note like this can lead to irreversible injury and even death.
	 CAUTION Dangers for persons. Non-observance can cause minor injuries.
	NOTICE Material damage Information about avoiding material damage.

1.2 Electrical connection

The module has separate input terminals for the motor voltage and the logic voltage (24V DC). We recommend you connect the two voltages separately. This ensures that the logic continues to work in the event of the motor voltage overloading and that the module status is always known. In modules with a motor voltage > 24V DC, a separate power supply is unavoidable in any case, since the logic voltage must always be in a range between 18V DC and 32V DC.

**NOTICE****Permanent damage to the electronics possible!**

- If the power supply is separated, you must make sure that potential equalization is carried out between the two supply voltages (join the grounds)
- Only the positive pole may be disconnected, the motor-GND must always be connected.

NOTE

If the power supply for the logic and motor is separate, the power can be switched off from the control unit by means of relays, but the module is still active on the bus system.

When the power unit (motor) is supplied with power, use a power supply unit that is capable of supplying enough power to the respective module. Make sure there is adequate cable cross-section when cabling.

The voltage drop on the cable can be calculated using the following formula:

$$\Delta U = (2 \times I \times l) / (\chi \times A) \text{ with:}$$

I: Current input of load

l: Length of the line

χ : Electrical conductivity

Cu: $\chi = 56 \text{ ((m) / (\Omega \times \text{mm}^2))}$

Al: $\chi = 35 \text{ ((m) / (\Omega \times \text{mm}^2))}$

A: Conductor cross-section

1.3 Display elements

All modules have three display elements in the form of LEDs.

NOTE

In some designs, however, these displays are not external and thus not visible.

A green LED (POW LED) is used as the status display for the motor voltage. If the LED does not light up, or only very weakly, then the motor supply voltage must be checked (24V - 48V DC).

The two other LEDs (green or red) indicate the status of the logic circuits:

Relevance of the LEDs

LED1 (green)	LED2 (red)	Meaning
permanently on	permanently on	The module is in <i>Flash mode</i> (☞ 2.6.8, Page 96) or no bus system is active.
on	flashes	New firmware is being transmitted (from V1.20)
briefly on	briefly on	The module is booting.
on	off	The module is ready for operation and the bus system is active.
flickers	off	Data is being exchanged.
off	off	Logic supply voltage is not present. If the two LEDs were previously briefly on (boot phase), then the initialization of the connected bus system has not yet been successful. Please check the bus cable. Is the master active?
off	on / flashes	An <i>error</i> (☞ 2.8.1, Page 105) has occurred in the module.
off / on	on / off	Profibus is active, but not yet in the "Data exchange" mode - or automatic interface detection is operating.
flickers	flashing / flickering / on	Data from the "Main interface" is being exchanged. At the same time, the <i>diagnosis interface</i> (☞ 1.6.2, Page 24) is active and data is also being exchanged through this.
off / on quickly	on / off quickly	Firmware is in an undefined state. (should never happen!)

1.3.1 Factory settings

If the module is reset to the factory setting, then the following values are overwritten or reset in EEPROM:

- Module ID
- Baudrate for CAN bus
- Baudrate for serial communication
- Communication system The exact values can be referred to in the data sheet for the respective product.

1.3.2 Booting

After the module has been successfully started, several parameters for movements are already set to start values. It is therefore possible to put the module directly into operation without having to previously set parameters. The following parameters are set:

- "Target speed" ([↗ 2.2.11, Page 47](#))
in [%] as [%] of the maximum value ([↗ 3.2.1, Page 117](#)). -> 10%
- "Target acceleration" ([↗ 2.2.12, Page 48](#))
as [%] of the maximum value ([↗ 3.2.1, Page 117](#)). -> 10%
- "Target jerk" ([↗ 2.2.13, Page 48](#))
as [%] of the maximum value ([↗ 3.2.1, Page 117](#)). -> 50%
- "Target current" ([↗ 2.2.14, Page 49](#))
Rated current ([↗ 3.2.1, Page 117](#)).
- Spontaneous messages ([↗ 2.3.7, Page 58](#)) activated.
- User is set to "User" ([↗ 1.6, Page 24](#)) gestellt

1.4 Protocol

1.4.1 Data format

Data is sent from the modules in the Intel format (little-endian) and interpreted in this same format when being received.

NOTE

If you are not sure when creating your own drivers, you can check the "rotation of the bytes" using "CHECK MC PC COMMUNICATION" ([↗ 2.6.11, Page 98](#)) or "CHECK PC MC COMMUNICATION" ([↗ 2.6.12, Page 100](#)) with defined test data.

Floating-point numbers The IEEE 754 standard (floating-point numbers) was developed in the early 1980s in order to, among other things, attain consistent floating-point number representation using various computer architecture. This standard is adhered to if the parameters are sent as floating-point numbers to the module or are transmitted from the module to the control unit. A floating-point number is shown here with 32 bits.

Sign bit	Exponent	Exponent
1 bit (bit 32)	8 bits (bit 23.. bit 30)	23 bits (bit 1.. bit 22)
<i>s</i>	<i>e</i>	<i>f</i>

Since the mantissa is always standardized to "1", only the digits after the decimal point are saved. The leading "1" is not saved concurrently. A float value can be calculated as follows.

$$(-1)^s \times 2^{e-127} \times (1.f)_{\text{bin}}$$

Some examples:

	Sign	Exponent	Mantissa
	1 bit	8 bits	23 bits
7/4	0	01111111	11000000000000000000000
-34.432175	1	10000100	00010011011101010001100
-959818	1	10010010	11010100101010010100000
+0	0	00000000	00000000000000000000000
-0	1	00000000	00000000000000000000000
$2^{-126} = 1.175 \times 10^{-38}$			
Smallest positive number	0	00000001	00000000000000000000000
$(2-2^{-23}) 2^{127} = 3.403 \times 10^{38}$			
Largest positive number	0	11111110	11111111111111111111111
Infinite	0	11111111	11111111111111111111111
NaN	0	11111111	Not all "0" or "1"
$2^{-23} = 1.192 \times 10^{-7}$			
Smallest number to be resolved	0	01101000	00000000000000000000000
2^{-128}	0	00000000	01000000000000000000000

Two's complement The two's complement is an option for representing negative numbers in the binary system. In the module, the two's complement is required for the representation of negative integers. (Integer unit system ([🔗 1.5.2, Page 23](#))).

Positive numbers are provided in the two's complement with a leading 0 (sign bit) and otherwise not changed. Negative numbers are provided with a leading 1 as the sign bit and coded as follows: All digits in the relevant positive number are negated. 1 is added to the result. Exemplary conversion of the negative decimal number -4(dec) into the two's complement:

- 1 Ignore sign and recalculate into binary system: $4_{\text{dec}} = 00000100_{\text{bin}} = 0x\ 04_{\text{hex}}$
- 2 Invert because negative: $11111011_{\text{bin}} = 0x\ FB_{\text{hex}}$
- 3 Add one because negative: $11111011_{\text{bin}} + 00000001_{\text{bin}} = 11111100_{\text{bin}} = 0x\ FC_{\text{hex}} = -4_{\text{dec}}$

Somewhat more mathematical:

If x is a negative number, then x is calculated in the two's complement (x_z) with n places as follows:

$$x_z = 2^n - |x|$$

Accordingly, the following also applies:

$$x_z + |x| = 2^n$$

Since the modules in the "Integer unit system" ([🔗 1.5, Page 22](#)) always work with Int32 (4 bytes), the succession of bytes for negative numbers x (e.g. -112) can be simply calculated as follows:

$$y = 4294967296_{\text{dec}} - |x| \rightarrow y = 4294967296_{\text{dec}} - 112_{\text{dec}} = 4294967184_{\text{dec}}$$

$$y = 0x\ 100000000_{\text{hex}} - |x| \rightarrow y = 0x\ 100000000_{\text{hex}} - 0x\ 70_{\text{hex}} = 0x\ FFFFFFF90_{\text{hex}}$$

1.4.2 Data frame

The data frame of the Motion protocol always includes the following elements:

- D-Len (1 byte)
- COMMAND CODE (1 byte)



Data frame

D-Len (Data Length) specifies the number of subsequent items of user data including the command byte. The data frame consists of one byte, therefore a Motion protocol message can transfer a maximum of 255 data bytes.

The D-Len byte is always followed by the command code, consisting of one byte. If necessary, the command code is followed by the relevant parameters required. The command code can also be extended through "Under command codes", if needed.

An example for this is the *GET CONFIG* ([👉 2.4.2, Page 61](#)) command.

As an option, the "Axis index" ([👉 2.1, Page 32](#)) can be inserted at the end.

NOTE

The axis index is only required for systems in which several motors can be addressed via one address. Should the axis index be used for systems with one motor, then the index is always "0"

All commands sent are immediately confirmed by the module with a response (acknowledge). This response also uses the data frame described above. (D-Len, command code, parameters, if necessary). If the request has been successfully processed, D-Len always has a value that is not equal to "0x02". If the request failed, D-Len has the value "0x02". The cause of the failed request 70 is described in the two following bytes.

The cause of the *failed request* ([👉 2.8.2, Page 107](#)) is described in the two following bytes.

Furthermore, the modules have the capacity to send messages without a request having been made. The data frame is also

adhered to in the event of such "spontaneous messages". A "spontaneous message" is triggered in the following events:

- When a serious error has occurred.
- When a motion has been concluded correctly.
- When there is a regular *status message* ([👉 2.3, Page 54](#)), if activated.

1.4.3 Special features in serial communication

Since the serial interface was not intended as a bus system in the development phase, some elements have to be integrated in addition to the data frame so that several modules can be operated with one, single serial interface.



Data frame for serial communication

Firstly, two bytes (Group/ID), which are used to determine which module is going to be addressed or from which module the response originated, are attached at the front of the data frame. Only the first three bits are used by the first byte. The second byte represents the unique module ID. => up to 255 modules can be addressed. The first three bits of the first byte are coded as follows:

- 0x03 Module error message
- 0x05 Message from the master to a module
- 0x07 Response from the module

All other states are not used.

NOTE

This way of clearly identifying a module with 11 bits was taken from the CAN protocol.

In order to be able to guarantee reliable serial data transmission, a checksum (CRC16 => 2 bytes), which is attached at the end of the data frame, is generated for all data, including Group/ID, D-Len and Cmd. An algorithm for calculating a CRC16 can be found in the *Appendix* ([👉 6.2, Page 192](#)).

1.4.4 Special features with CAN

As a message oriented bus system, CAN needs - in addition to the data frame - a unique identifier allocated to the message. The modules support the standard 11-bit identifier. The lower 8 bits are used here for the unique module ID. Therefore, a maximum of 255 modules can be addressed. The 3 bits still remaining are coded as follows:

- 0x03 Module error message
- 0x05 Message from the master to a module
- 0x07 Response from the module

All other states are not used.

- A message to the module thus has the following identifier 0x5XX. (XX = module address in hex representation)
- A message from the module has the identifier 0x7XX. (XX = module address in hex representation)
- In the event of an error, the messages from the module to the master are sent with the identifier 0x3XX. (XX = module address in hex representation).

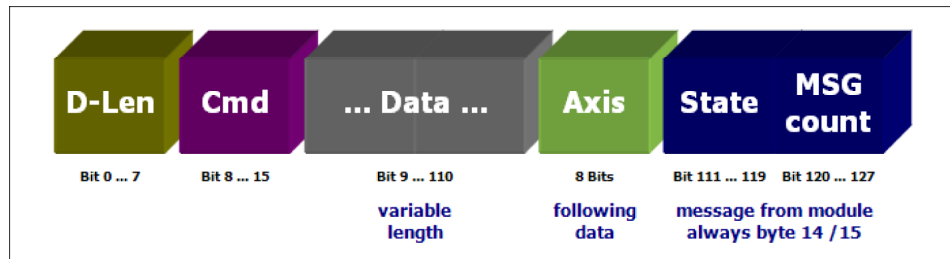
A maximum 8 bytes can be sent in a CAN message. Under certain circumstances, it will be necessary to pack a longer data frame (D-Len > 7) into several CAN messages. This is done by using the fragmentation protocol ([👉 1.4.6, Page 20](#)).

NOTE

Fragmentation is normally not required, since all commands needed to operate the modules can be packed into one CAN message.

1.4.5 Special features with Profibus

There are the following special features with Profibus DPV0: The maximum length of the data to be transmitted all at once from the master to the module is limited to 8 bytes. This can be used to fully operate a module. (A maximum 7 bytes are needed for a message from the master to the module)



Data frame for Profibus

The maximum length of the data sent from the module to the master (response) is limited to 16 bytes (GSD file). If you need to send/receive larger quantities of data, a *fragmentation* ([1.4.6, Page 20](#)) may be necessary. The longest message arising in normal operation from the module to the master can be accommodated in 16 bytes. There will even be 2 bytes still remaining. Contained in these 2 bytes, which are always at the end of the Profibus message (byte 14, byte 15), are

- 1 the actual *status* ([2.6.1, Page 84](#)) of the module (byte 14) and
- 2 a so-called command counter (MsgCount) (byte 15).

If bytes 10-13 are not required, the actual position in the respective *unit system* ([1.5, Page 22](#)) can be read out here.

NOTE

Bytes 10-13 are only used in a response to a GET STATE ([2.6.1, Page 84](#)) if all data (position, speed, and current) are requested. Bytes 10-15 are used for data in fragmented messages.

NOTE

Only the upper 8 bits of the status word are written. The error code is not used. You can use the extended diagnosis under Profibus on the one hand for this, and on the other, error code ([☞ 2.8.1, Page 105](#)) is displayed in the output data in the event of an error. If a message from the master to the module is sent, then with Profibus the MsgCount is increased by 1 in addition to the response. This ensures that every request is confirmed despite possible spontaneous messages.

NOTE

A spontaneous message ([☞ 2.3, Page 54](#)) does not increase the MsgCount!

If, for example, you want to move to a position where the module is currently located, the module will signal "Command understood" and "Position reached" immediately in the next Profibus cycle. Since in some circumstances a control unit connected to the Profibus does not query the data in every Profibus cycle, the acknowledgement of (response to) the motion command could go missing. The MsgCount ensures that a confirmation of the request was received. *Status-byte* ([☞ 2.6.1, Page 84](#)) (byte 14) is used to constantly receive the current information on the status of the present module.

NOTE

The last bit for MsgCount can be evaluated as a toggle-bit. During data transmission from the master to the module the byte 8 not used can be used as toggle-byte, or bit 63 as toggle-bit.

Groups are fully supported by the SYNC / FREEZE mechanism implemented in Profibus.

Changing the address is possible at any time via the "Set slave address" (SAP 55) service. "Real no add change" is saved in the *group byte* ([☞ 3.2.5, Page 135](#)). A set "Real no add change" (0xFF) can thus be deleted again using a *new configuration of the group byte* ([☞ 2.4.1, Page 59](#)).

If consistent data transmission is not possible, then the following options can be used to operate the module:

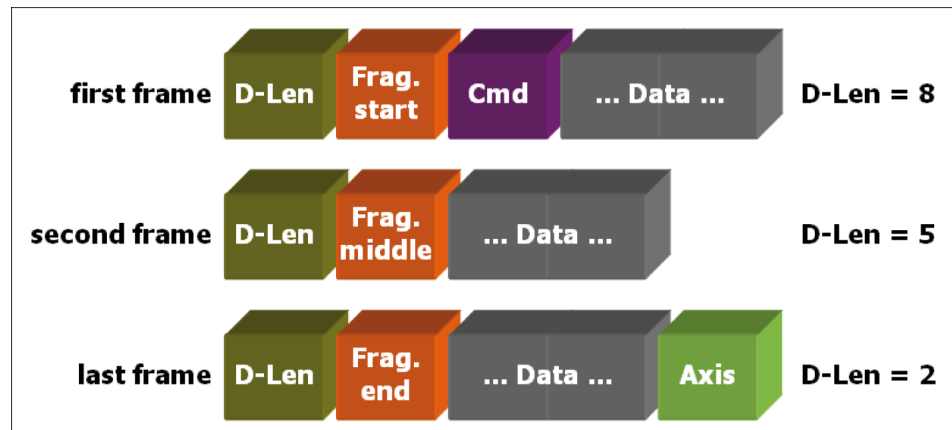
- Using the SYNC, UNSYNC mechanism.
- Set D-Len (first byte) to 0x00, populate all data and, finally, set D-Len to the required value. Populate all data and set the D-Len as soon as all data are present.

1.4.6 Fragmentation

NOTE

The fragmentation of messages is not required for normal operation!

Should a fragmentation of messages be required, this is done as follows:



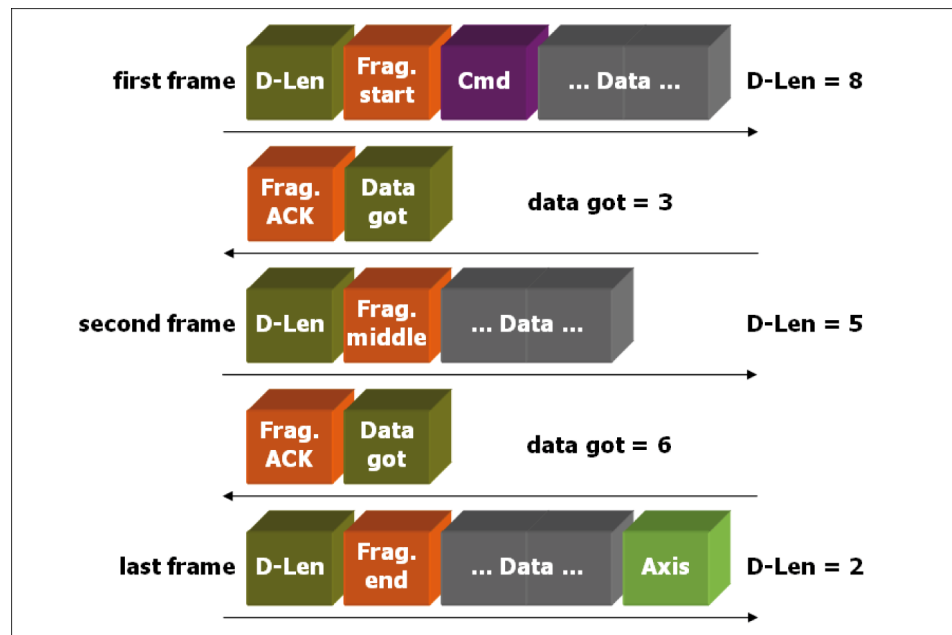
Fragmentation

The length of the user data still following is sent at the beginning of each message. Then a fragment identifier is sent. This fragment identifier is **not** recorded in the length byte (D-Len).

- FragStart -> First fragment ([☞ 2.7.2, Page 102](#)).
- FragMiddle -> A middle fragment ([☞ 2.7.3, Page 102](#)).
- FragEnd -> Last fragment ([☞ 2.7.4, Page 104](#)).

These individual fragments can thus be reassembled into a complete *data frame* ([☞ 1.4.2, Page 14](#)) which can then be interpreted.

Special feature of Profibus



Profibus fragmentation

Since, in Profibus, a "token" is constantly underway out of which the respective participants pick out the valid data for themselves or register the required data for the master, each fragment received must be confirmed using "FRAG ACK" ([2.7.1, Page 102](#)) and the D-Len byte for the received fragment. If a fragmented message is sent to the master, then the master has to confirm every single fragment using "FRAG ACK" and the D-Len byte for the received fragment so that the module can send the next fragment. If the master sends a fragmented message to the module, then it has to delay sending the following fragment until the module has confirmed the receipt of the fragment ("FRAG ACK" and the D-Len byte for the received fragment). The last fragment does not have to be confirmed.

1.5 Unit system

All parameter data affected by units are transferred in the *preset unit system* ([👉 3.2.5, Page 135](#)).

1.5.1 Float

- [mm] All parameters are transferred as float values =>
Position [mm], speed [(mm)/(s)], acceleration [(mm)/(s²)], jerk [(mm)/(s³)], current values [A], time [s]
- [m] All parameters are transferred as float values =>
Position [m], speed [(m)/(s)], acceleration [(m)/(s²)], jerk [(m)/(s³)], current values [A], time [s]
- [inches] All parameters are transferred as float values =>
Position [inches], speed [(inch)/(s)], acceleration [(inch)/(s²)], jerk [(inch)/(s³)], current values [A], time [s]
- [rad] All parameters are transferred as float values =>
Position [rad], speed [(rad)/(s)], acceleration [(rad)/(s²)], jerk [(rad)/(s³)], current values [A], time [s]
- [degrees] All parameters are transferred as float values =>
Position [degrees], speed [(degrees)/(s)], acceleration [(degrees)/(s²)], jerk [(degrees)/(s³)], current values [A], time [s]
- [internal] All parameters are transferred as float values =>
Position [internal], speed [(internal)/(s)], acceleration [(internal)/(s²)], jerk [(internal)/(s³)], current values [A], time [s]
This unit system should only be used for test purposes. All data are calculated internally in this unit system, which is based on motor revolutions. Gear ratios or reduction ratios are not taken into account.

1.5.2 Integers

- [μm] All parameters are transferred as integer values =>
Position [μm], speed [$(\mu\text{m})/(\text{s})$], acceleration [$(\mu\text{m})/(\text{s}^2)$], jerk [$(\mu\text{m})/(\text{s}^3)$], current values [mA], time [ms]
- [$\mu\text{degrees}$] All parameters are transferred as integer values =>
Position [$\mu\text{degrees}$], speed [$(\mu\text{degrees})/(\text{s})$], acceleration [$(\mu\text{degrees})/(\text{s}^2)$], jerk [$(\mu\text{degrees})/(\text{s}^3)$], current values [mA], time [ms]
- [μinches] All parameters are transferred as integer values =>
Position [μinches], speed [$(\mu\text{inch})/(\text{s})$], acceleration [$(\mu\text{inch})/(\text{s}^2)$], jerk [$(\mu\text{inch})/(\text{s}^3)$], current values [mA], time [ms]
- [millidegrees] All parameters are transferred as integer values =>
Position [millidegrees], speed [(millidegrees)/(s)], acceleration [(millidegrees)/(s²)], jerk [(millidegrees)/(s³)], current values [mA], time [ms]

NOTE

The configuration parameters ([👉 3.2, Page 117](#)) must be transferred in the respective unit system.

1.6 User administration

The module is equipped with a user administration to specifically protect certain actions. The user rights can be switched via "CHANGE USER" ([👉 2.6.10, Page 98](#)).

1.6.1 User

Is the standard user always activated after the module is switched on. The user can fully operate the module. Parameterizing is only permitted for the most important *parameters* ([👉 3.2, Page 117](#)).

1.6.2 Diagnosis

Is the diagnosis user. If this user is registered, then another interface will be opened*. This can be used to record bus traffic on the primary interface, or also to be able to call up targeted information on a very restricted basis. Module parameterizations are possible.

* CAN or PROFIBUS active => Serial communication is also opened; serial communication active => CAN will also be opened.

If the *Diagnosis user* ([👉 2.6.10, Page 98](#)) is activated via the primary interface, then a "recording" is active.

The "CMD TOGGLE IMPULSE MESSAGE" ([👉 2.3.7, Page 58](#)) command can be used to convert the status of the secondary interface from "listen" to "parameterize". This command must be sent via the secondary interface.

In the "parameterize" status, commands can be sent via the secondary interface.

If an error has occurred in the module, you can log in to the module directly as the *diagnosis user* ([👉 2.6.10, Page 98](#)) via the secondary interface.

Select via the parameterization software "find diagnosis" under "modules" (only by RS232).

NOTE

Please note! An active control unit for the modules is not possible!

NOTE

After completion of "diagnostic work", the module needs to restart (separating the supply voltage of the logic and connect again)!

NOTE

The behavior of the module can deviate from the "normal" behavior when the diagnosis interface is activated.

1.6.3 Profi

Is the professional user having the full range of functions of the "user" and can also adjust other parameters. Incorrect parameterization can result in unanticipated module behavior. However, the module cannot be destroyed.

The standard password for the professional rights is "Schunk".

The standard password for the professional rights is "HDAG".

1.6.4 Advanced

Is the advanced user having the full range of functions of the "Pro-fi" and can also adjust further parameters.



NOTICE

Incorrect operation or incorrect parameterization can lead to the destruction of the electronics or the motor.

1.6.5 Root

Is the root user having full access to the module. All parameters can be adjusted and other functions are accessible for test purposes.



NOTICE

Incorrect operation or incorrect parameterization can lead to the destruction of the electronics or the motor.

1.7 Pseudo-absolute encoder

1.7.1 Prerequisites

The modules support the Pseudo-absolute encoder function if the following prerequisites are met:

- Brake
- FRAM (*hardware version* [\(☞ 2.4.2, Page 61\)](#) odd-numbered)
- *Position measuring system* [\(☞ 3.2.6, Page 146\)](#) resolver
- or position measuring system Encoder with index track and
 - DC [\(☞ 3.2.1, Page 117\)](#) motor type
 - or BLDC motor type
 - or PMSM motor type with *standstill commutation* [\(☞ 1.8, Page 30\)](#)

1.7.2 Function

When the brake is applied, the current position is saved in a non-volatile memory. If the logic voltage is turned off, then an attempt is made to save the current position with the remaining residual energy.

Resolver When the module is switched on again, the position saved beforehand will now be compared with a control value. Should this check be successful, then the position saved will be compared with the current position of the resolver. If these positions are also equal, then the module does not need to be re-referenced.

NOTE

If the resolver is turned by precisely one revolution when deenergized, then the displayed position will be faulty when reactivation takes place.

Encoder with index track When the module is switched on again, the position saved beforehand will be compared with a control value. Should this check be successful, then the saved position will become the current position of the module. The interval between the next index pulse is calculated at the same time. In the first motion command that follows, the calculated interval is compared with the measured interval when the index pulse is reached. The module is deemed as referenced if the two values agree with each other. In addition, the index pulse must be reached within a certain period of time after the first motion command has been sent. If an error 76 occurs during the movement to the index pulse, then the referencing will be deleted.

If an *error 76* occurs during the movement to the index pulse, then the referencing will be deleted.

NOTE

After a successful reference movement, the index pulse has to be run over at least once in order to activate the function.

NOTE

If the encoder is moved in a deenergized state, then it is possible that the module will carry out a motion to the next index pulse (max. one motor revolution) after it has been switched on again with the incorrect position.

NOTE

If the encoder is turned by precisely one revolution when deenergized, then the displayed position will be faulty when reactivation takes place.

1.8 Standstill commutation

1.8.1 Prerequisites

- DC ([👉 3.2.1, Page 117](#)) motor type
- or BLDC motor type
- or PMSM motor type and
 - *Position measuring system* ([👉 3.2.6, Page 146](#)) "Encoder with index track" and available Hall sensors
 - or resolver position measuring system



NOTICE

Direction of motion for block commutation and sine commutation must agree. If the directions of rotation are different, then the phases must be changed and the commutation table ([👉 3.2.1, Page 117](#)) adapted.

1.8.2 Function

If all prerequisites are met, then the module will attempt to carry out the standstill commutation. In modules with absolute-value measuring systems, the sine commutation can be activated directly after switch-on, since the position of the "sine pointer" is known.

In modules with an encoder measuring system, the position of the "sine pointer" is not known until the index pulse has been reached (by the drive with block commutation, there can be a little less force). This is why the module is moved to the first index pulse with block commutation and then converted to sine commutation.

The position of the "sine pointer" to the index pulse is set or "readjusted" using the *Positioning offset* ([↗ 3.2.6, Page 146](#)) parameter. If this value is set to "0", then a "sine pointer" - which is *saved* ([↗ 3.2.6, Page 146](#)) - is searched for in the next motion command by energizing the motor phases. The *referencing* ([↗ 1.7, Page 27](#)) is deleted in the process.

NOTE

Zur Zeigersuche sollte das Modul in alle Richtungen frei beweglich sein. Modul wird ruckartig bis zu zwei Motorumdrehungen bewegt. Eine Kommunikation mit dem Modul ist in dieser Zeit nicht möglich.

2 Commands

All examples only list the data frame. The special features of the various bus systems are described as from the chapter "*Special features in serial communication*" ([👉 1.4.3, Page 16](#)). Several selected examples for the various bus systems are listed in the *Appendix* ([👉 6.1, Page 182](#)).

NOTE

It is assumed that with all listed examples [mm] is set as the unit system

In all examples, only the *necessary parameters* are listed, not the *optional parameters*. In the examples, "M" stands for master and "S" for slave (module).

2.1 Axis index

Apart from a few exceptions, an axis index can be added as an additional parameter in the commands described in the following sections as an option. This is required in modules that control several axes or even several robots. If available, the "axis index" added limits the effect of the respective command either to a specific axis or to a specific robot.

If the highest value bit (MSB, bit no. 7) is deleted, then the "axis index" parameter designates the number of an axis. If the highest value bit (MSB, bit no. 7) is set, then the lower value bits (bits nos. 6..0) for the "axis index" parameter designate the number of a robot.

- The "axis index" parameter is transmitted with one byte.
- The "axis index" parameter is added at the end of the normal parameter of a command.
- The parameter increases the D-Len of the command by one.
- A value of 0 for the "axis index" can be used in modules with only one axis.
- A value of [1..128] for the "axis index" denotes the number of an axis [1..128].
- A value of [129..255] for the "axis index" denotes the number of a robot [1..127] ("axis index" - 128).

Exceptions: In the following commands, *no* "axis index" can be added:

- *CMD DIO* ([👉 2.6.4, Page 91](#))

However, if an "axis index" is required here, then the ([☞ 2.6.5, Page 92](#)) command can be used as an alternative.

- *GET STATE* ([☞ 2.6.1, Page 84](#))

However, if an "axis index" is required here, then the *GET STATE AXIS* ([☞ 2.6.2, Page 87](#)) command can be used as an alternative.

2.2 Motion

2.2.1 CMD REFERENCE (0x92)

COMMAND CODE: 0x92

DESCRIPTION: A reference movement is executed. The type of referencing is defined once in *configuration parameter* ([☞ 3.2.3, Page 123](#)).

PARAMETER (Master -> Slave): None

EXAMPLE:

	D-Len	Cmd	Param
M->S	0x01	0x92	
S->M	0x03	0x92	0x4F 0x4B

MISCELLANEOUS: Spontaneous response possible. Depending on the *referencing type* ([☞ 3.2.3, Page 123](#)) this can result in „*CMD MOVE BLOCKED*“ ([☞ 2.3.2, Page 55](#)) or "*CMD POS REACHED*" ([☞ 2.3.3, Page 56](#)). This depends on the "*MOVE ZERO AFTER REFERENCING*" ([☞ 3.2.3, Page 123](#)) flag. A set flag triggers a positioning movement after the referencing => "*CMD POS REACHED*" ([☞ 2.3.3, Page 56](#)). During referencing, the *set parameters* ([☞ 3.2.3, Page 123](#)) are adopted for the movements. After referencing, the values set last (prior to the reference movement) are re-established. Please also observe the special features of the respective *position measuring system* ([☞ 3.2.6, Page 146](#)).

NOTE

Under certain conditions, a referencing process successfully performed once is maintained after the operating voltages have been switched off. See "Pseudo-absolute encoder" ([☞ 1.7, Page 27](#)).

NOTE

Prior to a reference movement, all workpieces must be removed for a gripper.

2.2.2 CMD REFERENCE HAND (0x97)

COMMAND CODE: 0x97

DESCRIPTION: A manual reference movement is executed. If necessary, any initialization movements required are carried out at first (sine pointer search, index pulse search). This process may require up to 30 seconds. If the module has carried out the initialization run successfully, the *warning* ([2.8.1, Page 105](#)) "Subcode" "NOT REFERENCED" ([2.8.2, Page 107](#)) is reported. After "CMD REFERENCE HAND" has been sent again, "MOVE POS TIME REL" ([2.2.6, Page 42](#)) can now be used for moving to a defined reference mark. This next step is confirmed with "NOT REFERENCED". (Acceleration and speeds must not exceed the values in the *Referencing* ([3.2.3, Page 123](#))) section. Sending "CMD REFERENCE HAND" again will set the current position to the *reference offset* ([3.2.3, Page 123](#)) parameterized beforehand. Referencing is concluded.

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): "OK" (0x4F4B) if successful. The module executes the next step.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0x97		
S->M	0x03	0x97	0x4F 0x4B	
S->M	0x02	0x89	0x06	Initialization finished
M->S	0x01	0x97		
S->M	0x02	0x97	0x06	"Jog mode" activated
M->S	0x05	0xB9	0xCD 0xCC 0x4C 0x3E	The module is moved
.				
.				

	D-Len	Cmd	Param	
.				
M->S	0x01	0x97		
S->M	0x03	0x97	0x4F 0x4B	Reference position set

MISCELLANEOUS: Referencing can be canceled at any time using *Stop* ([↗ 2.2.18, Page 52](#)). A set "*MOVE ZERO AFTER REFERENCING*" ([↗ 3.2.3, Page 123](#)) is ignored. After referencing, the values set last (prior to the reference movement) are re-established.

NOTE

If manual referencing has been performed successfully once, then the referencing type set previously will automatically be converted to "Manual".

NOTE

This procedure can be carried out simply by using the supplied "Motion Tool Schunk" software ([↗ 4.4.5, Page 175](#)).

NOTE

Under certain conditions, a referencing process successfully performed once is maintained after the operating voltages have been switched off. See "Pseudo-absolute encoder" ([↗ 1.7, Page 27](#))

2.2.3 MOVE POS (0xB0)

COMMAND CODE: 0xB0

DESCRIPTION: Moves the module to a specified position. The position is set in the configured *unit system* ([↗ 1.5, Page 22](#)). The positioning movement is based on the *configured motion profile* ([↗ 3.2.5, Page 135](#)).

PARAMETER (Master -> Slave)*:

- *Position* ([↗ 2.2.16, Page 50](#)) (optional), in the configured unit system.
- *Speed* ([↗ 2.2.11, Page 47](#)) (optional) which is used for the positioning movement. Not relevant for the motion profile "No ramp" ([↗ 3.2.5, Page 135](#)).
- *Acceleration* ([↗ 2.2.12, Page 48](#)) (optional) which is used for the positioning movement. Not relevant for the motion profile "No ramp" ([↗ 3.2.5, Page 135](#)).
- *Current* ([↗ 2.2.14, Page 49](#)) (optional) which must not be exceeded in the positioning movement. If the controller structure "CURRENT SPEED" ([↗ 3.2.4, Page 132](#)) is active, then this value must be transferred (jerk is required). The value has to be "0", otherwise "INFO WRONG PARAMETER" ([↗ 2.8.2, Page 107](#)) will be generated.
- *Jerk* (optional) which is used for the positioning movement. Should the motion profile not be equal to "jerk-limited" ([↗ 3.2.5, Page 135](#)) sein, , then this value cannot also be transferred (("INFO WRONG PARAMETER" ([↗ 2.8.2, Page 107](#))).

RESPONSE (Slave->Master): If possible, the timer is returned to the module for the expected movement needs. If the calculation of time should not be possible, „OK" (0x4F4B) is returned in case of success and the module executes the movement.

NOTE

If the traversing profile be 'configured to "jump" the jerk-limited traversing profile is used. When traversing profile is used "jump", the drive would run for violent motion.

MISCELLANEOUS: Spontaneous response when the position is reached "*CMD POS REACHED*" ([☞ 2.3.3, Page 56](#)) or when the positioning movement is blocked "*MOVE BLOCKED*" ([☞ 2.3.2, Page 55](#)).

All parameters must be transferred in the specified order. If just the current is to be specified, then it is vital that the position, speed, and acceleration are also specified. The following parameters do not have to be transferred as well. All parameters are retained up to the restart or until these parameters are changed.

NOTE

If new positioning parameters are entered during a movement, the movement might be temporarily halted. If you wish to enter new positions during a movement, e.g. to move along curved paths or to use your own path interpolators, then the "*MOVE POS TIME*" ([☞ 2.2.5, Page 40](#)) command must be used.

2.2.4 MOVE POS REL (0xB8)

COMMAND CODE: 0xB8

DESCRIPTION: Moves the module a specified distance. The change of position is set in the configured *unit system* ([↗ 1.5, Page 22](#)). The positioning movement is based on the *configured motion profile* ([↗ 3.2.5, Page 135](#)).

PARAMETER (Master -> Slave)*:

- *Change of position* ([↗ 2.2.17, Page 51](#)) (optional), in the configured unit system)
- *Speed* ([↗ 2.2.11, Page 47](#)) (optional) which is used for the positioning movement. Not relevant for the motion profile "No ramp" ([↗ 3.2.5, Page 135](#)).
- *Acceleration* ([↗ 2.2.12, Page 48](#)) (optional) which is used for the positioning movement. Not relevant for the motion profile "No ramp" ([↗ 3.2.5, Page 135](#)).
- *Current* ([↗ 2.2.14, Page 49](#)) (optional) which must not be exceeded in the positioning movement. If the controller structure "CURRENT SPEED" ([↗ 3.2.4, Page 132](#)) is active, then this value must be transferred (since jerk is required). The value has to be "0", otherwise "INFO WRONG PARAMETER" ([↗ 2.8.2, Page 107](#)) will be generated.
- *Jerk* (optional) which is used for the positioning movement. Should the motion profile not be equal to "jerk-limited" ([↗ 3.2.5, Page 135](#)), then this value cannot also be transferred ("INFO WRONG PARAMETER" ([↗ 2.8.2, Page 107](#))).

NOTE

If the motion profile is configured to "jump", then the jerk-limited motion profile is used. In motion profile "jump", the drive would execute vigorous movements. In motion profile "jump", the drive would execute vigorous movements.

RESPONSE (Slave->Master): If possible, the time that the module needs for the movement is returned. If a calculation of the time should not be possible, then the request will be confirmed with "OK" (0x4F4B) in the event of success, and the module will execute the movement. If a calculation of the time should not be possible,

then the request will be confirmed with "OK" (0x4F4B) in the event of success, and the module will execute the movement.

EXAMPLE1:

	D-Len	Cmd	Param	
M->S	0x05	0xB8	0x00 0x00 0x20 0x41	Move further by 10.0 [mm]
S->M	0x05	0xB8	0xCD 0xCC 0x04 0x41	Will expectedly need 8.3 [s] for this

EXAMPLE2:

	D-Len	Cmd	Param	
M->S	0x01	0xB8		Continue to move by the distance set last
S->M	0x05	0xB8	0xCD 0xCC 0x04 0x41	Will expectedly need 8.3 [s] for this

Miscellaneous: Spontaneous response when the position is reached "*CMD POS REACHED*" ([👉 2.3.3, Page 56](#)) or when the positioning movement is blocked "*MOVE BLOCKED*" ([👉 2.3.2, Page 55](#)). All parameters must be transferred in the specified order. If just the current is to be specified, then it is vital that the change of position, speed, and acceleration are also specified. The following parameters do not have to be transferred as well. All parameters are retained up to the restart or until these parameters are changed.

NOTE

If new positioning parameters are entered during a movement, the movement might be temporarily halted. If you wish to enter new positions during a movement, e.g. to move along curved paths or to use your own path interpolators, then the "MOVE POS TIME" 29 command must be used. If you wish to enter new positions during a movement, e.g. to move along curved paths or to use your own path interpolators, then the "MOVE POS TIME" ([👉 2.2.5, Page 40](#)) command must be used.

2.2.5 MOVE POS TIME (0xB1)

COMMAND CODE: 0xB1

DESCRIPTION: The module moves to a specified position. The position is set in the configured *unit system* ([↗ 1.5, Page 22](#)). During the movement, new positions can be preset, which are then immediately moved to. When calculating the path, the target speed and acceleration values as well as the current actual speed and acceleration values are taken into account. If the time parameter is also given, then the speed and acceleration are adjusted such that the position is reached in the specified time while taking the specified speed and acceleration limits into account.

PARAMETER (Master -> Slave):

- *Position* ([↗ 2.2.16, Page 50](#)) (optional), in the configured unit system. The position must be offset by at least the *position variation* ([↗ 3.2.4, Page 132](#)) when compared with the start position.
- *Speed* ([↗ 2.2.11, Page 47](#)) (optional) which must not be exceeded.
- *Acceleration* ([↗ 2.2.12, Page 48](#)) (optional) which must not be exceeded.
- *Current* ([↗ 2.2.14, Page 49](#)) (optional) which must not be exceeded in the positioning movement. Should the controller structure be equal to "CURRENT SPEED" ([↗ 3.2.4, Page 132](#)), then this value cannot also be transferred ("INFO WRONG PARAMETER" ([↗ 2.8.2, Page 107](#))).
- *Time* ([↗ 2.2.15, Page 50](#)) (optional) in which the positioning movement is to be concluded in accordance with the speed and acceleration.

RESPONSE (Slave->Master): The time that the module needs for the movement is returned and the module executes the movement.

EXAMPLE 1:

	D-Len	Cmd	Param	
M->S	0x05	0xB1	0x00 0x00 0x20 0x41	Move to position 10.0[mm]
S->M	0x05	0xB1	0x00 0x00 0xA0 0x40	Will reach position in 5.0[sec]

EXAMPLE 2:

	D-Len	Cmd	Param	
M->S	0x01	0xB1		Move to the position which was set last
S->M	0x05	0xB1	0x00 0x00 0xA0 0x40	Will reach position in 5.0[sec]

MISCELLANEOUS: Spontaneous response when the position is reached "*CMD POS REACHED*" ([👉 2.3.3, Page 56](#)) or when the positioning movement is blocked "*MOVE BLOCKED*" ([👉 2.3.2, Page 55](#)). All parameters must be transferred in the specified order. If just the current is to be specified, then it is vital that the position, speed, and acceleration are also specified. The following parameters do not have to be transferred as well. All parameters are retained up to the restart or until these parameters are changed. The motion profile is temporarily converted to "*trapezoid*" ([👉 3.2.5, Page 135](#)) for this type of movement. (Only this profile is currently possible owing to computing time problems for curved paths)

NOTE

New position parameters can be entered during the movement. The new movement is subsequently calculated based on the entered parameters and the current actual speed and acceleration values. This makes it possible to move along curved paths.

NOTE

If the "jump" ([↗ 3.2.5, Page 135](#)) profile is switched on, then the internal ramp generator is switched off. This enables you to specify your own position ramps externally. Here, an adjustment to the controller parameters ([↗ 3.2.4, Page 132](#)) may be necessary depending on the interpolation interval of the external specification.

2.2.6 MOVE POS TIME REL (0xB9)

COMMAND CODE: 0xB9

DESCRIPTION: The module moves a specified distance. The change of position is set in the configured *unit system* ([↗ 1.5, Page 22](#)). During the movement, new positions can be preset, which are then immediately moved to. When calculating the path, the target speed and acceleration values as well as the current actual speed and acceleration values are taken into account. If the time parameter is also given, then the speed and acceleration are adjusted such that the position is reached in the specified time while taking the specified speed and acceleration limits into account.

PARAMETER (Master -> Slave):

- *Change of position* ([↗ 2.2.17, Page 51](#)) (optional), in the configured unit system. The change of position must not be smaller than the *position variation* ([↗ 3.2.4, Page 132](#)).
- *Speed* ([↗ 2.2.11, Page 47](#)) (optional) which must not be exceeded.
- *Acceleration* ([↗ 2.2.12, Page 48](#)) (optional) which must not be exceeded.
- *Current* ([↗ 2.2.14, Page 49](#)) (optional) which must not be exceeded in the positioning movement. Should the controller structure be equal to "CURRENT SPEED" ([↗ 3.2.4, Page 132](#)), then this value cannot also be transferred ("INFO WRONG PARAMETER" ([↗ 2.8.2, Page 107](#))).
- *Time* ([↗ 2.2.15, Page 50](#)) (optional) in which the positioning movement is to be concluded in accordance with the parameterized speed and acceleration limits.

RESPONSE (Slave->Master): The time that the module needs for the movement is returned. The module executes the command.

EXAMPLE 1:

	D-Len	Cmd	Param	
M->S	0x05	0xB9	0x00 0x00 0x20 0x41	Move further by 10.0 [mm]
S->M	0x05	0xB9	0x00 0x00 0xA0 0x40	Will expectedly need 5.0 [s] for this

EXAMPLE 2:

	D-Len	Cmd	Param	
M->S	0x01	0xB9		Continue to move by the distance set last
S->M	0x05	0xB9	0x00 0x00 0xA0 0x40	Will expectedly need 5.0[sec] for this

MISCELLANEOUS: Spontaneous response when the position is reached "CMD POS REACHED" ([👉 2.3.3, Page 56](#)). All parameters must be transferred in the specified order. If just the current is to be specified, then it is vital that the change of position, speed, and acceleration are also specified. The following parameters do not have to be transferred as well. All parameters are retained up to the restart or until these parameters are changed. The motion profile is temporarily changed to "trapezoid" ([👉 3.2.5, Page 135](#)) for this type of movement. (Only this profile is currently possible owing to computing time problems for curved paths)

NOTE

New change of position parameters can be entered during the movement. The new movement is subsequently calculated based on the entered parameters and the current actual speed and acceleration values. This makes it possible to move along curved paths.

NOTE

If the "jump" ([👉 3.2.5, Page 135](#)) profile is switched on, then the internal ramp generator is switched off. This enables you to specify your own position ramps externally. Here, an adjustment to the controller parameters ([👉 3.2.4, Page 132](#)) may be necessary depending on the interpolation interval of the external specification.

2.2.7 MOVE POS LOOP (0xBA)

COMMAND CODE: 0xBA

DESCRIPTION: Variant of the *MOVE POS* ([2.2.3, Page 36](#)) command. The module executes a cyclical movement between the start position (current position) and the defined position.

2.2.8 MOVE CUR (0xB3)

COMMAND CODE: 0xB3

DESCRIPTION: A current movement is executed.

PARAMETER (Master -> Slave):

- *Current* ([2.2.14, Page 49](#)) in the configured *unit system* ([1.5, Page 22](#)).

RESPONSE (Slave->Master): "OK" (0x4F4B) if successful. The module executes the command.

EXAMPLE 1:

	D-Len	Cmd	Param	
M->S	0x05	0xB3	0x00 0x00 0x60 0x40	Execute current movement with 3.5[A]
S->M	0x03	0xB3	0x4F 0x4B	

EXAMPLE 2:

	D-Len	Cmd	Param	
M->S	0x01	0xB3		Execute the current movement with the current set last
S->M	0x03	0xB3	0x4F 0x4B	

MISCELLANEOUS: Spontaneous message ("CMD MOVE BLOCKED" ([2.3.2, Page 55](#))) is possible.

NOTE

Owing to the controller structures used, the module can "overspeed". If the module exceeds the configured maximum speed ([↗ 3.2.1, Page 117](#)), this will result in an "ERROR TOW" ([↗ 2.8.2, Page 107](#)) for reasons of safety.

NOTE

When the controller structure ([↗ 3.2.4, Page 132](#)) "CASCADE" is set, then this command will not be required. All motion types work with subordinate current control that can be adjusted ([↗ 2.2.14, Page 49](#)).

2.2.9 MOVE VEL (0xB5)

COMMAND CODE: 0xB5

DESCRIPTION: A speed movement is executed.

PARAMETER (Master -> Slave):

- *Speed* ([↗ 2.2.11, Page 47](#)) in the configured *unit system* ([↗ 1.5, Page 22](#)).
- *Current* ([↗ 2.2.14, Page 49](#)) (optional) which must not be exceeded in the speed movement. Generates "INFO WRONG PARAMETER" ([↗ 2.8.2, Page 107](#)) in controller structure "CURRENT SPEED" ([↗ 3.2.4, Page 132](#)).

RESPONSE (Slave->Master): "OK" (0x4F4B) if successful. The module executes the command.

EXAMPLE 1:

	D-Len	Cmd	Param	
M->S	0x05	0xB5	0x9A 0x99 0x31 0x41	Execute speed movement with 11.1[(mm)/(s)]
S->M	0x03	0xB5	0x4F 0x4B	

EXAMPLE 2:

	D-Len	Cmd	Param	
M->S	0x05	0xB5	0x9A 0x99 0x31 0x41	Execute speed movement with speed set last
S->M	0x03	0xB5	0x4F 0x4B	

MISCELLANEOUS: Miscellaneous: Spontaneous message "*CMD MOVE BLOCKED*" ([↗ 2.3.2, Page 55](#)) is possible if the module was blocked during the movement.

2.2.10 MOVE GRIP (0xB7)

COMMAND CODE : 0xB7

Description: A "gripping movement" is executed.

PARAMETER (Master -> Slave):

- *Current* ([↗ 2.2.14, Page 49](#)) in the configured *unit system* ([↗ 1.5, Page 22](#)).
- *Max. speed* ([↗ 2.2.11, Page 47](#)) (optional) which must not be exceeded in the configured *unit system* ([↗ 1.5, Page 22](#)) during the gripping movement.

RESPONSE (Slave->Master): "OK" (0x4F4B) if successful. The module executes the command.

EXAMPLE 1:

	D-Len	Cmd	Param	
M->S	0x05	0xB7	0x00 0x00 0x60 0x40	Execute "gripping movement" with 3.5 [A]
S->M	0x03	0xB7	0x4F 0x4B	

EXAMPLE 2:

	D-Len	Cmd	Param	
M->S	0x01	0xB7		Execute the "gripping movement" with the current set last
S->M	0x03	0xB7	0x4F 0x4B	

MISCELLANEOUS: Spontaneous message ("CMD MOVE BLOCKED" [\(↗ 2.3.2, Page 55\)](#) is possible if the movement was blocked (an object was gripped).

NOTE

The command only works correctly if subordinate current control [\(↗ 3.2.4, Page 132\)](#) is available.

NOTE

Gripping is also possible via a speed movement [\(↗ 2.2.9, Page 45\)](#) or positioning movement [\(↗ 2.2.3, Page 36\)](#) with subordinate current control [\(↗ 3.2.4, Page 132\)](#).

2.2.11 SET TARGET VEL (0xA0)

COMMAND CODE: 0xA0

DESCRIPTION: This is used to set the velocity parameter.

PARAMETER (Master -> Slave):

- *Speed* in the specified *unit system* [\(↗ 1.5, Page 22\)](#).

RESPONSE (Slave->Master): "OK" (0x4F4B) if successful

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x05	0xA0	0x33 0x33 0x43 0x41	Set speed to 12.2 [mm/s]
S->M	0x03	0xA0	0x4F 0x4B	

MISCELLANEOUS: This value is retained after it has been set successfully once until the module is restarted or this value is changed again. The value is not adopted until the next motion command and is not immediately active.

2.2.12 SET TARGET ACC (0xA1)

COMMAND CODE: 0xA1

DESCRIPTION: This is used to set the acceleration parameter.

PARAMETER (Master -> Slave):

- *Acceleration* in the specified *unit system* ([👉 1.5, Page 22](#))

RESPONSE (Slave->Master): "OK" (0x4F4B) if successful

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x05	0xA1	0x00 0x00 0xF0 0xA2	Set the acceleration [mm/s ²]
S->M	0x03	0xA1	0x4F 0x4B	

MISCELLANEOUS: This value is retained after it has been set successfully once until the module is restarted or this value is changed again. The value is not adopted until the next motion command and is not immediately active.

2.2.13 SET TARGET JERK (0xA2)

COMMAND CODE: 0xA2

DESCRIPTION: This is used to set the jerk parameter.

PARAMETER (Master -> Slave): *Jerk* in the specified *unit system* ([👉 1.5, Page 22](#))

RESPONSE (Slave->Master): "OK" (0x4F4B) if successful

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x05	0xA2	0x00 0x00 0x7A 0x44	Set the jerk [mm/s ³]
S->M	0x03	0xA2	0x4F 0x4B	

MISCELLANEOUS: This value is retained after it has been set successfully once until the module is restarted or this value is

changed again. The value is not adopted until the next motion command and is not immediately active.

2.2.14 SET TARGET CUR (0xA3)

COMMAND CODE: 0xA3

DESCRIPTION: This is used to set the current parameter.

PARAMETER (Master -> Slave): *Current* in the specified *unit system* ([👉 1.5, Page 22](#))

RESPONSE (Slave->Master): "OK" (0x4F4B) if successful

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x05	0xA3	0xCD 0xCC 0x2C 0x40	Set the current to 2.7 [A]
S->M	0x03	0xA3	0x4F 0x4B	

MISCELLANEOUS: This value is retained after it has been set successfully once until the module is restarted or this value is changed again. The value is adopted immediately and is also immediately active.

NOTE

This can be used to change the target specifications for the current during the movement. Thus it is possible with grippers to "re-grip" an already gripped object (increase or reduce gripping force).

NOTE

Only considered for movements if the controller structure allows it. Controller structure ([👉 3.2.4, Page 132](#)) "CURRENT SPEED" does not allow subordinate current control.

2.2.15 SET TARGET TIME (0xA4)

COMMAND CODE: 0xA4

DESCRIPTION: The "time" parameter for the next command "MOVE POS TIME" ([2.2.5, Page 40](#)) or "MOVE POS TIME REL" is set with this.

PARAMETER (Master -> Slave):

- *Time* in the specified *unit system* ([1.5, Page 22](#))

RESPONSE (Slave->Master): "OK" (0x4F4B) if successful

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x05	0xA4	0x66 0x66 0x96 0x40	Set the time to 4.7 [s]
S->M	0x03	0xA4	0x4F 0x4B	

MISCELLANEOUS: Only applies to the next "MOVE POS TIME" ([2.2.5, Page 40](#)), "MOVE POS TIME REL" ([2.2.6, Page 42](#)) command. This value will be deleted again after one of the previously named commands is sent.

2.2.16 SET TARGET POS (0xA6)

COMMAND CODE: 0xA6

DESCRIPTION: The "position" parameter for the next "MOVE POS" ([2.2.3, Page 36](#)) or „MOVE POS TIME" ([2.2.5, Page 40](#)) command is set with this.

PARAMETER (Master -> Slave):

- *Position* in the specified *unit system* ([1.5, Page 22](#))

RESPONSE (Slave->Master): "OK" (0x4F4B) if successful

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x05	0xA6	0x00 0x00 0xc8 0x43	Set the position to 400.0 [°]
S->M	0x03	0xA6	0x4F 0x4B	„OK“

MISCELLANEOUS: This value is retained after it has been set successfully once until the module is restarted or this value is changed again. The value is not adopted until the next motion command and is not immediately active.

2.2.17 SET TARGET POS REL (0xA7)

COMMAND CODE: 0xA7

DESCRIPTION: The "relative position" parameter for the next "MOVE POS REL" ([2.2.4, Page 38](#)) or "MOVE POS TIME REL" ([2.2.6, Page 42](#)) command is set with this.

PARAMETER (Master -> Slave):

- *relative position* in the specified *unit system* ([1.5, Page 22](#))

RESPONSE (Slave->Master): "OK" (0x4F4B) if successful

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x05	0xA7	0x00 0x00 0xc8 0x43	Set the relative position to 400.0 [°]
S->M	0x03	0xA7	0x4F 0x4B	„OK“

MISCELLANEOUS: This value is retained after it has been set successfully once until the module is restarted or this value is changed again. The value is not adopted until the next motion command and is not immediately active.

2.2.18 CMD STOP (0x91)

COMMAND CODE: 0x91

DESCRIPTION: The module is braked and stopped in the current position. In modules with an appropriately *configured holding brake* ([↗ 3.2.7, Page 150](#)), the brake will be activated, otherwise, the module will be controlled actively.

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): "OK" (0x4F4B) if successful

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0x91		
S->M	0x03	0x91	0x4F 0x4B	

MISCELLANEOUS: If the command cannot be executed correctly, then it is answered with an *error message* ([↗ 2.8.2, Page 107](#)). For example, the module is to remain in active control, but cannot do this because a serious error is present.

NOTE

In modules without a brake or appropriately configured holding brake ([↗ 3.2.7, Page 150](#)), the maximum permissible current for control is limited to the nominal current ([↗ 3.2.1, Page 117](#)) to avoid overheating of the motor. This is why the module may "stall".

2.2.19 CMD FAST STOP (0x90)

COMMAND CODE: 0x90

DESCRIPTION: If a brake is fitted and appropriately *configured* ([↗ 3.2.7, Page 150](#)), it is activated immediately and the motor phases are short circuited. The motor is "de-energized".

PARAMETER (Master -> Slave): None

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0x90		
S->M	0x02	0x88	0xD9	Fast stop executed

RESPONSE (Slave->Master): Error message "ERROR FAST STOP" ([↗ 2.8.2, Page 107](#)) is triggered.

MISCELLANEOUS: Can only be reset by "CMD ACK" ([↗ 2.8.1, Page 105](#)).



⚠ WARNING

Risk of injury

In modules without a brake, the module may "stall", since the motor is de-energized during a fast stop.

NOTE

This type of stopping leads to heavy mechanical wear on the brake.

2.3 Spontaneous messages

The module can report independently for certain events. A so-called "spontaneous message" is triggered. These messages are sent via the standard *data frame* ([☞ 1.4.2, Page 14](#)). (D-Len, CmdCode, parameters). Spontaneous messages can be *deactivated* ([☞ 2.3.7, Page 58](#)).

NOTE

In Profibus, the MsgCount is not increased for these types of messages, since no data has been requested by the control unit.

2.3.1 CMD INFO (0x8A)

COMMAND CODE: 0x8A

EXAMPLE:

	D-Len	Cmd	Param	
S->M	0x02	0x8A	0x10	"INFO TIMEOUT" (☞ 2.8.2, Page 107)

DESCRIPTION: The module sends an information message.

REMARK: "Info messages" are also sent when errors are rectified and the module is restarted.

("INFO BOOT" ([☞ 2.8.2, Page 107](#)), "INFO NO ERROR" ([☞ 2.8.2, Page 107](#)))

2.3.2 CMD MOVE BLOCKED (0x93)

COMMAND CODE: 0x93

DESCRIPTION: The current motion command was interrupted. The motor was briefly blocked or is still blocked.

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Current *position* in the set *unit system* ([↗ 1.5, Page 22](#))

MISCELLANEOUS: The motor is considered blocked if all the following conditions are met:

- The motor turns at a speed below the *motion threshold* ([↗ 3.2.6, Page 146](#))
- The target current is reached (+/-15%)
- The *brake timeout* ([↗ 3.2.7, Page 150](#)) has expired.

Also see the *flag "Motion blocked"* ([↗ 2.6.1, Page 84](#)).

NOTE

When the blocking is detected, the set current will be reduced by $\sqrt{2}$ in sine commutation. (Form factor for effective value to peak value in sinusoidal currents).

NOTE

Absolute reliable detection as to whether an object has been gripped is not possible using this.

2.3.3 CMD POS REACHED (0x94)

COMMAND CODE: 0x94

DESCRIPTION: A positioning movement has reached the target position.

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): *Position* (current) in the set *unit system* ([↗ 1.5, Page 22](#))

EXAMPLE:

	D-Len	Cmd	Param	
S->M	0x05	0x94	0xCD 0xCC 0x2C 0x40	Has reached position 2.7 [mm].

MISCELLANEOUS: The module has stopped. If a brake is fitted, it will activate according to the *configuration* ([↗ 3.2.7, Page 150](#))
ein.

2.3.4 CMD ERROR (0x88)

COMMAND CODE: 0x88

DESCRIPTION: A serious error has occurred which makes user intervention necessary. These errors must be acknowledged with "CMD ACK" ([↗ 2.8.1, Page 105](#)). The module is not ready for operation. The motor is de-energized, and the brake is activated. These error messages are sent regularly every 15 seconds from the module to the control unit until the error is acknowledged.

- Serial communication: The first byte in a message changes to "0x03"
- CAN: The first three bits in the identifier result in "0x3"
- Profibus: An extended diagnosis is generated

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): *Error code* ([↗ 2.8.2, Page 107](#))

EXAMPLE:

	D-Len	Cmd	Param	
S->M	0x02	0x88	0xDE	The error " <i>ERROR CURRENT</i> " (👉 2.8.2, Page 107) has occurred.

**WARNING**

Risk of injury! In modules without a brake, the module may "stall", since the motor is de-energized in the event of a serious error.

2.3.5 GET STATE (0x95)

COMMAND CODE: 0x95

DESCRIPTION: Provides the status of the module and some further information, if required. The module can independently send this status at regular intervals.

For details, see "*GET STATE*" ([👉 2.6.1, Page 84](#)).

2.3.6 GET STATE AXIS (0x98)

COMMAND CODE: 0x98

DESCRIPTION: Provides the status of an axis or a robot in the module and some further information, if required. The module can independently send this status at regular intervals. For details, see "*GET STATE AXIS*" ([👉 2.6.2, Page 87](#)).

2.3.7 CMD TOGGLE IMPULSE MESSAGE (0xE7)

COMMAND CODE: 0xE7

DESCRIPTION: This can be used to activate or deactivate spontaneous messages.

If the "secondary interface" ([1.6.2, Page 24](#)) is active, then this is used to activate or deactivate the "listening mode".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): If the command is confirmed with "ON" (0x4F 0x4E), then spontaneous messages are active. If the command is confirmed with "OFF" (0x4F 0x46 0x46), then spontaneous messages are deactivated.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xE7		
S->M	0x04	0xE7	0x4F 0x46 0x46	Spontaneous message deactivated.

MISCELLANEOUS: Spontaneous messages are always activated in the restart of the module.

(Exception: *Serial communication without spontaneous message* ([3.2.5, Page 135](#)))

NOTE

Spontaneous messages should not be deactivated, if possible. This may possibly be necessary for serial communication if a great many modules are controlled and there are frequent collisions. It is possible to automatically deactivate spontaneous messages after a restart ([3.2.5, Page 135](#)), particularly during serial communication. Spontaneous messages can then be activated at any time using CMD TOGGLE IMPULSE MESSAGE.

2.4 Settings

2.4.1 SET CONFIG (0x81)

COMMAND CODE: 0x81

DESCRIPTION: Configuration parameters in the module are set and *saved permanently* ([↗ 3.2, Page 117](#)).

PARAMETER (Master -> Slave): Transmitted as the command parameter is which configuration parameters are to be written.

- EEPROM (0xFE) + EEPROM structure

All configuration parameters are written all at once (the complete EEPROM structure must be transferred in the data).

Depending on the *user* ([↗ 1.6.1, Page 24](#)), it could be that some data are not written. After completion of the writing process, the module will reboot.

NOTE

This command should not be used in your own applications, since the structure for the data to be sent is not known.

NOTE

The module has to be stopped for this, and the control has to be deactivated. For example, execute fast stop ([↗ 2.2.19, Page 53](#)) beforehand.

NOTE

To do this, the fragmentation protocol ([↗ 1.4.6, Page 20](#)) must be used (except for serial communication).

- Module ID (0x01) + data (1 byte)

The ID of the module is changed. Valid values (0 .. 255). The new settings are immediately saved in EEPROM but not activated until a restart has been carried out.

NOTE

Profibus supports "Set slave address" (SAP 55)

- Group ID (0x02) + data (1 byte)

The group of the module is changed. Valid values (0 .. 255). The new settings are immediately saved in EEPROM but not activated until a restart has been carried out.

NOTE

In Profibus, "Real no add change" is saved here. Groups are fully supported by the SYNC / FREEZE mechanism.

- Baudrate for serial communication (0x03) + data (2 bytes)
The baudrate for serial communication can be changed. Valid values (1200, 2400, 4800, 9600, 19200, 38400). The new settings are immediately saved in EEPROM but not activated until a restart has been carried out.
- CAN baudrate (0x04) + data (2 bytes)
The CAN baudrate can be set. Valid values (50, 100, 125, 250, 500, 800, 1000). The new settings are immediately saved in EEPROM but not activated until a restart has been carried out.
- Communication (0x05) + data (1 byte)
The communication interface is set. Valid values: AUTO (0x00), serial (0x01), CAN (0x02), Profibus DPV0 (0x03), serial without spontaneous message (0x04). The new settings are immediately saved in EEPROM but not activated until a restart has been carried out.
- Unit system (0x06) + data (1 byte)
The unit system can be converted. Valid values ([mm] = 0x00, [m] = 0x01, [inches] = 0x02, [rad] = 0x03, [degrees] = 0x04, [internal] = 0x05, [μm] integer = 0x06, [μdegrees] integer = 0x07, [μinches] integer = 0x08, [millidegrees] integer = 0x09). The new settings are immediately saved in EEPROM but not activated until a restart has been carried out.
- Max. software limit range (0x07) + data (4 bytes)
The upper software limit range is changed temporarily (observe *unit system* ([1.5, Page 22](#))). The transmitted value is not written in EEPROM. The setting is immediately active.
(This is required for SRU in Teach mode. Enabled in SRU for "USER").

NOTE

This function is enabled as from the "Profi" user ([1.6.3, Page 26](#)).

- Min. software limit range (0x08) + data (4 bytes)
The lower software limit range is changed temporarily (observe *unit system* ([1.5, Page 22](#))). The transmitted value is not written in EEPROM. The setting is immediately active.
(This is required for SRU in Teach mode. Enabled in SRU for "USER").

NOTE

This function is enabled as from the "Profi" user ([1.6.3, Page 26](#))

- Gear ratio (0x18) + data (4 bytes as float)
Gear ratio 1 ([3.2.2, Page 123](#)) is changed (the command does not apply to the *integer unit systems* ([1.5, Page 22](#))).
 The transmitted value is written in EEPROM and is immediately active.

NOTE

This function is enabled as from the "Profi" user ([1.6.3, Page 26](#)).

RESPONSE (Slave->Master): "OK" (0x4F4B) if successful. To be able to recognize which parameter has been set successfully, the parameter code is attached subsequent to "OK" (1 byte).

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x03	0x81	0x01 0x0C	Set the module ID to 12
S->M	0x04	0x81	0x4F 0x4B 0x01	

MISCELLANEOUS: The supplied *software tool* ([3.2.7, Page 150](#)) can be used for making it easier to set the configuration parameters. If all the parameters are to be written all at once, this is only possible when the module is at standstill.

2.4.2 GET CONFIG (0x80)

COMMAND CODE: 0x80

DESCRIPTION: Various configuration parameters can be read out from the module.

PARAMETER (Master -> Slave):

- None
 The following information is returned:
 - Module type in plain text (8 ASCII characters)
 - Order number (UInt32)
 - Firmware version (UInt16)
 - Protocol version (UInt16)
 - Hardware version (UInt16)

- Firmware creation date and time (21 ASCII characters)

NOTE

Modules with an even-numbered hardware version have EEPROM installed. Modules with an odd-numbered hardware version have FRAM installed. See also "Pseudo-absolute encoder" ([☞ 1.7, Page 27](#))

- EEPROM (0xFE)

All configuration parameters are read out at once. (*PC parameterization software* ([☞ 3.2.7, Page 150](#)) uses this subcode)

NOTE

The module has to be stopped for this and the control deactivated (for example, fast stop beforehand ([☞ 2.2.19, Page 53](#))).

NOTE

This command should not be used in your own applications, since the structure for the received data is not known.

NOTE

To do this, the fragmentation protocol ([☞ 1.4.6, Page 20](#)) must be used (except for serial communication).

- Module ID (0x01)
The ID of the module is read (1 byte).
- Group ID (0x02)
The module's group assignment is read (1 byte). (currently, this is only relevant to the *Profibus* ([☞ 1.4.5, Page 18](#)) Set slave address)
- Serial baudrate (0x03)
The baudrate for serial communication is read (2 bytes).
- CAN baudrate (0x04)
The CAN baudrate is read (2 bytes).
- Communication (0x05)
The configured communication interface is read out (1 byte).
Valid values: AUTO (0x00), serial (0x01), CAN (0x02), Profibus DPV0 (0x03), serial communication without spontaneous message (0x04).
- Unit system (0x06)
The configured unit system is read out. Possible values ([mm] = 0x00, [m] = 0x01, [inches] = 0x02, [rad] = 0x03, [degrees] = 0x04, [internal] = 0x05, [μm] integer = 0x06, [μdegrees] integer = 0x07, [μinches] integer = 0x08, [millidegrees] integer = 0x09).

- Max. software limit range (0x07)
Current max. software limit range is returned (4 bytes).
Observe *unit system* ([☞ 1.5, Page 22](#)).
- Min. software limit range (0x08)
Current min. software limit range is returned (4 bytes). Observe *unit system* ([☞ 1.5, Page 22](#)).
- Max. speed (0x09)
Max. permissible speed for module is returned (4 bytes).
Observe *unit system* ([☞ 1.5, Page 22](#)).
- Max. acceleration (0x0A)
Max. permissible acceleration for module is returned (4 bytes).
Observe *unit system* ([☞ 1.5, Page 22](#)).
- Max. current (0x0B)
Max. permissible current for module is returned (4 bytes).
Observe *unit system* ([☞ 1.5, Page 22](#)).
- Nom. current (0x0C)
Nom. current for module is returned (4 bytes). Observe *unit system* ([☞ 1.5, Page 22](#)).
- Max. jerk (0x0D)
Max. permissible jerk for module is returned (4 bytes). Observe *unit system* ([☞ 1.5, Page 22](#)).
- Offset phase A (0x0E)
The offset for current sensor A is returned (2 bytes).
- Offset phase B (0x0F)
The offset for current sensor B is returned (2 bytes).
- CRC data (0x13)
The CRC on EEPROM entries without individual data (serial number, sensor-specific offsets (current, resolver)) for a module is transmitted.
- Reference offset (0x14)
The reference offset is returned (4 bytes). Observe *unit system* ([☞ 1.5, Page 22](#)).
- Serial number (0x15)
The device's serial number is returned (4 bytes).
- Order number (0x16)
The device's order number is returned (4 bytes).
- Error History (0xFD)
The last 20 errors will be read out of the ring memory.

RESPONSE (Slave->Master): Parameter ID (to keep track on which data have been requested). Following this, the requested data.

EXAMPLE 1:

	D-Len	Cmd	Param	
M->S	0x02	0x80	0x06	Read the unit system
S->M	0x03	0x80	0x06 0x00	The unit system is [mm]

EXAMPLE 2:

	D-Len	Cmd	Param	
M->S	0x01	0x80		Read the module information
S->M	0x28	0x80	0x50 0x52 0x2D 0x37 0x30 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x79 0x00 0x03 0x00 0x12 0x02 0x31 0x31 0x3A 0x32 0x32 0x3A 0x32 0x37 0x20 0x20 0x4A 0x75 0x6C 0x20 0x20 0x33 0x20 0x32 0x30 0x30 0x38	Module type "PR-70", order number 0, firmware version 1.21, protocol version 3, hardware version 5.30, firmware created "11:22:27 Jul 3 2008"

MISCELLANEOUS: The supplied *software tool* ([🔗 3.2.7, Page 150](#)) can be used for making it easier to read the configuration parameters. If all the parameters are to be read all at once, this is only possible when the module is at standstill.

2.4.3 SET CONFIG EXT (0x83)

COMMAND CODE: 0x83

DESCRIPTION: Individual configuration parameters in the module are set and *saved permanently* ([🔗 3.2, Page 117](#)). The "SET

CONFIG EXT" command is a generalization of *SET CONFIG* ([🔗 2.4.1, Page 59](#)) and differs from this as follows:

- Only one, single configuration parameter can be set specifically per command.
- The configuration parameters are identified via a 16 bits wide configuration code.
- The significance of a configuration code depends on the device connected. For more on this, see the relevant supplementary instructions.
- The value of a configuration parameter to be set can be transferred in various data formats, depending on the device (see definition of "data type" below).

PARAMETER (Master -> Slave):

- Configuration code (2 bytes)
- Data type (1 byte)
- Configuration parameter (n bytes)
- Axis index (1 byte) (optional)

The "Configuration code" identifies the configuration parameter to be set. The code is device-specific. A description of the valid configuration code can be found in the respective supplementary instructions. The optional "Axis index" identifies the index of the axis or robot for which the configuration parameters are to be set. If this is not specified, then a standard setting is used. The "Data type" is defined as follows:

DATA TYPES:

Name	Value	Description
DT_UNKNO WN	0	Unknown data type, must not be used
DT_UINT8	1	Unsigned integer with 8 bits (1 byte)
DT_INT8	2	Signed integer with 8 bits (1 byte)
DT_UINT16	3	Unsigned integer with 16 bits (2 bytes)
DT_INT16	4	Signed integer with 16 bits (2 bytes)
DT_UINT32	5	Unsigned integer with 32 bits (4 bytes)
DT_INT32	6	Signed integer with 32 bits (4 bytes)
DT_UINT64	7	Unsigned integer with 64 bits (8 bytes)
DT_INT64	8	Signed integer with 64 bits (8 bytes)

Name	Value	Description
DT_FLOAT	9	Floating point number with single precision (4 bytes)
DT_DOUBLE	10	Floating point number with double precision (8 bytes)
DT_STRING	11	"String length" + ASCII string, see below (n bytes)

Dieser zusätzliche Parameter „Stringlänge“ wird benötigt um entscheiden zu können, ob das letzte Byte einer SMP-Nachricht den optionalen Achsindex darstellt oder noch Teil des Strings ist.

RESPONSE (Slave->Master): If successful,

- "OK" (0x4F4B)
- *Configuration code* (2 bytes)
- *Axis index* (1 byte) (optional)

The returned values of "Configuration code" and, where necessary, "Axis index" match the requested values and thus allow the assignment of the response to the request. If the request contains an "Axis index", then the response is also sent with "Axis index"

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x08	0x83	0x12 0x07 0x0B 0x04 0x54 0x65 0x73 0x74	Set configuration code 1810 (0x12 0x07) with data type DT_STRING (0x0b) and string length 4 (0x04) to the "Test" (0x54 0x65 0x73 0x74) value
S->M	0x05	0x83	0x4F 0x4B 0x12 0x07	Success (0x4f 0x4b) when setting configuration code 1810 (0x12 0x07)

MISCELLANEOUS: The supplied *software tool* ([🔗 3.2.7, Page 150](#)) can be used for making it easier to set the configuration parameters.

NOTE

Some configuration parameters can only be changed if the module has stopped and the control is deactivated. Therefore, trigger fast stop ([☞ 2.2.19, Page 53](#)) beforehand, if necessary. In the event of an error, a response is made with "INFO NO RIGHTS" ([☞ 2.8.2, Page 107](#)).

NOTE

Some configuration parameters cannot be changed. If "SET CONFIG EXT" is used to access such "Read-only" data, then a response is made with error code "INFO NO RIGHTS" ([☞ 2.8.2, Page 107](#)) geantwortet.

2.4.4 GET CONFIG EXT (0x82)

COMMAND CODE: 0x82

DESCRIPTION: Certain configuration parameters can be read out from the module. The "GET CONFIG EXT" command is a generalization of *GET CONFIG* ([👉 2.4.2, Page 61](#)) and differs from this as follows:

- Only one, single configuration parameter can be queried specifically per command.
- The configuration parameters are identified via a 16 bits wide configuration code.
- The significance of a configuration code depends on the device connected. For more on this, see the respective supplementary instructions.
- The value of a returned configuration parameter can be transferred in various *data formats* ([👉 2.4.3, Page 64](#)), depending on the device.

PARAMETER (Master -> Slave):

- Configuration code (2 bytes)
- Axis index (1 byte) (optional)

The "Configuration code" identifies the configuration parameter requested. The code is device-specific. A description of the valid configuration code can be found in the respective device description. The "Axis index" identifies the index of the axis or robot from which the configuration parameters are to be read.

RESPONSE (Slave->Master):

- Configuration code (2 bytes)
- Data type (1 byte)
- Configuration parameter (n bytes)
- Axis index (1 byte) (optional)

The returned values of "Configuration code" and, where necessary, "Robot index" match the requested values and thus allow the assignment of the response to the request. The "Data type" is defined according to *SET CONFIG EXT* ([👉 2.4.3, Page 64](#)). The n amount for bytes in the configuration parameter depends on the "Data type". With the "DT_STRING" data type, the number of the following ASCII characters is contained in the first byte of the configuration parameter. 1

This additional "String length" parameter is required for deciding whether the last byte in an SMP message represents the optional axis index or is still a part of the string. If the request contains an "Axis index", then the response is also sent with "Axis index".

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x03	0x82	0x72 0x00 0x00	Read the config. parameter of configuration code=114 (0x072 0x00)
S->M	0x17	0x82	0x72 0x00 0x0B 0x13 0x37 0x32 0x30 0x31 0x2D 0x30 0x30 0x37 0x36 0x2D 0x34 0x35 0x36 0x34 0x2D 0x33 0x34 0x62 0x35 0x00	Read config. parameter: Configuration code=114 (0x072 0x00); data type=DT_STRING (0x0B); string length=19 (0x13); data="7201-0076-4564-34b5" (0x37 0x32, ..., 0x35);

MISCELLANEOUS: The supplied *software tool* ([🔗 3.2.7, Page 150](#)) can be used for making it easier to read the configuration parameters.

2.4.5 CMD_DEFAULT (0x8C)

COMMAND CODE: 0x8C

DESCRIPTION: The module is set to factory settings.

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): The command has to be confirmed with "OK" (0x4F4B). The factory settings are activated after reboot.

2.5 Commands for internal programming

The modules can be operated using internal sequential programs *without external control* ([☞ 3.2.5, Page 135](#)). There is also the option of allowing previously stored programs to run via the digital inputs or outputs and thus operating the module only via digital I/Os. To do this, the *inputs* ([☞ 3.2.5, Page 135](#)) or *outputs* ([☞ 3.2.5, Page 135](#)) have to be configured accordingly.

Furthermore, the module can be operated with 2 bytes by starting previously programmed, more complex sequences via *special commands* ([☞ 2.5.5, Page 73](#)).

A *fragmentation* ([☞ 1.4.6, Page 20](#)) of the data is required to be able to program the modules. The easiest way to program the module is by using the supplied *software tool* ([☞ 3.2.7, Page 150](#)) via the serial interface.

Programs can only be written "at one stretch". If you need to change a section, then the entire program must be read out, the relevant correction must be performed, and the entire program must be written in the module again.

Limited jumps, loops, and branchings are not possible.

(The program data are densely packed in EEPROM if possible and equipped with checksums, which prevents a change in the middle of the complex data structure)

2.5.1 SET PHRASE (0xC0)

COMMAND CODE: 0xC0

DESCRIPTION: If this command is put in front of any type of data frame, then that data frame will be stored in the non-volatile memory. This allows all available commands to be used for programming. A stored data frame is also known as a "phrase" in this mode. The phrases are automatically increased until the SET PHRASE command is called up without parameters (data frames). This is the indicator for "Programming completed".

PARAMETER (Master -> Slave):

- None
Programming completed.
- Valid data frame
The phrase is saved and the phrase counter increased.

RESPONSE (Slave->Master): "OK" (0x4F4B) if the phrase was saved successfully.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x07	0xC0	0x05 0xB1 0x00 0x00 0x60 0x40	Program move to position 3.5 [mm]
S->M	0x03	0xC0	0x06 0x00	

MISCELLANEOUS: The easiest way to carry out the programming is by using the supplied *software* ([👉 3.2.7, Page 150](#)).

NOTE

The fragmentation protocol ([👉 1.4.6, Page 20](#)) must be used for the programming (except for serial communication).

2.5.2 GET PHRASES (0xC2)

COMMAND CODE: 0xC2

DESCRIPTION: Reads out the complete program from the module

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): The first data frame contains the number of phrases in the program (UInt16). Then the contents of the program will be transmitted phrase for phrase.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xC2		
S->M	0x03	0xC2	0x05 0x00	The program contains 5 phrases
S->M	0x05	0xB1	0x00 0x00 0x60 0x40	Phrase 0 contains one "move to position 3.5 [mm]" command
S->M	0x05	0xB1	0x00 .. .	Phrases to end of program memory

MISCELLANEOUS: Line for line of the program will be transmitted as soon as the command has been initiated once. The easiest way to manage programs is by using the supplied *software* ([☞ 3.2.7, Page 150](#)).

NOTE

To do this, the fragmentation protocol ([☞ 1.4.6, Page 20](#)) must be used (except for serial communication).

2.5.3 PRG EXE (0xCF)

COMMAND CODE: 0xCF

DESCRIPTION: A program is executed

PARAMETER (Master -> Slave): None. The program execution begins at line "0"

RESPONSE (Slave->Master): Program line (UInt16) with the command code included is sent and the corresponding phrase executed.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xCF		Start the program at line "0"
S->M	0x01	0xCF		Confirmation from module
S->M	0x04	0xC1	0x00 0x00 0xB0	Line "0" contains a "MOVE POS" command
S->M	0x04	0xC1	0x01 .. .	Phrases to end of program

2.5.4 EXE PHRASE (0xC1)

COMMAND CODE: 0xC1

DESCRIPTION: Line from the saved program is executed.

PARAMETER (Master -> Slave): *Program line* (UInt16) that is to be executed.

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x03	0xC1	0x02 0x00	Execute phrase "2".
S->M	0x04	0xC1	0x02 0x00 0xB0	Phrase 2 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in the specified program line, then the complete subprogram will be automatically executed.

2.5.5 EXE PHRASE0 (0xD0)

COMMAND CODE: 0xD0

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "0".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xD0		Execute phrase "0".
S->M	0x04	0xD0	0x00 0x00 0xB0	Phrase 0 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 0, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte. This command together with the corresponding phrase number is generated internally if the *digital inputs* ([3.2.5, Page 135](#)) are used for operating the module.

2.5.6 EXE PHRASE1 (0xD1)

COMMAND CODE: 0xD1

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "1".

DESCRIPTION: None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xD1		Execute phrase "1".
S->M	0x04	0xD1	0x01 0x00 0xB0	Phrase 1 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 1, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte. This command together with the corresponding phrase number is generated internally if the *digital inputs* ([3.2.5, Page 135](#)) are used for operating the module.

2.5.7 EXE PHRASE2 (0xD2)

COMMAND CODE: 0xD2

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "2".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xD2		Execute phrase "2".
S->M	0x04	0xD2	0x02 0x00 0xB0	Phrase 2 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 2, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte. This command together with the corresponding phrase number is generated internally if the *digital inputs* ([3.2.5, Page 135](#)) are used for operating the module.

2.5.8 EXE PHRASE3 (0xD3)

COMMAND CODE: 0xD3

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "3".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xD3		Execute phrase "3".
S->M	0x04	0xD3	0x03 0x00 0xB0	Phrase 3 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 3, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte. This command together with the corresponding phrase number is generated internally if the *digital inputs* ([3.2.5, Page 135](#)) are used for operating the module.

2.5.9 EXE PHRASE4 (0xD4)

COMMAND CODE: 0xD4

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "4".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xD4		Execute phrase "4".
S->M	0x04	0xD4	0x04 0x00 0xB0	Phrase 4 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 4, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte. This command together with the corresponding phrase number is generated internally if the *digital inputs* ([3.2.5, Page 135](#)) are used for operating the module.

2.5.10 EXE PHRASE5 (0xD5)

COMMAND CODE: 0xD5

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "5".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xD5		Execute phrase "5".
S->M	0x04	0xD5	0x05 0x00 0xB0	Phrase 5 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 5, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte. This command together with the corresponding phrase number is generated internally if the *digital inputs* ([3.2.5, Page 135](#)) are used for operating the module.

2.5.11 EXE PHRASE6 (0xD6)

COMMAND CODE: 0xD6

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "6".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xD6		Execute phrase "6".
S->M	0x04	0xD6	0x06 0x00 0xB0	Phrase 6 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 6, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte. This command together with the corresponding phrase number is generated internally if the *digital inputs* ([3.2.5, Page 135](#)) are used for operating the module.

2.5.12 EXE PHRASE7 (0xD7)

COMMAND CODE: 0xD7

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "7".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xD7		Execute phrase "7".
S->M	0x04	0xD7	0x07 0x00 0xB0	Phrase 7 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 7, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte.

2.5.13 EXE PHRASE8 (0xD8)

COMMAND CODE: 0xD8

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "8".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xD8		Execute phrase "8".
S->M	0x04	0xD8	0x08 0x00 0xB0	Phrase 8 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 8, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte.

2.5.14 EXE PHRASE9 (0xD9)

COMMAND CODE: 0xD9

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "9".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xD9		Execute phrase "9".
S->M	0x04	0xD9	0x09 0x00 0xB0	Phrase 9 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 9, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte.

2.5.15 EXE PHRASE10 (0xDA)

COMMAND CODE: 0xDA

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "10".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xDA		Execute phrase "10".
S->M	0x04	0xDA	0x0A 0x00 0xB0	Phrase 10 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 10, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte.

2.5.16 EXE PHRASE11 (0xDB)

COMMAND CODE: 0xDB

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "11".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xDB		Execute phrase "11".
S->M	0x04	0xDB	0x0B 0x00 0xB0	Phrase 11 contains a "MO-

	D-Len	Cmd	Param	
				VE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 11, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte.

2.5.17 EXE PHRASE12 (0xDC)

COMMAND CODE: 0xDC

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "12".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xDC		Execute phrase "12".
S->M	0x04	0xDC	0x0C 0x00 0xB0	Phrase 12 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 12, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte.

2.5.18 EXE PHRASE13 (0xDD)

COMMAND CODE: 0xDD

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "13".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xD D		Execute phrase "13".
S->M	0x04	0xD D	0x0D 0x00 0xB0	Phrase 13 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 13, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte.

2.5.19 EXE PHRASE14 (0xDE)

COMMAND CODE: 0xDE

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "14".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xDE		Execute phrase "14".
S->M	0x04	0xDE	0x0E 0x00 0xB0	Phrase 14 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 14, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte.

2.5.20 EXE PHRASE15 (0xDF)

COMMAND CODE: 0xDF

DESCRIPTION: Special command, only 1 byte in size, for calling up phrase "15".

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Program line (UInt16) with command code included is sent.

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xDF		Execute phrase "15".
S->M	0x04	0xDF	0x0F 0x00 0xB0	Phrase 15 contains a "MOVE POS" command

MISCELLANEOUS: If "PRG EXE" is included in program line 15, then the complete subprogram will be automatically executed. This allows subprograms to be executed using only 1 byte.

2.5.21 PRG GOTO (0xC3)

COMMAND CODE: 0xC3

DESCRIPTION: Jumps to phrase number

PARAMETER (Master -> Slave): *Phrase number*

RESPONSE (Slave->Master): None

EXAMPLE: Only practical in the context of programming.

MISCELLANEOUS: Special command that is only required in sequential programs. Nothing happens in the event of a direct call-up. Special *programming software* tools should be used for programming the modules.

See also

 Brake [► 150]

2.5.22 PRG WAIT (0xC4)

COMMAND CODE 0xC4

DESCRIPTION Waits a specified period of time in [ms]

PARAMETER (Master -> Slave) Time [ms]

RESPONSE (Slave->Master) None

EXAMPLE Only practical in the context of programming.

MISCELLANEOUS Special command that is only required in sequential programs. In a direct call-up, the processor pauses for the specified period of time. Special programming software ([👉 3.2.7, Page 150](#)) should be used for programming the modules.

2.6 Other

2.6.1 GET STATE (0x95)

COMMAND CODE: 0x95

DESCRIPTION: Provides the status of the module and some further information, if required. The module can independently update this status at regular intervals.

PARAMETER (Master -> Slave):

- None
The module provides the data once. This can be used to switch off the previously set cyclical sending of data.
- *Time* (4 bytes)
The module independently transmits its status at the entered time interval (in the respective *unit system* ([👉 1.5, Page 22](#))).
- *Time* (4 bytes) *Modus* (1 byte)
The module independently transmits its status at the entered time interval (in the respective *unit system* ([👉 1.5, Page 22](#))).
The "Mode" code is used to parameterize which data is also to be provided in addition to the status:
Bit 1 (0x01): Position
Bit 2 (0x02): Speed
Bit 3 (0x04): Current

RESPONSE (Slave->Master): Optional data (observe the "Mode" code), then the status (2 bytes), which is structured as follows:

Referenced	Bit 1	0x01
Motion	Bit 2	0x02
Program sequence	Bit 3	0x04
Warning	Bit 4	0x08
Error	Bit 5	0x10

Brake	Bit 6	0x20
Motion blocked	Bit 7	0x40
Position reached	Bit 8	0x80

- Bit 1: The module is referenced.
- Bit 2: The module moves.
- Bit 3: The module is in program mode (an internal sequential program is active).
- Bit 4: There is a *warning* ([👉 2.8.1, Page 105](#)).
- Bit 5: There is an *error* ([👉 2.8.1, Page 105](#)).
- Bit 6: The brake is activated.
- Bit 7: A *movement was interrupted* ([👉 2.3.2, Page 55](#)).
- Bit 8: The *target position Position was reached* ([👉 2.3.3, Page 56](#)).
- In the case of an error, bits 9-16 also contain the *error code* ([👉 2.8.2, Page 107](#)).

EXAMPLE 1:

	D-Len	Cmd	Param	
M->S	0x06	0x95	0x00 0x00 0x80 0x3F 0x07	Status information is to be provided cyclically by the module every second. Aside from status bits, the position, speed, and current are also sent.
S->M (cyclically every second)	0x0F	0x95	0xD6 0xA3 0x70 0x41 0x56 0xC9 0x41 0x40 0x3C 0x41 0xEB 0x3E 0x03 0x00	Position: 0xD6..0x41, Speed: 0x56..0x40, Current: 0x3C..0x3E; module is in motion and referenced (0x03); no error (0x00)

EXAMPLE 2:

	D-Len	Cmd	Param	
M->S	0x01	0x95		Query status information once. The parameters queried last will also be sent.
S->M	0x0F	0x95	0x0x53 0x63 0xB7 0x41 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x61 0xD9	Position: 0x53..0x41, Speed: 0x00..0x00, Current: 0x00..0x00; module referenced, motion ended, brake actuated (0x61); fast stop actuated (0xD9)

EXAMPLE 3:

	D-Len	Cmd	Param	
M->S	0x06	0x95	0x00 0x00 0x00 0x00 0x01	Query status information once. Aside from status bits, the position is also sent.
S->M	0x07	0x95	0x00 0x00 0x00 0x00 0x20 0x00	Position: 0x00..0x00; module not referenced, brake actuated (0x20); no error (0x00)

MISCELLANEOUS: If you need to receive the position, speed, and current under CAN in one message, then the *fragmentation protocol* ([👉 1.4.6, Page 20](#)) must be used. Under Profibus, all information is accommodated in one Profibus message. A "Data" data code set once is retained and thus does not need to be reset each time. If the module is switched on, then the data code is set to "0x07"; all possible status information is therefore transferred completely.

NOTE

If all parameters (position, speed, current) have been transferred, only the lower 8 bits of the status are displayed under Profibus. These now come to be in byte 14, where the status is always provided up-to-date Profibus specifically ([☞ 1.4.5, Page 18](#)). The MsgCount that overwrites the upper 8 bits of the status word follows in byte 15.

NOTE

Under Profibus, the current position is transferred in bytes 10-13 as standard. If GET STATE is used to request all parameters, then the up-to-date position in bytes 10-13 is overwritten with the up-to-date value of the current. This is why it is recommended you set under Profibus the "Mode" byte to a maximum of "0x06" (speed and current will be transferred).

NOTE

Under Profibus, one should be careful when calling up "automatically". Under certain circumstances, it may be more favorable here to "poll" the data; in particular when the FREEZE mechanism is used.

2.6.2 GET STATE AXIS (0x98)

COMMAND CODE: 0x98

DESCRIPTION: Provides the status of an axis or a robot in the module and some further information, if required. The module can independently update this status at regular intervals.

PARAMETER (Master -> Slave):

- Only *Axis index* (1 byte)
The module provides the data of the axis once with the "Axis index" index. This can be used to switch off the previously set cyclical sending of data.
- *Time* (4 bytes) + *Axis index* (1 byte)
The module independently transmits the status of the axis at the entered time interval (in the respective *unit system* ([☞ 1.5, Page 22](#))) with the "Axis index" index.
- *Time* (4 bytes) + *Mode* (1 byte) *Axis index* (1 byte)
The module independently transmits the status of the axis at the entered time interval (in the respective *unit system* ([☞ 1.5,](#)

[Page 22](#)) with the "Axis index" index.

The "Mode" code is used to parameterize which data is also to be provided in addition to the status:

Bit 1 (0x01): Position

Bit 2 (0x02): Speed

Bit 3 (0x04): Current

RESPONSE (Slave->Master): Optional data (in accordance with the set "Mode"), then the status (2 bytes) followed by the axis index. The bits for the status are structured in the same way as the *GET STATE* ([2.6.1, Page 84](#)) command.

Examples: EXAMPLE 1:

	D-Len	Cmd	Param	
M->S	0x07	0x98	0x00 0x00 0x80 0x3F 0x07 0x01	Status information is to be provided cyclically by axis 1 on the module every second. Aside from status bits, the position, speed, and current are also sent.
S->M (cyclically every second)	0x10	0x98	0xD6 0xA3 0x70 0x41 0x56 0xC9 0x41 0x40 0x3C 0x41 0xEB 0x3E 0x03 0x00 0x01	Position: 0xD6..0x41, Speed: 0x56..0x40, Current: 0x3C..0x3E; module is in motion and referenced (0x03); no error (0x00); Axis index: 1 (0x01)

EXAMPLE 2:

	D-Len	Cmd	Param	
M->S	0x02	0x98	0x02	Query status information from axis 2 once. The parameters queried last will also be sent.
S->M	0x10	0x98	0x0x53 0x63 0xB7 0x41 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x61 0xD9 0x02	Position: 0x53..0x41, Speed: 0x00..0x00, Current: 0x00..0x00; module referenced, motion ended, brake actuated (0x61); fast stop actuated (0xD9); Axis index: 2 (0x02)

EXAMPLE 3:

	D-Len	Cmd	Param	
M->S	0x07	0x98	0x00 0x00 0x00 0x00 0x01 0x03	Query status information from axis 3 once. Aside from status bits, the position is also sent.
S->M	0x07	0x98	0x00 0x00 0x00 0x00 0x20 0x00 0x03	Position: 0x00..0x00; module not referenced, brake actuated (0x20); no error (0x00); Axis index: 3 (0x03)

MISCELLANEOUS: If you need to receive the position, speed, and current under CAN in one message, then the *fragmentation protocol* ([1.4.6, Page 20](#)) must be used. Under Profibus, all information is accommodated in one Profibus message. A "Mode" set once is retained and thus does not need to be reset each time. If the module is switched on, then the mode is set to "0x07"; all possible status information is therefore transferred completely.

NOTE

If all parameters (position, speed, current) have been transferred, only the lower 8 bits of the status are displayed in Profibus. These now come to be in byte 14, where the status is always provided up-to-date PROFIBUS specifically ([1.4.5, Page 18](#)). The MsgCount that overwrites the upper 8 bits of the status word follows in byte 15.

NOTE

Under Profibus, the current position is transferred in bytes 10-13 as standard. If GET STATE is used to request all parameters, then the up-to-date position in bytes 10-13 is overwritten with the up-to-date value of the current. This is why it is recommended you set under Profibus the "Mode" byte to a maximum of "0x07" (position, speed, and current will be transferred).

NOTE

In Profibus, one should be careful when calling up "automatically". Under certain circumstances, it may be more favorable here to "poll" the data; in particular when the FREEZE mechanism is used.

2.6.3 CMD REBOOT (0xE0)

COMMAND CODE: 0xE0

DESCRIPTION: The module is restarted.

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): The command is confirmed with "OK" (0x4F4B). The module then restarts and indicates *"INFO BOOT"* ([👉 2.8.2, Page 107](#)) after a successful reboot. (Exception: Serial communication without spontaneous message)

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xE0		
S->M	0x03	0xE0	0x4F 0x4B	Confirmation with "OK"
S->M	0x03	0x8A	0x00 0x01	Module successfully restarted

2.6.4 CMD DIO (0xE1)

COMMAND CODE: 0xE1

DESCRIPTION: Digital inputs/outputs can be set or read.

PARAMETER (Master -> Slave):

- None
The current status of the digital inputs/outputs is read.

- *1 byte*
The upper 4 bits can be used to set the 4 digital outputs.

RESPONSE (Slave->Master): If successful, "OK" (0x4F4B); with attached byte for the current status of the digital inputs in the lower 4 bits and the status of the digital outputs in the upper 4 bits.

Input 1	Bit 1	0x01
Input 2	Bit 2	0x02
Input 3	Bit 3	0x04
Input 4	Bit 4	0x08
Output 1	Bit 5	0x10
Output 2	Bit 6	0x20
Output 3	Bit 7	0x40
Output 4	Bit 8	0x80

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xE1		
S->M	0x04	0xE1	0x4F 0x4B 0x00	No inputs/outputs set

MISCELLANEOUS: Observe setting of digital inputs and outputs.



NOTICE

When the module is switched on, the outputs briefly have undefined states. Under certain circumstances, this may lead to the destruction of the connected up hardware.

2.6.5 CMD DIO AXIS(0xEB)

COMMAND CODE: 0xEB

DESCRIPTION: Digital inputs/outputs of individual axes can be set or read.

PARAMETER (Master -> Slave):

- Axis index
The current status of the digital inputs/outputs is read.
- Axis index + 1 byte
The upper 4 bits can be used to set the 4 digital outputs.

RESPONSE (Slave->Master): If successful, "OK" (0x4F4B); with attached byte for the current status of the digital inputs and the axis index in the lower 4 bits and the status of the digital outputs in the upper 4 bits.

Input 1	Bit 1	0x01
Input 2	Bit 2	0x02
Input 3	Bit 3	0x04
Input 4	Bit 4	0x08
Output 1	Bit 5	0x10
Output 2	Bit 6	0x20
Output 3	Bit 7	0x40
Output 4	Bit 8	0x80

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x02	0xE1	0x01	
S->M	0x05	0xE1	0x4F 0x4B 0x00 0x01	No inputs/outputs set for axis 1

MISCELLANEOUS: Observe setting of digital inputs and outputs.

**NOTICE**

When the module is switched on, the outputs briefly have undefined states. Under certain circumstances, this may lead to the destruction of the connected up hardware.

2.6.6 CMD GET DIO (0xE9)

COMMAND CODE: 0xE9

DESCRIPTION: Digital inputs/ outputs can be read.

PARAMETER (Master -> Slave):

3 types of requests are possible which differentiate in the number of transmitted parameters and in the type and number of requested digital inputs/ outputs. Differing responses are also returned by the module depending on the type (1-3) of request.

Type 1: The current status of all digital inputs is read as a bit vector. (The outputs cannot be read with type 1 requests):

PARAMETER (Master -> Slave):

- None

RESPONSE (Slave->Master):

- *Bit vector* (4 bytes)

Type 2: A specific digital input or output is read. Which input or output is to be read is determined by the "Address" parameter. The "status" of the digital input or output addressed is transferred in bit 0 for the transmitted data byte in the response:

PARAMETER (Master -> Slave):

- *Address* (4 bytes)

RESPONSE (Slave->Master):

- *Status* (1 byte)

Type 3: A sequential set of digital inputs or outputs is read. Which inputs or outputs are to be read is determined by the "Address" and "Length" parameters. Beginning with the input or output at "Address", as many inputs/outputs are read as the "Length" parameter specifies. In the response, the status of the digital inputs or outputs addressed is transferred as a bit vector in the transmitted "States" n data bytes. The number of transferred data bytes depends on the requested "Length" and is calculated as follows:

"(Length + 7) / 8", where "/" designates the integer division (division without remainder). Thus, for the "Length" of 1-8, the response constitutes one byte; for the "Length" of 9-16, the response constitutes two bytes, etc.:

PARAMETER (Master -> Slave):

- *Address* (4 bytes)
- *Length* (1 byte)

RESPONSE (Slave->Master):

- *Status* (n bytes)

REMARK

- Type 1 requests can only read inputs (not outputs).
- Which addresses are valid for type 2 or 3 depends on the module. For more on this, see the respective supplementary instructions.

Example:

Type 1	D-Len	Cmd	Param	
M->S	0x01	0xE9		Type 1: Read all inputs
S->M	0x05	0xE9	0x01 0x02 0x04 0x08	Type 1: Inputs 1, 10, 19 and 28 are set, all other inputs are deleted

2.6.7 CMD SET DIO (0xEA)

COMMAND CODE: 0xEA

DESCRIPTION: Digital outputs can be written. 3 types of requests are possible:

Type 1: The status of all digital outputs is written:

PARAMETER (Master -> Slave):

- *Bit array* (4 bytes)

RESPONSE (Slave->Master) If successful:

- "OK" 0x4F4B (2 bytes)

Type 2: A specific digital output is written. Which output is to be written is determined by the "Address" parameter. The output is set or deleted in accordance with bit 0 for the transmitted "Status":

PARAMETER (Master -> Slave):

- *Address* (4 bytes)
- *Status* (1 byte)

RESPONSE (Slave->Master) If successful:

- "OK" 0x4F4B (2 bytes)

Type 3: A sequential set of digital outputs is written. Which outputs are to be written is determined by the "Address" and "Length" parameters. Beginning with the output at "Address", as many outputs are written as the "Length" parameter specifies. The values that are to be written are coded as a bit vector in the "Status" parameter. The number of the data bytes to be transferred here depends on the "Length" and is calculated as follows: $(\text{Length} + 7) / 8$, where "/" designates the integer division (division without remainder). Thus, for the "Length" of 1-8, one byte will be transferred; for the "Length" of 9-16, two bytes will be transferred, etc.:

PARAMETER (Master -> Slave):

- *Address* (4 bytes)
- *Length* (1 byte)
- *Status* (n bytes)

RESPONSE (Slave->Master) If successful:

- "OK" 0x4F4B (2 bytes)

Irrespective of the type (1-3) of request, a response is thus always made with "OK" (0x4F4B), if successful. The module then sets/deletes the digital outputs addressed.

REMARK

- Which addresses are valid for type 2 or 3 depends on the module. For more on this, see the respective supplementary instructions.

Example:

Type 1	D- Len	Cmd	Param	
M->S	0x05	0xEA	0x00 0x00 0x00 0x00	Type 1: Set all outputs to 0
S->M	0x03	0xEA	0x4F 0x4B	"OK", outputs were set

**NOTICE**

When the module is switched on, the outputs briefly have undefined states. Under certain circumstances, this may lead to the destruction of the connected up hardware.

2.6.8 FLASH MODE (0xE2)

COMMAND CODE: 0xE2

DESCRIPTION: The module is prepared for a *firmware update* ([👉 4.4.4, Page 169](#)).

PARAMETER (Master -> Slave): Flash password

RESPONSE (Slave->Master): After the password has been successfully checked, "OK" (0x4F4B).

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x??	0xE2	<Password>	
S->M	0x03	0xE2	0x4F 0x4B	

MISCELLANEOUS: If the password input was successful, then it takes approx. 30 [s] (*) until the module moves independently into Flash mode (LED continually lights up green and red).

* Time is needed in order to charge a capacitor.

NOTE

After the update process via the serial interface with several modules, the switched off modules ("CMD DISCONNECT" ([👉 2.6.9, Page 97](#))) have to be restarted. (Disconnect the supply voltage).

2.6.9 CMD DISCONNECT (0xE6)

COMMAND CODE: 0xE6

DESCRIPTION: The module is disconnected from the bus system and deactivated.

PARAMETER (Master -> Slave):Flash password

RESPONSE (Slave->Master): After the password has been successfully checked, "OK" (0x4F4B).

EXAMPLE:

	D-Len	Cmd	Param	
M->S	0x01	0xE6	<Password>	
S->M	0x03	0xE6	0x4F 0x4B	

MISCELLANEOUS: If the password input was successful, then the module is switched off. The LEDs go out (except for motor voltage). This is required in a firmware update via the serial interface if the bus is not going to be separated. Only one, single module can be updated. Switched off modules can only be activated through switching the logic voltage off and then switching it on again.

2.6.10 CHANGE USER (0xE3)

COMMAND CODE: 0xE3

DESCRIPTION: The current user of the module is changed.

PARAMETER (Master -> Slave):

- None
User of "User" ([1.6.1, Page 24](#)) is set.
- *Password*
For the respective user. RESPONSE (Slave->Master): The command is always confirmed with "OK (0x4F4B)". The current user (0 = User, 1 = Diag, 2 = Profi, 3 = Advanced) is attached (1 byte).

EXAMPLE

	D-Len	Cmd	Param	
M->S	0x01	0xE3	<Password>	
S->M	0x04	0xE3	0x4F 0x4B 0x00	User of "User" active.

MISCELLANEOUS: If an incorrect password is entered, then "User" is always set. "User" is always active when restarting a module.

NOTE

With the "Diagnosis" user ([1.6.2, Page 24](#)), a second interface (serial, CAN, Bluetooth) is opened through which no motion commands can be sent to the module. However, "CMD FAST STOP" ([2.2.19, Page 53](#)) is possible.

2.6.11 CHECK MC PC COMMUNICATION (0xE4)

COMMAND CODE: 0xE4

DESCRIPTION: Communication from the module to the control unit can be checked. Defined data are sent from the module to the control unit. Both individual data can be requested as well as all data all at once. When requesting individual data, fragmentation is **not** required.

PARAMETER (Master -> Slave):

- None
All test data are sent at one stretch; fragmentation is required.

- **TEST FLOAT 1 (0x0101)**
The float value "-1.2345" (0x19 0x04 0x9E 0xBF) is sent and attached to this, the parameter code (0x0101).
- **TEST FLOAT 2 (0x0202)**
The float value "47.11" (0xA4 0x70 0x3C 0x42) is sent and attached to this, the parameter code (0x0202).
- **TEST INT32 1 (0x0303)**
The Int32 value "0x11223344" (287454020) is sent and attached to this, the parameter code (0x0303).
- **TEST INT32 2 (0x0404)**
The Int32 value "0xFFEEDDCC" (-1122868) is sent and attached to this, the parameter code (0x0404).
- **TEST INT16 1 (0x0505)**
The Int16 value "0x0200" (512) is sent and attached to this, the parameter code (0x0505).
- **TEST INT16 2 (0x0606)**
The Int16 value "0xAFFE" (-20482) is sent and attached to this, the parameter code (0x0606).

RESPONSE (Slave->Master): If no parameter was transferred, the following values have to be received by the control unit in the given order:

Data type	HEX value	DEC value
Float	0x19 0x04 0x9E 0xBF	-1.2345
Float	0xA4 0x70 0x3C 0x42	47.11
Int32	0x11223344	287454020
Int32	0xFFEEDDCC	-1122868
Int16	0x0200	512
Int16	0xAFFE	-20482

The appropriate parameters are used to transfer only one of the values (according to parameter).

EXAMPLE: See *Selected examples*. ([🔗 6.1, Page 182](#))

MISCELLANEOUS: This command can be used to check your own drivers. This command is not required for normal operation. The data exchange, including fragmentation from the module to its own control unit, can be tested using defined values.

NOTE

The parameter code is put at the end of the response to make sure that the test data come to be precisely at those positions where the real data will be.

2.6.12 CHECK PC MC COMMUNICATION (0xE5)

COMMAND CODE: 0xE5

DESCRIPTION: This can be used to check the communication from the control unit to the module.

PARAMETER (Master -> Slave) :

- Data can be transferred all at once in the following order:

Data type	HEX value	DEC value
Float	0x19 0x04 0x9E 0xBF	-1.2345
Float	0xA4 0x70 0x3C 0x42	47.11
Int32	0x11223344	287454020
Int32	0xFFEEDDCC	-1122868
Int16	0x0200	512
Int16	0xAFFE	-20482

A fragmentation process must be carried out here.

- To test individual data, the test data must first be transferred and, attached to that, the code for which test data are being referred to:
 - Send float "-1.2345" (0x19 0x04 0x9E 0xBF) and attached to this, the parameter code (0x0101)
 - Send float "47.11" (0xA4 0x70 0x3C 0x42) and attached to this, the parameter code (0x0202)
 - Send Int32 "0x11223344" (287454020) and attached to this, the parameter code (0x0303)
 - Send Int32 "0xFFEEDDCC" (-1122868) and attached to this, the parameter code (0x0404)
 - Send Int16 "0x0200" (512) and attached to this, the parameter code (0x0505)
 - Send Int16 "0xAFFE" (-20482) and attached to this, the parameter code (0x0606)

RESPONSE (Slave->Master): The module responds with "OK (0x4F4B)" if the corresponding test value was able to be

interpreted correctly. When transferring the test data all at once, the module responds with "OK (0x4F4B)" and attached to this is 1 byte in which bitwise coding indicates which data were not able to be interpreted correctly (bit is set to "1").

- Bit 1: The first float value (-1.2345) was not detected
- Bit 2: The second float value (47.11) was not detected
- Bit 3: The first Int32 value (0x11223344) was not detected
- Bit 4: The second Int32 value (0xFFEEDDCC) was not detected
- Bit 5: The first Int16 value (512) was not detected
- Bit 6: The second Int16 value (0xAFFE) was not detected

EXAMPLE: See *Selected examples* ([🔗 6.1, Page 182](#))

MISCELLANEOUS: This command can be used to check your own drivers. This command is not required for normal operation. It is possible to test the data exchange from the control unit to the module using individually defined values or using a defined data package in which fragmentation is required.

NOTE

The parameter code is put at the end of the response to make sure that the test data come to be precisely at those positions where the real data will be.

2.7 Fragmentation

2.7.1 FRAG ACK (0x87)

COMMAND CODE: 0x87

DESCRIPTION: Confirmation of a correctly processed fragment.

PARAMETER (Master -> Slave): *D-Len code for received fragment* (UInt16) if the control unit confirms the fragment to the module.

RESPONSE (Slave->Master): *D-Len code for received fragment* (UInt16) if the module confirms the fragment to the control unit.

EXAMPLE: See *Selected examples* ([👉 6.1, Page 182](#)).

NOTE

This command is only required for Profibus ([👉 1.4.6, Page 20](#)) if a fragmentation of messages is necessary.

2.7.2 FRAG START (0x84)

COMMAND CODE 0x84

DESCRIPTION: Indicates in a fragmented message that it is the first fragment.

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): None

EXAMPLE: See *Selected examples* ([👉 6.1, Page 182](#)).

MISCELLANEOUS: It is written directly after the D-Len byte. However, it is not included in D-Len since it only serves as a marker.

2.7.3 FRAG MIDDLE (0x85)

COMMAND CODE: 0x85

DESCRIPTION: Indicates in a fragmented message that it is a middle fragment.

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): None

EXAMPLE: See *Selected examples* ([👉 6.1, Page 182](#)).

MISCELLANEOUS: It is written directly after the D-Len byte. However, it is not included in D-Len since it only serves as a marker.

2.7.4 FRAG END (0x86)

COMMAND CODE 0x86

DESCRIPTION: Indicates in a fragmented message that it is the last fragment.

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): None

EXAMPLE: See *Selected examples*. ([👉 6.1, Page 182](#)).

Miscellaneous: It is written directly after the D-Len byte. However, it is not included in D-Len since it only serves as a marker.

2.8 Error messages

In the event of an error, D-Len always has exactly the value "2" or "3" in the data frame from the module to the control unit if the optional axis index is used. Either the defective command is in the command byte or one of the following "error commands". The parameter byte contains information on the *cause of the error* ([👉 2.8.2, Page 107](#)).



Error message

2.8.1 Error commands

2.8.1.1 CMD ERROR (0x88)

COMMAND CODE: 0x88

EXAMPLE:

	D-Len	Cmd	Param	
S->M	0x02	0x88	0xDA	A towing error has occurred. ("ERROR TOW" (↗ 2.8.2, Page 107))

DESCRIPTION:

A serious error has occurred which makes user intervention necessary. These errors must be acknowledged with "CMD ACK". The module is not ready for operation. The motor is de-energized. This error message is regularly sent every 15 seconds from the module to the control unit. All *error codes* ([↗ 6.4, Page 197](#)) as an overview.

- Serial communication: The first byte in a message changes to "0x03"
- CAN: The first three bits in the identifier result in "0x3"
- Profibus: An extended diagnosis is generated



WARNING

Risk of injury

In modules without a brake, the module may "stall", since the motor is de-energized in the event of a serious error.

2.8.1.2 CMD WARNING (0x89)

COMMAND CODE: 0x89

EXAMPLE:

	D-Len	Cmd	Param	
S->M	0x02	0x89	0xD6	The module is still in the upper software limit range ("ERROR SOFT HIGH" (↗ 2.8.2, Page 107))

DESCRIPTION:

- Software limit ranges were transgressed. A fast stop is initially triggered at this point and must be acknowledged. The module is then partially ready for operation. (Traveling out of the software limit range is permitted). If the module is moved out of the software limit range, then the warning will be deleted.
- *Upper temperature limit* ([↗ 3.2.5, Page 135](#)) was exceeded. The module will shut down in 1[min] if the temperature is not lowered.

These error messages are regularly sent from the module to the control unit (every 30 seconds). All *error codes* ([↗ 6.4, Page 197](#)) as an overview.

- Serial communication: The first byte in a message changes to "0x03"
- CAN: The first three bits in the identifier result in "3"
- Profibus: An extended diagnosis is generated

2.8.1.3 CMD INFO (0x8A)

See "CMD INFO" ([↗ 2.3.1, Page 54](#))

2.8.1.4 CMD ACK (0x8B)

COMMAND CODE: 0x8B

DESCRIPTION: Acknowledgement of an active error message.

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): "OK" (0x4F4B)

EXAMPLE:

	D-Len	Cmd	Param
M->S	0x01	0x8B	
S->M	0x03	0x8B	0x4F 0x4B

MISCELLANEOUS: When all errors have been successfully acknowledged, after sending "OK" (0x4F4B), an info message "INFO NO ERROR" ([👉 2.8.2, Page 107](#)) is also sent.

2.8.1.5 GET DETAILED ERROR INFO (0x96)

COMMAND CODE: 0x96

DESCRIPTION: Further information on the current error. The command must be sent immediately after the error has occurred. The displayed value is used by Schunk Service for a more exact fault diagnosis.

PARAMETER (Master -> Slave): None

RESPONSE (Slave->Master): Command (1 byte), error code (1 byte), data (float)

2.8.1.6 Examples:

EXAMPLE:

	D-Len	Cmd	Param
M->S	0x01	0x96	
S->M	0x07	0x96	0x88 0xD9 0x00 0x00 0x00 0x00

MISCELLANEOUS: If no error or no further information about it is available, then the command is answered with "INFO FAILED" ([👉 2.8.2, Page 107](#)).

2.8.2 Information codes and error codes

2.8.2.1 ERROR CABLE BREAK (0x76)

CODE: 0x76

DESCRIPTION: The communication cable is defective.

REMARK: This error is caused by a defective communication cable. It is only displayed once the communication has been re-established.

2.8.2.2 ERROR COMMUTATION (0xDD)

CODE: 0xDD

DESCRIPTION: The motor cannot commute.

REMARK: If this error occurs frequently, either the wrong commutation type has been selected or the Hall sensors for block commutation are defective or not connected. With regard to sine commutation, there is an error in the position measuring system.

2.8.2.3 ERROR CONFIG MEMORY (0xD2)

CODE: 0xD2

DESCRIPTION: The *configuration range* ([↗ 3.2, Page 117](#)) is incorrect.

REMARK: Data could not be written to EEPROM or EEPROM is defective.

2.8.2.4 ERROR CURRENT (0xDE)

CODE: 0xDE

DESCRIPTION: The *maximum current* ([↗ 3.2.1, Page 117](#)) was exceeded.

REMARK: Reduce the motor load and introduce intermediate steps, if necessary.

2.8.2.5 ERROR FAST STOP (0xD9)

CODE: 0xD9

DESCRIPTION: A fast stop was triggered.

2.8.2.6 ERROR FRAGMENTATION (0xDC)

CODE: 0xDC

DESCRIPTION: An error has occurred in the *fragmentation protocol* ([↗ 1.4.6, Page 20](#)).

REMARK: Data packets have been lost.

2.8.2.7 ERROR I²T (0xDF)

CODE: 0xDF

DESCRIPTION : An I²T error has occurred.

REMARK: Reduce load of module.

2.8.2.8 ERROR INITIALIZE (0xE0)

CODE: 0xE0

DESCRIPTION: The module could not be initialized properly.

REMARK: Check *configuration parameters* ([👉 3.2, Page 117](#)).

2.8.2.9 ERROR INTERNAL (0xE1)

CODE: 0xE1

DESCRIPTION: An internal error has occurred.

REMARK : The firmware is in an undefined status. Contact the service department and give exact details as to why the error may have occurred.

2.8.2.10 ERROR INVALID PHRASE (0xD4)

CODE: 0xD4

DESCRIPTION: The programmed phrase to be executed is incorrect.

REMARK: Check the programming and parameter limits.

2.8.2.11 ERROR LOGIC HIGH (0x73)

CODE: 0x73

DESCRIPTION: The logic voltage is too high.

REMARK: Check the logic voltage.

2.8.2.12 ERROR LOGIC LOW (0x72)

CODE: 0x72

DESCRIPTION: The logic voltage is too low.

REMARK: Check the logic voltage.

2.8.2.13 ERROR MATH (0xEC)

CODE: 0xEC

DESCRIPTION: A "mathematical" error has occurred, e.g. division by zero.

REMARK: Usually, a configuration parameter is incorrect and this results in the value range being exceeded. In most cases, a *controller parameter* ([👉 3.2.4, Page 132](#)) is set incorrectly. The *detailed*

error information ([🔗 2.8.1, Page 105](#)) can be used by Schunk Service to localize the error precisely.

2.8.2.14 ERROR MOTOR VOLTAGE HIGH (0x75)

CODE: 0x75

DESCRIPTION: The motor voltage is too high.

REMARK: Check the motor voltage.

2.8.2.15 ERROR MOTOR VOLTAGE LOW (0x74)

CODE: 0x74

DESCRIPTION: The motor voltage is too low.

REMARK: Check the motor voltage. If this error occurs frequently, the power supply unit of the motor voltage might be underdimensioned or the voltage supply cables to the module are not dimensioned correctly.

NOTE

This error can crop up directly after starting up the module if the module is connected with logic voltage and subsequently the motor voltage is connected.

2.8.2.16 ERROR OVERSHOOT (0x82)

CODE: 0x82

DESCRIPTION: The module has overrun the target position by "*max. overshoot*" ([🔗 3.2.4, Page 132](#)).

REMARK:

- The current needed for deceleration is too low. Increase the specified current.
- Check the controller parameters.

2.8.2.17 ERROR PROGRAM MEMORY (0xD3)

CODE: 0xD3

DESCRIPTION: The program memory is defective.

REMARK: The entire program memory must be deleted.

2.8.2.18 ERROR RESOLVER CHECK FAILED (0xEB)

CODE: 0xEB

DESCRIPTION: A parameter for the *resolver setting* ([↗ 3.2.6, Page 146](#)) is incorrect.

REMARK: -

2.8.2.19 ERROR SERVICE (0xD8)

CODE: 0xD8

DESCRIPTION: Please contact after-sales service.

2.8.2.20 ERROR SOFT HIGH (0xD6)

CODE: 0xD6

DESCRIPTION: The module has exceeded the software limit range.

REMARK: If "*CMD ERROR*" ([↗ 2.8.1, Page 105](#)) is present, it must be acknowledged. => The error is converted to a "*CMD WARNING*" ([↗ 2.8.1, Page 105](#)). Only now is it possible to move the module away from the software limit range using any *motion command* ([↗ 2.2, Page 33](#)).

2.8.2.21 ERROR SOFT LOW (0xD5)

CODE: 0xD5

DESCRIPTION: The module has exceeded the software limit range.

REMARK: If the command byte "*CMD ERROR*" ([↗ 2.8.1, Page 105](#)) is present, it must be acknowledged. => The error is converted to a "*CMD WARNING*" ([↗ 2.8.1, Page 105](#)). Only now is it possible to move the module away from the software limit range using any *motion command* ([↗ 2.2, Page 33](#)).

2.8.2.22 ERROR TEMP HIGH (0x71)

CODE: 0x71

DESCRIPTION: The temperature *has exceeded the permissible temperature range* ([↗ 3.2.5, Page 135](#)).

REMARK: Allow the module to cool down. Reduce the load.

2.8.2.23 ERROR TEMP LOW (0x70)

CODE: 0x70

DESCRIPTION: The temperature *has fallen below the permissible temperature range* ([↗ 3.2.5, Page 135](#)).

REMARK: Warm up the module.

2.8.2.24 ERROR TOO FAST (0xE4)

CODE: 0xE4

DESCRIPTION: The maximum *speed* ([↗ 3.2.1, Page 117](#)) was exceeded. (Motor is overspeeding)

2.8.2.25 ERROR TOW (0xDA)

CODE: 0xDA

DESCRIPTION: A *towing error* ([↗ 3.2.5, Page 135](#)) has occurred.

REMARK: Reduce load of module.

2.8.2.26 ERROR VPC3 (0xDB)

CODE: 0xDB

DESCRIPTION: The Profibus controller is not working correctly.

REMARK: This can only be determined by the "*Diagnosis*" ([↗ 1.6.2, Page 24](#)) user, since this message cannot be sent via Profibus (the Profibus controller is defective of course).

2.8.2.27 ERROR WRONG RAMP TYPE (0xC8)

CODE: 0xC8

DESCRIPTION: No valid *motion profile* ([↗ 3.2.5, Page 135](#)) has been selected for the positioning movement.

2.8.2.28 INFO BOOT (0x0001)

CODE: 0x0001

DESCRIPTION: The module has booted successfully. (*Info message* ([↗ 2.3.1, Page 54](#)))

REMARK: This is triggered after the module has been switched on and undergone a successful initialization after a *reboot* ([↗ 2.6.3, Page 90](#)). If this message appears frequently during operation, the logic voltage should be checked. The performance driver could also be defective.

NOTE

The code comprises 2 bytes => D-Len > "2" => if just the D-Len byte is queried, then this message will not be interpreted as an error. The standard D-Len == "2" => the error can therefore always be adhered to.

NOTE

If the Serial communication without spontaneous message ([👉 3.2.5, Page 135](#)) interface is selected, then this message will be suppressed.

2.8.2.29 INFO CHECKSUM (0x19)

CODE: 0x19

DESCRIPTION: The checksum is incorrect => the data is invalid.

2.8.2.30 INFO COMMUNICATION ERROR (0x09)

CODE: 0x09

DESCRIPTION: A serious error has occurred in serial communication.

2.8.2.31 INFO FAILED (0x05)

CODE: 0x05

DESCRIPTION: The command has failed.

REMARK: All of the parameters are correct, but the execution of the command is not possible at this time due to other reasons, e.g. the module is in "fast stop" mode. With "MOVE POS TIME" ([↗ 2.2.5, Page 40](#)), "MOVE POS TIME REL" ([↗ 2.2.6, Page 42](#)), this message appears if the parameters are correct, but the position cannot be reached with the values in the specified time. The error message also appears when a looping movement command "MOVE POS LOOP" ([↗ 2.2.7, Page 44](#)) is executed if the module is in the *software limit range* ([↗ 2.8.1, Page 105](#)).

2.8.2.32 INFO MESSAGE LENGTH (0x1D)

CODE: 0x1D

DESCRIPTION: D-Len does not match the received data.

2.8.2.33 INFO NO ERROR (0x0008)

CODE: 0x0008

DESCRIPTION: There are no other error messages present.

REMARK: This is generated immediately after "CMD ACK" if no other errors are present or if the module has moved outside of the software limit range.

NOTE

The code comprises 2 bytes => D-Len > "2" => if just the D-Len byte is queried, then this message will not be interpreted as an error. The standard D-Len == "2" => the error can therefore always be adhered to.

2.8.2.34 INFO NO RIGHTS (0x03)

CODE: 0x03

DESCRIPTION: You do not have the appropriate rights to execute the command.

2.8.2.35 INFO SEARCH SINE VECTOR (0x0007)

CODE: 0x0007

DESCRIPTION: A search is running for the space vector for the sine commutation. 60% of the maximum current is used for the phases.

REMARK: Following power-up of the device, this is carried out once before any motion commands are executed if *standstill commutation* ([☞ 1.8, Page 30](#)) is not active and *commutation type* ([☞ 3.2.1, Page 117](#)) is set to PMSM.

NOTE

The code comprises 2 bytes => D-Len > "2" => if just the D-Len byte is queried, then this message will not be interpreted as an error. The standard D-Len == "2" => the error can therefore always be adhered to.

2.8.2.36 INFO TIMEOUT (0x10)

CODE: 0x10

DESCRIPTION: A timeout occurred during communication.

REMARK: The data could not be sent, or more data was anticipated and was not received on time.

2.8.2.37 INFO UNKNOWN AXIS INDEX (0x11)

CODE: 0x11

DESCRIPTION: The module has received the wrong axis index.

2.8.2.38 INFO UNKNOWN COMMAND (0x04)

CODE: 0x04

DESCRIPTION: The sent command is not recognized.

2.8.2.39 INFO WRONG BAUDRATE (0x16)

CODE: 0x16

DESCRIPTION: Wrong baudrate during communication.

2.8.2.40 INFO WRONG PARAMETER (0x1E)

CODE: 0x1E

DESCRIPTION: One of the specified parameters is outside of the permitted range.

REMARK: If a parameter is found to be incorrect, all old parameters will be retained, even if the other parameters should lie in the valid range.

2.8.2.41 ERROR REFERENCED (0x06) or NOT REFERENCED (0x06)

CODE: 0x06

DESCRIPTION: The module is not referenced and can therefore not execute the command.

REMARK: The module must be referenced in order to carry out a positioning movement.

NOTE

Further information on the current error "GET DETAILED ERROR INFO" ([👉 2.8.1.5, Page 107](#)) When reporting "REF SWITCH MOVED" check whether the index pulse has been shifted relative to the reference point. Solution to the Problem above "Distance to the index" ([👉 3.2.3.9, Page 131](#))

2.8.2.42 ERROR HARDWARE VERSION (0x83)

The hardware version of the control board could not be determined.

3 Configuration parameters

3.1 In general

All configuration parameters are permanently saved in the internal EEPROM.

3.2 EEPROM

Access to the respective elements is controlled in the module's firmware. There are various users here who are each protected by passwords. A change of user is possible by using the "*CHANGE USER*" ([👉 2.6.10, Page 98](#)) command. The module recognizes the following users:

- User:
Standard user. This is active when the module is switched on. It has very limited parameterizing capacities for the module, but can fully operate it.
- Diag:
This is used for activating a second interface (serial, Bluetooth, CAN) for diagnosis purposes. The second interface may not be used for sending motion commands.
- Profi:
This can change all important parameters. Incorrect parameterization can result in unanticipated module behavior. However, the module cannot be destroyed.
- Advanced:
This can change all important parameters. Incorrect parameterization can lead to the destruction of the module!
- Root:
Access to all parameters. Incorrect parameterization can lead to the destruction of the module!

3.2.1 Motor

3.2.1.1 Serial number

ACCESS RIGHTS: *Root* ([👉 1.6.5, Page 26](#))

DESCRIPTION: Serial number of the installed motor.

DATA: (UInt32) => 0 .. 4294967296

3.2.1.2 Voltage

ACCESS RIGHTS: *Root* ([🔑 1.6.5, Page 26](#))

DESCRIPTION: Rated voltage of the motor. This can be used to calculate both the derived brake control PWM (duty cycle) as well as the maximum permissible PWM for test purposes. This value is also needed for automatic controller configuration.

DATA: (UInt16) => 0 .. 65535



NOTICE

An incorrect entry can lead to the destruction of the electronics.

3.2.1.3 Type

ACCESS RIGHTS: *Advanced* ([🔑 1.6.4, Page 26](#))

DESCRIPTION: The motor type can be selected.

- DC (0x00): Brush type DC motor
- BLDC (0x01): Electronically commutated brushless DC motor with block commutation.
- PMSM (0x02): Electronically commutated brushless DC motor with sine commutation.

DATA: (UInt16) => 0 .. 65535



NOTICE

An incorrect entry can lead to the destruction of the electronics.

3.2.1.4 I²T

ACCESS RIGHTS: *Advanced* ([🔑 1.6.4, Page 26](#))

DESCRIPTION: I²T monitoring can be activated. If the load is too high, an I²T error ([🔑 2.8.2, Page 107](#)) is triggered.

With I²T monitoring, it is assumed that the maximum current may be present for 3 seconds (is equal to 100%). If a value of < 100% is entered, the time will be shortened accordingly (I²T monitoring is triggered earlier). Values greater than 100%, and also 0% lead to *ERROR INITIALIZE* ([🔑 2.8.2, Page 107](#)).

DATA : (UInt8) => 1 .. 100%

3.2.1.5 Pole pairings

ACCESS RIGHTS: *Root* ([🔑 1.6.5, Page 26](#))

DESCRIPTION: Electric pole pairings of the motor. Only required for brushless DC motors. Has effects on the calculation of speeds, positions, and commutation patterns.

DATA: (UInt16) => 0 .. 65535

3.2.1.6 Terminal resistance

ACCESS RIGHTS: *Root* ([🔑 1.6.5, Page 26](#))

DESCRIPTION: Terminal resistance is needed both for test functions to limit maximum currents as well as for automatic controller configuration.

DATA: (Float) [Ohm]



NOTICE

An incorrect entry can lead to the destruction of the electronics.

3.2.1.7 Inductance

ACCESS RIGHTS: *Root* ([👉 1.6.5, Page 26](#))

DESCRIPTION: Terminal inductance is needed for automatic controller configuration.

DATA: (Float) [mH]



NOTICE

An incorrect entry can lead to the destruction of the electronics.

3.2.1.8 Max. current

ACCESS RIGHTS: *Advanced* ([👉 1.6.4, Page 26](#))

DESCRIPTION: Maximum permissible current that may briefly flow through the motor's phases. If this current is exceeded for a long period of time (several ms), then a fast stop will be triggered with the error message *ERROR CURRENT* ([👉 2.8.2, Page 107](#)).

DATA: (4 bytes) 0.00 .. 29.99[A] or 0 .. 29999[mA] depending on the *unit system* ([👉 1.5, Page 22](#))



NOTICE

An incorrect entry can lead to the destruction of the electronics.

3.2.1.9 Nom. current

ACCESS RIGHTS: *Advanced* ([👉 1.6.4, Page 26](#))

DESCRIPTION: Current that may flow permanently through the motor's phases. If this is exceeded for a long period of time, then an *I²T* error ([👉 2.8.2, Page 107](#)) will be triggered.

Type: (4 bytes) Configured in the preset *unit system* ([👉 1.5, Page 22](#)).

**NOTICE**

An incorrect entry can lead to the destruction of the electronics.

3.2.1.10 Max. speed

ACCESS RIGHTS: *Advanced* ([☞ 1.6.4, Page 26](#))

DESCRIPTION: Maximum permissible speed of the system (output side).

DATA: (4 bytes) Configured in the preset *unit system* ([☞ 1.5, Page 22](#)).

3.2.1.11 Max. acceleration

ACCESS RIGHTS: *Advanced* ([☞ 1.6.4, Page 26](#))

DESCRIPTION: Maximum permissible acceleration of the system (output side).

DATA: (4 bytes) Configured in the preset *unit system* ([☞ 1.5, Page 22](#)).

3.2.1.12 Max. jerk

ACCESS RIGHTS: *Advanced* ([☞ 1.6.4, Page 26](#))

DESCRIPTION: Maximum permissible jerk of the system (output side). The jerk is the temporal alteration of acceleration. This parameter is only evaluated if a *positioning movement with jerk limitation* ([☞ 3.2.5, Page 135](#)) is executed.

DATA: (4 bytes) Configured in the preset *unit system* ([☞ 1.5, Page 22](#)).

3.2.1.13 Commutation table

ACCESS RIGHTS: *Root* ([☞ 1.6.5, Page 26](#))

DESCRIPTION: The Hall sensor table valid for the unit is set here for block commutation using Hall sensors. If the entry is incorrect, the motor will not move at all or only generates very little torque.

DATA: (UInt16) 0 .. 12

3.2.1.14 Offset phase A

ACCESS RIGHTS: Root ([🔑 1.6.5, Page 26](#))

DESCRIPTION: Zero point alignment for the first current sensor. This value should vary in a range between 1700 - 2200. Otherwise it is thought that the hardware may be defective.

DATA: (UInt16) 0 .. 65535



NOTICE

An incorrect value may lead to unpredictable behavior in the drive (only travels in one direction, jerks badly, overspeed).

3.2.1.15 Offset phase B

ACCESS RIGHTS: Root ([🔑 1.6.5, Page 26](#))

DESCRIPTION: Zero point alignment for the second current sensor. This value should vary in a range between 1700 - 2200. Otherwise it is thought that the hardware may be defective.

DATA: (UInt16) 0 .. 65535



NOTICE

An incorrect value may lead to unpredictable behavior in the drive (only travels in one direction, jerks badly, overspeed).

3.2.1.16 Metering current range

ACCESS RIGHTS: Root ([🔑 1.6.5, Page 26](#))

DESCRIPTION: Maximum metering current range for the current sensor used internally.

DATA: (Float)



NOTICE

An incorrect entry can lead to the destruction of the electronics.

3.2.2 Gears

3.2.2.1 Serial number

ACCESS RIGHTS: *Root* ([🔑 1.6.5, Page 26](#))

DESCRIPTION: Serial number of the gear.

DATA: (UInt32)

3.2.2.2 Ratio 1

ACCESS RIGHTS: *Profi* ([🔑 1.6.3, Page 26](#))

DESCRIPTION: Gear ratio or reduction ratio from motor to output.

DATA: (Float)

3.2.2.3 Ratio 2

ACCESS RIGHTS: *Profi* ([🔑 1.6.3, Page 26](#))

DESCRIPTION: Gear ratio or reduction ratio from position measuring system to output. This is only required if the *position measuring system is installed between the gearing* ([🔑 3.2.6, Page 146](#)).

DATA: (Float)

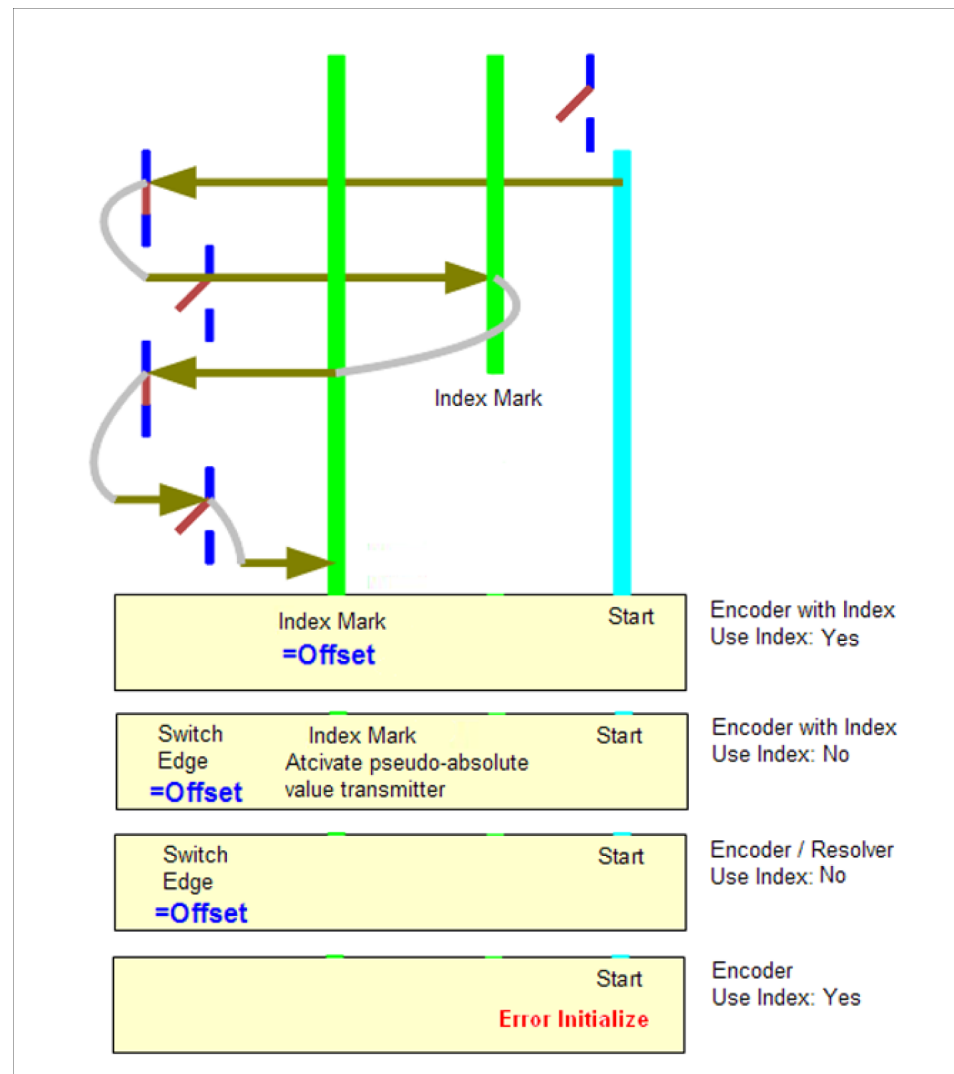
3.2.3 Referencing

3.2.3.1 Type

ACCESS RIGHTS: *Profi* ([↗ 1.6.3, Page 26](#))

DESCRIPTION: The type of referencing can be defined. (Observe *Pos type* ([↗ 3.2.6, Page 146](#)) section when using the encoder with index track).

- Internal switch, left (0x00) / right (0x01)
The internal reference switch is used for referencing. The direction of motion when the reference switch is active is determined by direction "left" or "right".
- External switch IN0, left (0x02) / right (0x03)
An external reference switch (IN0) (*digital inputs* ([↗ 3.2.5, Page 135](#))) is used for referencing. The direction of motion when the reference switch is active is determined by direction "left" or "right".



NOTE

Ensure that the proximity switch remains in either status for at least 200 ms before switching. (Adjust referencing speed, adjust control cams).

Functional principle:

If the parameter "Distance to index" is set to "0", then the distance from the reference event (switching detected) to the index pulse is measured and saved during the reference movement. For subsequent reference movements, this is measured again and compared with the saved value. If the values are within an internally set tolerance range, then reference can be successfully completed. Furthermore, an index pulse at an "unfavorable" point (index pulse shortly before or after switching) can be "corrected".

NOTE

If the distance between index pulse and reference event exceeded the permissible tolerance, then referencing will be aborted and "ERROR REFERENCED" displayed. It may be necessary to measure the distance from the reference event to the index pulse again. To do this, the parameter "Distance to index" must be set to "0".

NOTE

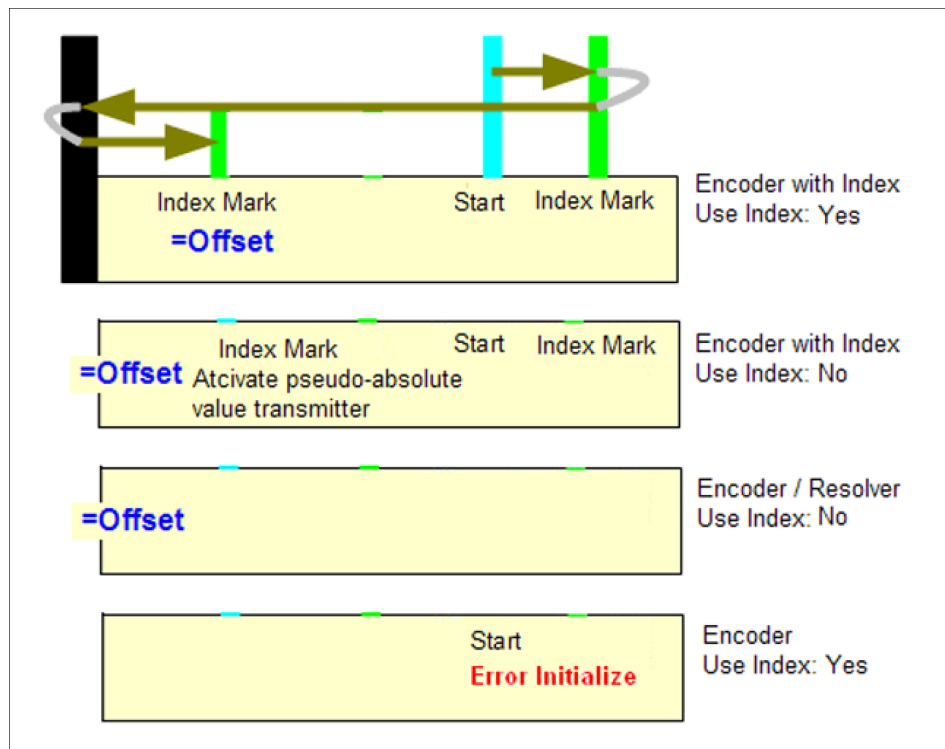
The module should always be referenced in the same position.

NOTE

If error continues to occur regularly, contact SCHUNK Service.

- Speed, left (0x04) / right (0x05)

A speed movement is executed for referencing. If the module moves to a fixed stop, then this will be recognized as a reference point. The direction of rotation is defined via "left" or "right".



NOTE

This type of referencing is only recommended if a fixed end stop is available.

- Speed with stroke monitoring, left (0x06) / right (0x07)
In addition to the procedure described above, a movement is made to the opposite fixed stop after the first fixed stop has been approached. The traveled distance must be greater than the difference in the *software limit ranges* ([3.2.5, Page 135](#))
=> Referencing successful.

NOTE

This type of referencing is only recommended if a fixed end stop is available.

- Current, left (0x08) / right (0x09)
A current movement is executed. The current is increased until the module moves. If the current exceeds the *max. reference current* ([3.2.3, Page 123](#)), then it is assumed that a fixed stop has been reached which is recognized as a reference point.

NOTE

This type of referencing is only recommended if a fixed end stop is available.

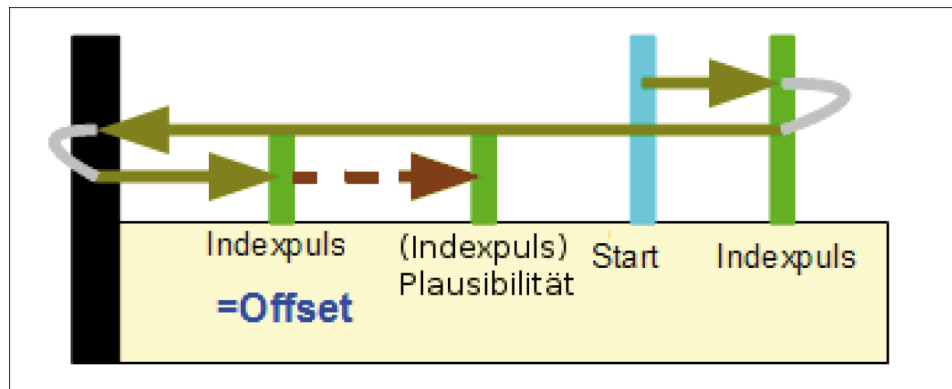
NOTE

Jamming, sluggishness in the mechanical system, or a "forgotten" workpiece can also lead to the rated current being exceeded. This would then also be interpreted as a fixed end stop, even if there are none present.

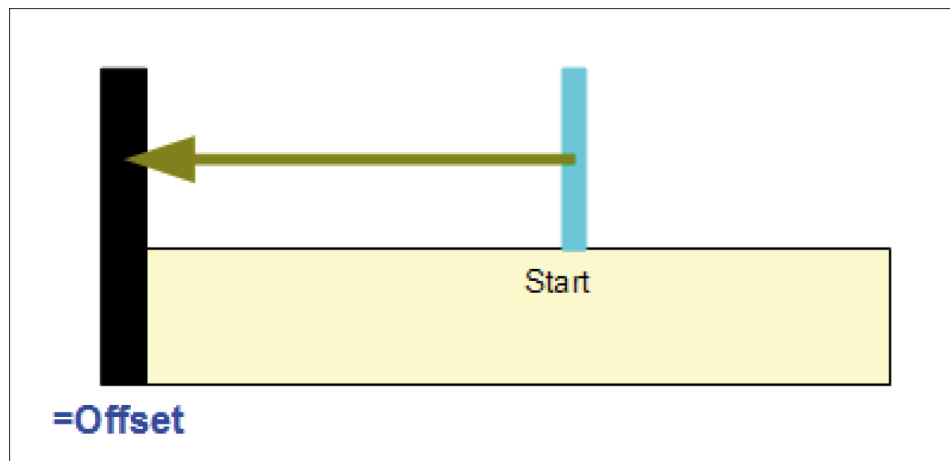
- Current with stroke monitoring, left (0x0A) / right (0x0B)
In addition to the procedure described above, a movement is made to the opposite fixed end stop after the first fixed end stop has been approached. The traveled distance must be greater than the difference in the *software limit ranges* ([3.2.5, Page 135](#)) => Referencing successful.

NOTE

This type of referencing is only recommended if a fixed end stop is available.



Referencing up to stop with activated index track

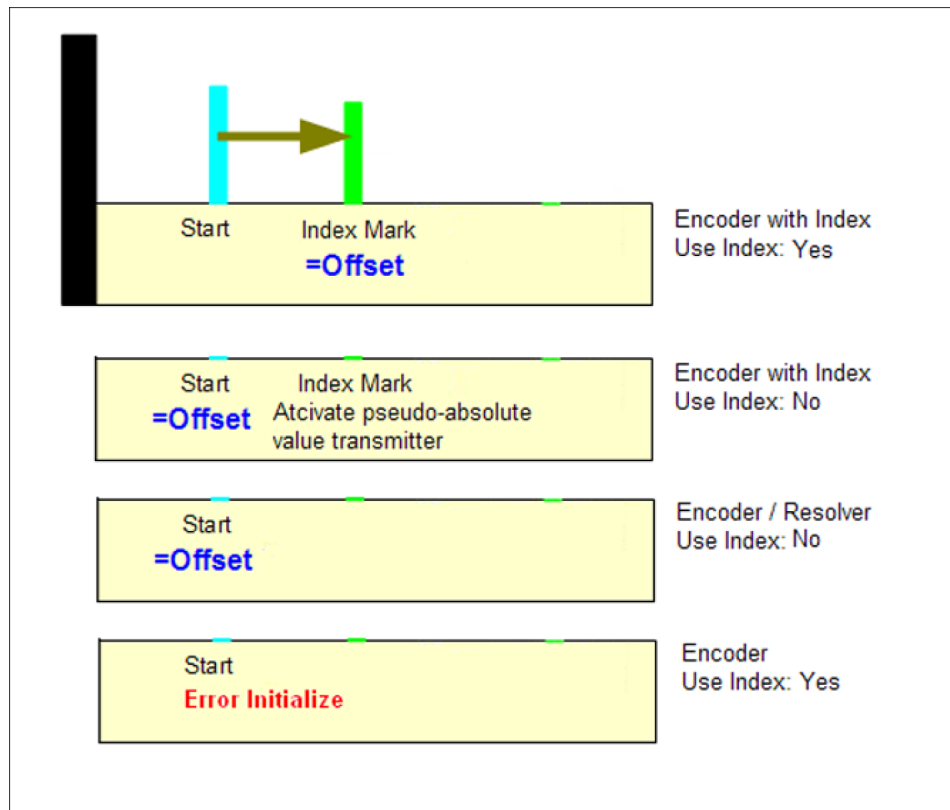


Referencing up to stop

- None (0x0C)

After the *CMD REFERENCE*

([👉 2.2.1, Page 33](#)) command has been sent, the current position will be seen as the reference position.



Referencing of "none"

- Manual (0x0D)

After the *CMD REFERENCE*

([👉 2.2.1, Page 33](#)) command has been sent, a "simulated" reference movement will be carried out should the module already be referenced. The module will move to the reference mark and, depending on the *setting* ([👉 3.2.3, Page 123](#)), will then

move to position "0".

If the module is not referenced, then *manual referencing* ([👉 2.2.2, Page 34](#)) will be started.

NOTE

If manual referencing ([👉 2.2.2, Page 34](#)) has already been performed successfully once, then the referencing types set previously will automatically be converted to "Manual".

DATA: (UInt16) 0 .. 65535

3.2.3.2 Max. reference current

ACCESS RIGHTS: *Profi* ([☞ 1.6.3, Page 26](#))

DESCRIPTION: Specified current in [%] of the *motor's rated current* ([☞ 3.2.1, Page 117](#)). The reference current does not exceed the specified value.

DATA: (UInt8) 0 .. 255

3.2.3.3 Speed

ACCESS RIGHTS: *Profi* ([☞ 1.6.3, Page 26](#))

DESCRIPTION: Specification of speed for reference movements with internal or external reference switch, and speed reference runs. This also applies to manual referencing. The value here represents 100% speed.

If the *position "0.0"* ([☞ 3.2.3, Page 123](#)) is to be moved to after the referencing, then this speed will be used for the positioning command.

DATA: (4 bytes) Configured in the preset *unit system* ([☞ 1.5, Page 22](#)).

3.2.3.4 Acceleration

ACCESS RIGHTS: *Profi* ([☞ 1.6.3, Page 26](#))

DESCRIPTION: Specification of acceleration for reference movements using internal or external reference switch, speed reference movements, and using manual referencing. If the *position "0.0"* ([☞ 3.2.3, Page 123](#)) is to be moved to after the referencing, then this acceleration value will be used for the positioning command.

DATA: (4 bytes) Configured in the preset *unit system* ([☞ 1.5, Page 22](#)).

3.2.3.5 Offset Zero Position

ACCESS RIGHTS: *Profi* ([☞ 1.6.3, Page 26](#))

DESCRIPTION: Position offset after successful referencing (zero offset)

DATA: (4 bytes) Configured in the preset *unit system* ([☞ 1.5, Page 22](#)).

3.2.3.6 Approach 0 after referencing

ACCESS RIGHTS: Profi ([1.6.3, Page 26](#))

DESCRIPTION: The position "0.0" is approached after successful referencing. The *speed* ([3.2.3, Page 123](#)) and *acceleration* ([3.2.3, Page 123](#)) are observed.

DATA: (Bool) YES / NO

3.2.3.7 Timeout

ACCESS RIGHTS: Profi ([1.6.3, Page 26](#))

DESCRIPTION: Maximum period of time that the reference movement may take.

DATA: (Float) According to *unit system* ([1.5, Page 22](#))[s] or [ms]

3.2.3.8 The use of the index

ACCESS RIGHTS: Profi ([1.6.3, Page 26](#))

DESCRIPTION: The index track is used for referencing.



⚠ CAUTION

It should be referenced with the index track. If the index puls is located on an inconvenient point, it is possible that the respective positions, during a repeated referencing, differs itself by one motor- rotation.

Remedy: Postpone the reference mark slightly.

This applies for all reference marks except "Internal switch" and "External switch".

3.2.3.9 Distance to the index

ACCESS RIGHTS: Advanced

DESCRIPTION: The distance of the "reference event" for the index puls is saved here. If this value is set to "0", this value is determined again during the next referencing.

3.2.3.10 Max. Distance Switch

ACCESS RIGHTS: Advanced

DESCRIPTION: Maximum allowed distance to the reference event (detects switching edge) to the index pulse. Is indicated in encoder ticks. See also *Distance to Index* ([↗ 3.2.3.9, Page 131](#)).

3.2.4 Controllers

3.2.4.1 KR current

ACCESS RIGHTS: *Profi* ([↗ 1.6.3, Page 26](#))

DESCRIPTION: Proportional term of current controller. With *current limiting control* ([↗ 3.2.4, Page 132](#)), this is the proportional term for the current limiting control

DATA: (Float)

3.2.4.2 TN current

ACCESS RIGHTS: *Profi* ([↗ 1.6.3, Page 26](#))

DESCRIPTION: Integral term of the current controller. This is not required for *current limiting control* ([↗ 3.2.4, Page 132](#)).

DATA: (Float)

3.2.4.3 KR velocity

ACCESS RIGHTS: *Profi* ([↗ 1.6.3, Page 26](#))

DESCRIPTION: Proportional term of velocity controller

DATA: (Float)

3.2.4.4 TN velocity

ACCESS RIGHTS: *Profi* ([↗ 1.6.3, Page 26](#))

DESCRIPTION: Integral term of velocity controller

DATA: (Float)

3.2.4.5 KR position

ACCESS RIGHTS: *Profi* ([↗ 1.6.3, Page 26](#))

DESCRIPTION: Proportional term of position controller

DATA: (Float)

3.2.4.6 Position variation

ACCESS RIGHTS: *Profi* ([↗ 1.6.3, Page 26](#))

DESCRIPTION: Position window in which position control is canceled (depending on *brake configuration* ([↗ 3.2.7, Page 150](#))), control is continued or the brake is activated or Position reached is signaled.

DATA: (4 bytes) Configured in the preset *unit system* ([↗ 1.5, Page 22](#)).

3.2.4.7 Structure

ACCESS RIGHTS: *Profi* ([↗ 1.6.3, Page 26](#))

DESCRIPTION: Structural design of the control loop.

- Current - Speed (0x00)
Current control and speed control operate independently of each other.
- Cascade (0x01)
Position, speed, and current controllers are cascade connected => Current controlled (set current is not to be exceeded) positioning or speed movements are possible (e.g. no prepositioning required for a gripping process). In this mode, the specified current ("*SET TARGET CURRENT*" ([↗ 2.2.14, Page 49](#))) is not exceeded in all motion types.
- Speed with current limiting (0x02)
Current control is not active. In velocity or positioning movements, the specified current ("*SET TARGET CURRENT*" ([↗ 2.2.14, Page 49](#))) is limited. In contrast to the cascade, the current is not controlled, but limited (current limiting control).
- Speed with PWM limiting (0x02)
Current control is not active. In velocity or positioning movements, the PWM's duty cycle is limited. The ratio of current to duty cycle is calculated using the *terminal resistance* ([↗ 3.2.1, Page 117](#)) of the motor.

NOTE

Since the PWM's duty cycle is directly limited (voltage limitation), it is possible that the drive no longer reaches its full speed, which is why positioning movements could possibly take considerably longer than anticipated.

DATA: (3 bits) 0 .. 7

NOTE

If the controller structure is changed, the controller parameters might have to be adapted.

3.2.4.8 Max. overshoot

ACCESS RIGHTS: *Profi* ([☞ 1.6.3, Page 26](#))

DESCRIPTION: If the module moves via this position window during a positioning movement, then the error message *ERROR OVERSHOOT* ([☞ 2.8.2, Page 107](#)) will be generated. This value must be set greater than the maximum permissible *position variation* ([☞ 3.2.4, Page 132](#)).

DATA: (4 bytes) Configured in the preset *unit system* ([☞ 1.5, Page 22](#)).

3.2.5 Device

3.2.5.1 Serial number

ACCESS RIGHTS: *Root* ([☞ 1.6.5, Page 26](#))

DESCRIPTION: Serial number of the module.

DATA: (UInt32)

3.2.5.2 Unit system

ACCESS RIGHTS: *User* ([☞ 1.6.1, Page 24](#))

DESCRIPTION: All inputs and outputs to the module or from the module use the set *unit system* ([☞ 1.5, Page 22](#)). If the unit system is converted from float

([mm](0x00),[m](0x01),[inches](0x02),[rad](0x03),[degrees](0x04)) to integer

[μm](0x06),[μdegrees](0x07),[μinches](0x08),[millidegrees](0x09)), then all units of time and units of current are also interpreted as integer => [s] -> [ms]; [A] -> [mA]. The internal unit system has the number (0x05).

DATA: (5 bits) 0 .. 31



NOTICE

Changing the unit system ([☞ 1.5, Page 22](#)) affects a large number of EEPROM parameters, which are also adapted immediately.

3.2.5.3 Communication mode

ACCESS RIGHTS:

User ([↗ 1.6.1, Page 24](#))

DESCRIPTION: The active communication system is set.

- Auto (0x00)
An attempt is made to automatically find and set the communication system. To date, testet *Hardware* ([↗ 6.5, Page 199](#)).

NOTE

If a module is connected to a serial and Profibus interface at the same time, then the Profibus interface (depending on the hardware) will be immediately recognized after the module is switched on. In this case, there is then no possibility of communicating via the serial interface.

NOTE

This operating mode tries to find the connected communication module independently. In that case the first data packets that have been sent to the module could be destroyed or distorted. It is possible that the superordinated system can be disturbed.

NOTE

When the start-up is completed, the communication-type "Auto" should be changed to the actual used communication system.

- Serial communication (0x01)
- CAN (0x02)
- Profibus DPV0 (0x03)
- REPLACEMENT Serial communication without spontaneous message (0x04)
Spontaneous messages ([↗ 2.3, Page 54](#)) are deactivated.
(incl. INFO BOOT)

DATA: (5Bit) 0 ..31

3.2.5.4 Invert direction of rotation of motor

ACCESS RIGHTS: *Profi* ([↗ 1.6.3, Page 26](#))

DESCRIPTION: The direction of rotation of the motor is defined.


! WARNING

Incorrect configuration may lead to unexpected results. (The module rotates at an unexpectedly high speed!)

DATA: (Bool) YES / NO

3.2.5.5 Invert position measurement

ACCESS RIGHTS: *Profi* ([🔗 1.6.3, Page 26](#))

DESCRIPTION: The direction of measurement for the position measuring system is defined. Interchanged A and B encoder tracks can be exchanged using software.


! WARNING

Incorrect configuration may lead to unexpected results. (The module rotates at an unexpectedly high speed!)

NOTE

If the direction of rotation of the motor and the position measuring system are inverted at the same time, then a "right-turning" module can be configured from a "left-turning" one, or a "positive" closing gripper can be configured from a "positive" opening gripper.

DATA: (Bool) YES / NO

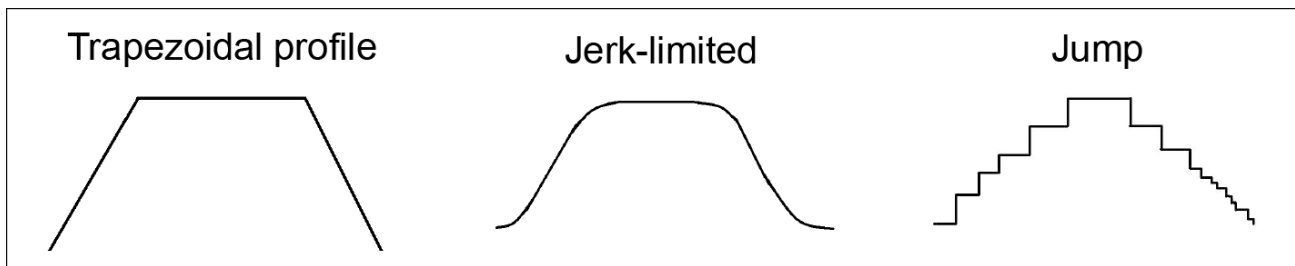
3.2.5.6 Position ramp

ACCESS RIGHTS: *Profi* ([↗ 1.6.3, Page 26](#))

DESCRIPTION: The ramp type for the positioning movement is entered.

- Trapezoid (0x00)
A trapezoid is taken as the basis for the calculation of the motion profile. This profile is, if necessary, dynamically connected in the case of "MOVE POS TIME" ([↗ 2.2.5, Page 40](#)) (curved paths). The travel time can be calculated. (Change-over points are controlled according to times)
- Jerk-limited (0x01)
A path with jerk limitation is calculated for positioning movements "MOVE POS" ([↗ 2.2.3, Page 36](#)), "MOVE POS REL" ([↗ 2.2.4, Page 38](#)), and "MOVE POS LOOP" ([↗ 2.2.7, Page 44](#)). The parameter "jerk-limited" ("SET TARGET JERK" ([↗ 2.2.13, Page 48](#))) is only used in this ramp type. In "MOVE POS TIME" ([↗ 2.2.5, Page 40](#)), "MOVE POS TIME REL" ([↗ 2.2.6, Page 42](#)) curved paths, this profile is deactivated (this procedure is necessary owing to computing time reasons).
- Trapezoid SRU (0x02)
A trapezoid is taken as the basis for the calculation of the motion profile. The travel time is not calculated. (Change-over points are controlled according to positions)
- Jump (0x03)
A path profile is not calculated here, but rather the position jump directly specified. Internal path planning is switched off. Depending on the interpolation interval of the external interpolator, it may be necessary to adapt the *controller parameters* ([↗ 3.2.4, Page 132](#)). This motion profile is only supported by "MOVE POS TIME" ([↗ 2.2.5, Page 40](#)), "MOVE POS TIME REL" ([↗ 2.2.6, Page 42](#)), "MOVE POS" ([↗ 2.2.3, Page 36](#)), "MOVE POS REL" ([↗ 2.2.4, Page 38](#)), and "MOVE POS LOOP" ([↗ 2.2.7, Page 44](#)) use the jerk-limited path profile in this case.

DATA: (3 bits) 0-7



3.2.5.7 Program after restart

ACCESS RIGHTS: *Advanced* ([🔑 1.6.4, Page 26](#))

DESCRIPTION: The program stored in EEPROM is started from line "0" directly after the module is switched on.

	! WARNING
	The module may move immediately after being switched on without there being a control unit connected.

NOTE

The program is not started until the module finds a communication interface, which is why the serial interface should have been set as a communication type ([🔑 2.4.1, Page 59](#)). All other bus systems do not allow a start until an active master has been found.

DATA: (Bool) YES / NO

3.2.5.8 Endless

ACCESS RIGHTS: *Profi* ([🔑 1.6.3, Page 26](#))

DESCRIPTION: The axis can turn endlessly.

DATA: (Bool) YES / NO

NOTE

This option is not recommended for grippers!

3.2.5.9 Digital inputs

ACCESS RIGHTS: *Profi* ([🔑 1.6.3, Page 26](#))

DESCRIPTION: Using digital inputs.

- Normal (0x00)
Digital inputs can be activated externally without triggering any actions in the module. They can be queried at any time via "CMD DIO" ([2.6.4, Page 91](#)).
- Program (0x01)

NOTE

Program mode via digital inputs is only possible when serial communication ([3.2.5, Page 135](#)) is set in every operating condition of the module. All other communication types do not allow a start until an active master has been found.

Previously programmed "phrases" can be executed. Before the very first phrase can be executed, all inputs have to be set to "low" once. Up to 7 phrases ([2.5.5, Page 73](#)) can then be selected. Input 1 is used as "enable". The positions of inputs 2 - 4 are only adopted when there is a rising edge at input 1. If just input 1 is set, then a reference movement ([2.2.1, Page 33](#)) is executed.

Switching procedure:

- 1 Set all inputs to "low".
- 2 Set inputs 2, 3, 4 arbitrarily (see table below).
- 3 Set input 1 to "high".

	Input 1	Input 2	Input 3	Input 4	Version
FSK	IN 0	IN 1	IN 2	IN 3	
	low	low	low	low	Original state
	low -> high	low	low	low	Reference movement
	low -> high	high	low	low	Program block 0
	low -> high	low	high	low	Program block 1
	low -> high	high	high	low	Program block 2
	low -> high	low	low	high	Program block 3

	Input 1	Input 2	Input 3	Input 4	Version
	low -> high	high	low	high	Program block 4
	low -> high	low	high	high	Program block 5
	low -> high	high	high	high	Program block 6

NOTE

When the external switches ([3.2.3, Page 123](#)) are activated, the "Program" operating mode is no longer an option.

DATA: (3 bits) 0 .. 7

3.2.5.10 Digital outputs

ACCESS RIGHTS: *Profi* ([1.6.3, Page 26](#))

DESCRIPTION: Using digital outputs.

- Normal (0x00)
Digital outputs can be set via "CMD DIO" ([2.6.4, Page 91](#)).
- Status + movement OUT2 (0x01)
The status of the module is reported to the digital outputs.
Out2 reports that the module is in motion.

	FSK	Status of module at "low"
Output 1	OUT0	Referenced
Output 2	OUT1	Error message
Output 3	OUT2	Movement running
Output 4	OUT3	Movement finished

- Status + position reached OUT2 (0x02)
The status of the module is reported to the digital outputs.
Out2 reports whether the module has reached the target position.

	FSK	Status of module at "low"
Output 1	OUT0	Referenced
Output 2	OUT1	Error message
Output 3	OUT2	Target position reached
Output 4	OUT3	Movement finished

- Status + brake OUT2 (0x03)
The status of the module is reported to the digital outputs.
Out2 reports the status of the brake.

	FSK	Status of module at "low"
Output 1	OUT0	Referenced
Output 2	OUT1	Error message
Output 3	OUT2	Status of brake
Output 4	OUT3	Movement finished

- Status + warning OUT2 (0x04)
The status of the module is reported to the digital outputs.
Out2 reports whether a warning is present.

	FSK	Status of module at "low"
Output 1	OUT0	Referenced
Output 2	OUT1	Error message
Output 3	OUT2	Warning present
Output 4	OUT3	Movement finished

- Status + program sequence OUT2 (0x05)
The status of the module is reported to the digital outputs.
Out2 indicates whether the module is in "program mode".

	FSK	Status of module at "low"
Output 1	OUT0	Referenced
Output 2	OUT1	Error message
Output 3	OUT2	Program sequence
Output 4	OUT3	Movement finished

DATA: (3 bits) 0 .. 7



NOTICE

When the module is switched on, the outputs briefly have undefined states. Under certain circumstances, this may lead to the destruction of the connected up hardware.

3.2.5.11 Analog output

ACCESS RIGHTS: *Profi* ([🔗 1.6.3, Page 26](#))

DESCRIPTION: Using the analog output (0 - 10V).

- None (0x00)
Not used => 0V
- Position (0x01)
The position is converted to an analog value.
Position 0.0 .. 5V
Maximum for absolute values of minimum or maximum position is depicted at 0V or 10V. For example, minimum position is -5.0, maximum position is +10.0 => Position = 0[V], +10.0 = 10.0[V], -5.0 = 2.5[V]
- Speed (0x02)
Speed is converted to an analog value:
 - Negative maximum speed = 0V
 - Standstill = 5V
 - Positive maximum speed = 10V
- Current (0x03)
0A => 0V
Maximum current = 10V, configured *here* ([🔗 3.2.1, Page 117](#)).
- Maximum (0x04)
Analog output = 10V

DATA: (3 bits) 0 .. 7

3.2.5.12 ID

ACCESS RIGHTS: *User* ([🔗 1.6.1, Page 24](#))

DESCRIPTION: Unique individual module identifier. Also see *SET CONFIG* ([🔗 2.4.1, Page 59](#)).

DATA: (UInt8) 0 .. 255

3.2.5.13 Group

ACCESS RIGHTS: *User* ([🔗 1.6.1, Page 24](#))

DESCRIPTION: Unique group identifier for the module. Also see *SET CONFIG* ([🔗 2.4.1, Page 59](#)). In Profibus, "Real no add change" is saved here.

DATA: (UInt8) 0 .. 255

3.2.5.14 CAN baudrate

ACCESS RIGHTS: *User* ([☞ 1.6.1, Page 24](#))

DESCRIPTION: Transmission rate for CAN. Valid values: 50, 100, 125, 250, 500, 1000. Also see *SET CONFIG* ([☞ 2.4.1, Page 59](#)).

DATA: (UInt16) 0 .. 65535

3.2.5.15 Min. position

ACCESS RIGHTS: *Profi* ([☞ 1.6.3, Page 26](#))

DESCRIPTION: Minimum position allowed (software limit range). This is ignored if "*endless*" ([☞ 3.2.5, Page 135](#)) is set. Used when *referencing* ([☞ 3.2.3, Page 123](#)) with "stroke control".

DATA: (4 bytes) Configured in the preset *unit system* ([☞ 1.5, Page 22](#)).

3.2.5.16 Max. position

ACCESS RIGHTS: *Profi* ([☞ 1.6.3, Page 26](#))

DESCRIPTION: Maximum position allowed (software limit range). This is ignored if "*endless*" ([☞ 3.2.5, Page 135](#)) is set. Used when *referencing* ([☞ 3.2.3, Page 123](#)) with "stroke control".

DATA: (4 bytes) Configured in the preset *unit system* ([☞ 1.5, Page 22](#)).

3.2.5.17 Towing error

Image for towing error ACCESS RIGHTS: *Profi* ([☞ 1.6.3, Page 26](#))

DESCRIPTION: Towing error. This value must not be exceeded during a positioning movement. Exceeding this value results in an error message ("*ERROR TOW*" ([☞ 2.8.2, Page 107](#))).

DATA: (4 bytes) Configured in the preset *unit system* ([☞ 1.5, Page 22](#)).

3.2.5.18 Min. temperature

ACCESS RIGHTS: *Advanced* ([☞ 1.6.4, Page 26](#))

DESCRIPTION: Minimum permissible working temperature. If this value is not reached, then an error message will result. ("*ERROR TEMP LOW*" ([☞ 2.8.2, Page 107](#)))

DATA: (Float) [°C]

3.2.5.19 Max. temperature

ACCESS RIGHTS: *Root* ([👉 1.6.5, Page 26](#))

DESCRIPTION: Maximum permissible working temperature. If this value is exceeded, there will be a warning. There will be an error message if the temperature does not drop within 1[min]. ("ERROR TEMP HIGH" ([👉 2.8.2, Page 107](#)))

DATA: (Float) [°C]

3.2.5.20 Serial communication baudrate

ACCESS RIGHTS: *User* ([👉 1.6.1, Page 24](#))

DESCRIPTION: Transmission rate for serial communication. Valid values: 1200, 2400, 4800, 9600, 19200, 38400. Also see *SET CONFIG* ([👉 2.4.1, Page 59](#)).

DATA: (UInt16) 0 .. 65535

3.2.6 Positioning

3.2.6.1 Serial number

ACCESS RIGHTS: *Advanced* ([👉 1.6.4, Page 26](#))

DESCRIPTION: Serial number of the position measuring system

DATA: (UInt32)

3.2.6.2 Type

ACCESS RIGHTS: *Root* ([↗ 1.6.5, Page 26](#))

DESCRIPTION: The type of measuring system is defined.

- Encoder (0x00)
Measuring system encoder without index track. Observe *parameter 1* ([↗ 3.2.6, Page 146](#)).
- Index encoder (0x01)
Encoder with index track. The index track is evaluated for all *reference movements* ([↗ 3.2.3, Page 123](#)). To ensure that an index track can be found at the correct location, it is possible that with some referencing types the drive moves back and forth several times with short movements or makes small movements in the "wrong" direction. The modules also move in referencing type "None", because a search is made for the next index pulse. Observe *parameter 1* ([↗ 3.2.6, Page 146](#)).
- Resolver (0x02)
Resolver system in which the excitation current can be set. Observe *parameter 1* ([↗ 3.2.6, Page 146](#)) and *parameter 2* ([↗ 3.2.6, Page 146](#)).

NOTE

In motors with a resolver ([↗ 3.2.6, Page 146](#)), "juddering" may occur in "Positioning ramp type" ([↗ 3.2.5, Page 135](#)) "jerk-limited". Another "Positioning ramp type" ([↗ 3.2.5, Page 135](#)) has to be selected in that case.

- Differential encoder (0x06)
Differential encoder without index track. Observe *parameter 1* ([↗ 3.2.6, Page 146](#)).
- Index encoder, differential (0x07)
Differential encoder with index track. The index track is evaluated for all *reference movements* ([↗ 3.2.3, Page 123](#)). To ensure that an index track can be found at the correct location, it is possible that with some referencing types the drive moves back and forth several times with short movements or makes small movements in the "wrong" direction. The modules also move in referencing type "None", because a search is made for the next index pulse. Observe *parameter 1* ([↗ 3.2.6, Page 146](#)).

DATA: (UInt8)

3.2.6.3 Installation position

ACCESS RIGHTS: *Advanced* ([🔗 1.6.4, Page 26](#))

DESCRIPTION: The installation position of the position measuring system.

- Input side: The position measuring system is mounted directly on the input end.
- Output side: The position measuring system is mounted directly on the output end.
- Between the gearing: The position measuring system is mounted in the middle of the gear.

DATA: (2 bits) 0 .. 3

3.2.6.4 Parameter 1

ACCESS RIGHTS: *Advanced* ([🔗 1.6.4, Page 26](#))

DESCRIPTION: Parameter 1 for the position measuring system

- Encoder: Ticks per revolution (4x evaluation)
- Resolver: Frequency of the voltage at the exciter coil in [kHz].
Permitted values: 8 [kHz], 4 [kHz], 2 [kHz], 1 [kHz].

DATA: (UInt16)

3.2.6.5 Parameter 2

ACCESS RIGHTS: *Advanced* ([🔗 1.6.4, Page 26](#))

DESCRIPTION: Parameter 2 for the position measuring system

- Encoder: not required
- Resolver: Amplitude of the input voltage at the exciter coil [%].
This has to be determined using technical measuring equipment. The output voltage at the receiver coils must not get into saturation. **Data:** (UInt16)

3.2.6.6 Offset Measuring System

ACCESS RIGHTS: *Advanced* ([↗ 1.6.4, Page 26](#))

DESCRIPTION: Twisting the position measuring system opposite the motor phases. Configured in the preset *unit system* ([↗ 1.5, Page 22](#)). Under certain circumstances, this value is determined automatically. Also see *Standstill commutation* ([↗ 1.8, Page 30](#)). If this value is set at "0", a new search is made for the "*sine pointer*" ([↗ 1.8, Page 30](#)). The *referencing* ([↗ 1.7, Page 27](#)) is deleted in the process.

DATA: (Float)

NOTE

The module should be able to move freely in all directions for the pointer search. The module is moved in a jerking fashion up to two motor revolutions. Communication with the module is not possible during this period of time. This process may require up to 30 seconds.

3.2.6.7 Motion threshold

ACCESS RIGHTS: *Profi* ([↗ 1.6.3, Page 26](#))

DESCRIPTION: The value in [%] of the *maximum speed* ([↗ 3.2.1, Page 117](#)). If this value is not reached, then the module is treated as stationary. See *Schunk Motion Tool status display* ([↗ 4.4.1, Page 165](#))

DATA: (Float)

3.2.6.8 ADC offset

ACCESS RIGHTS: *Root* ([↗ 1.6.5, Page 26](#))

DESCRIPTION: ADC offset for "centering" the input signal. Only required for cos/sin measuring systems and resolvers.

DATA: (Int16)

3.2.7 Brake

3.2.7.1 Serial number

ACCESS RIGHTS: *Root* ([🔗 1.6.5, Page 26](#))

DESCRIPTION: Serial number of the brake.

DATA: (UInt32)

3.2.7.2 Type

ACCESS RIGHTS: *Advanced* ([🔗 1.6.4, Page 26](#))

DESCRIPTION: Brake type. The brake voltage is adjusted per software using the configured motor voltage and the brake type.

DATA: (UInt16)



NOTICE

A configuration error leads to a non-functioning brake.

3.2.7.3 Use of brakes

ACCESS RIGHTS *Profi* ([🔗 1.6.3, Page 26](#))

DESCRIPTION: Use of the brake.

- Not used (0x00)
The brake is not used. It is only activated in the event of power failure. It is released one time only, immediately after the module has been started.
- Only in the event of an error (0x01)
The brake is only activated in the event of an error. Otherwise, the drive is permanently controlled.
- Normal (0x02)
The brake is activated in the event of an error and at the end of the movement. If a brake is configured, then the *pseudo-absolute encoder* ([🔗 1.7, Page 27](#)) is possibly active should further conditions be met.

DATA: (UInt16) 0 .. 65535

3.2.7.4 Timeout

ACCESS RIGHTS: *Profi* ([🔗 1.6.3, Page 26](#))

DESCRIPTION: Time needed by the brake to build up or reduce the magnetic field. Depending on *unit system* ([🔗 1.5, Page 22](#)), [s] or [ms].

DATA: (Float)

NOTE

A change in this value has no effect on the cycle speed of motion commands. An incorrect setting may lead to premature wear on the brakes.

4 Schunk Motion Tool

This software is used for the start-up and testing of SCHUNK-modules with SCHUNK motion protocol.

4.1 Prerequisites

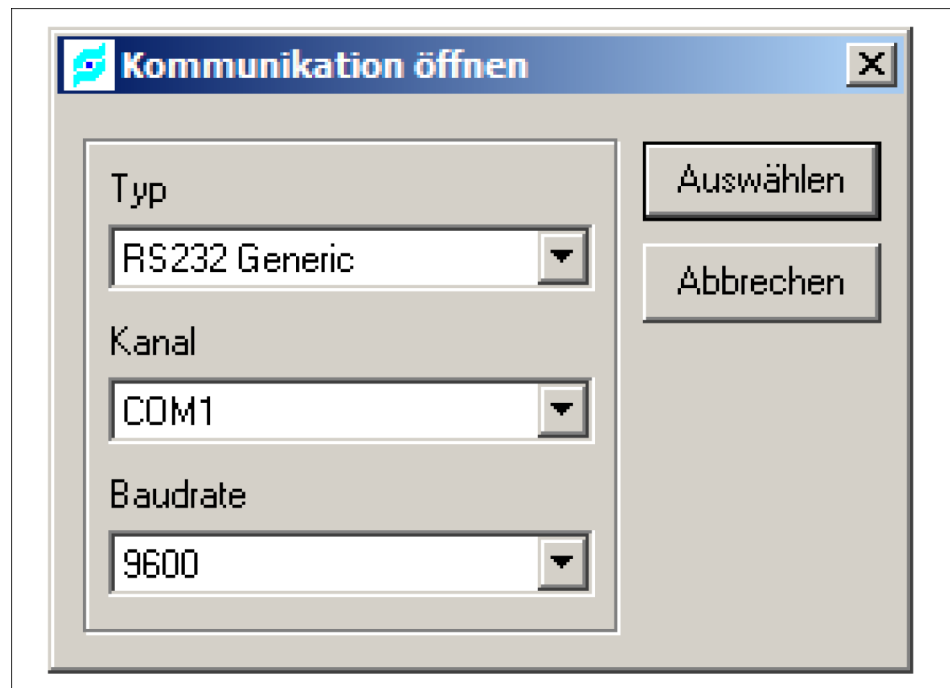
- Operating system: Windows 2000, XP, Vista, 7
- Graphics: 1024x768 with 16-bit color depth

4.2 First steps

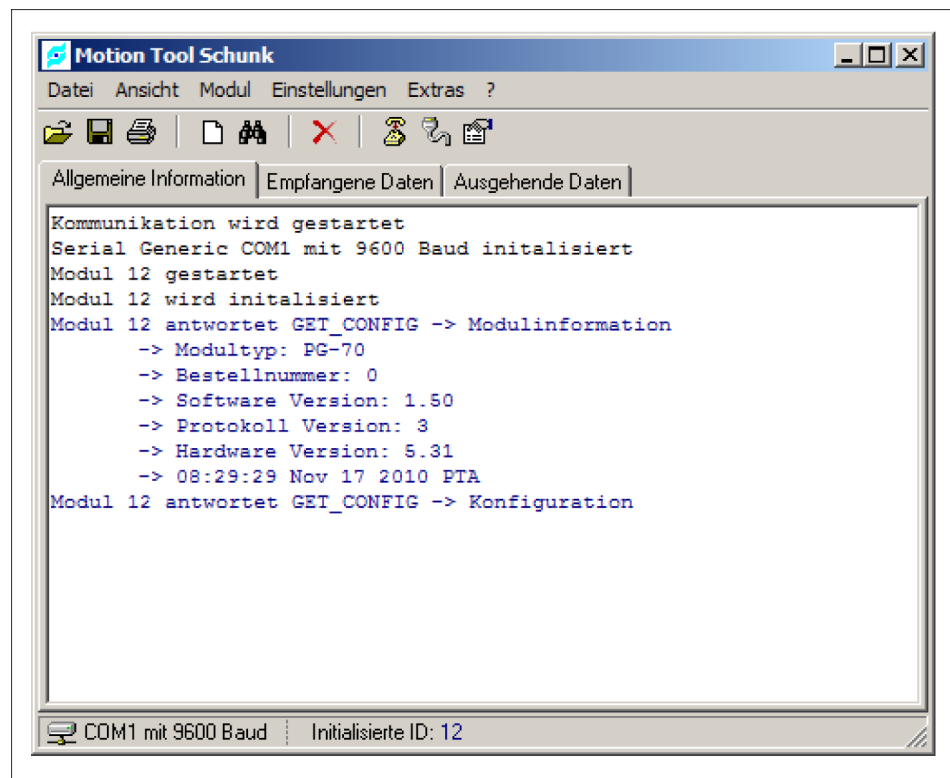
All modules have to be connected to the PC through at least one communication system. Right now, the following communication interfaces are supported by the software:

- Serial communication (COM ports)
- CAN: Cards from *Vector Informatik GmbH*
- CAN: Cards from *esd electronic system design gmbh*
- CAN: PCAN-USB converter from *PEAK-System Technik GmbH*
- CAN: Cards from *IXXAT Automation GmbH* (VCI V3 driver)
- CAN: Cards from *Softing AG*
- Profibus DPV0: Cards from *Hilscher* (preconfiguration required!)
- Profibus DPV0: *Siemens AG* CP56xx controller series (preconfiguration required!)

Please refer to the respective manufacturers for installation instructions. A list of the supported hardware, with further details, can be found in the *Appendix* ([👉 6.5, Page 199](#)).



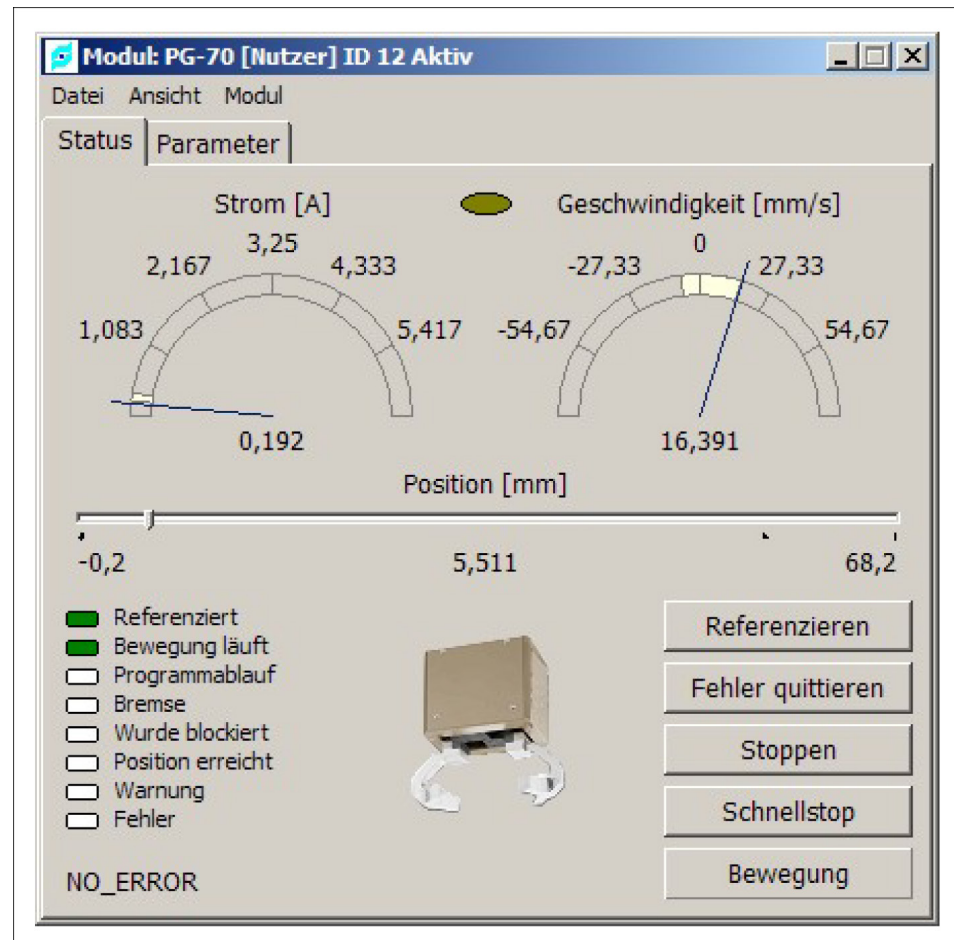
After selecting the communication interface , this will be opened automatically. Thanks to the bus search , all connected up and switched on modules will now be detected automatically. Should, under *Basic settings* 114, the "No automatic initialization of modules" option be activated, the module only has to be initialized (open module window) manually .



In addition to the main window, there is a module window for each module found which can be used to test all functions of the

respective module.

Using *F1*, the help system can be started at any time to receive further information.



See also

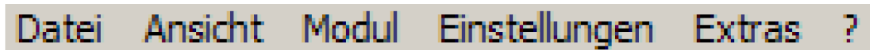
📄 [Appendix \[► 182\]](#)

4.3 Main window

In the *main window* (Fig. [4.2, Page 152](#)), connections are managed, communication on the active bus system is reported, and application settings are processed.

The main window consists of:

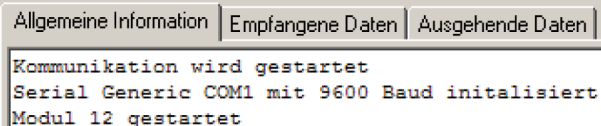
- *Menu bar*
- *Toolbar*
- *Output window*
- *Status bar*



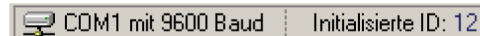
Menu bar



Toolbar



Output window



Status bar

4.3.1 Toolbar

The toolbar can be used to quickly address the most important functions.

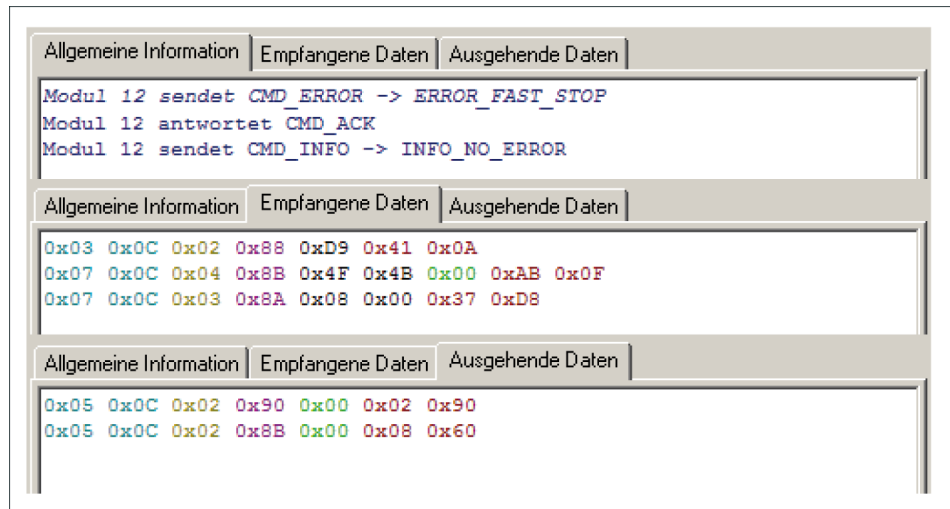
	The EEPROM content of all initialized modules will be saved.
	The EEPROM content of all initialized modules will be printed out.
	A module with a known ID can be initialized.
	The entire communication interface will be searched for active modules.
	"CMD FAST STOP" (🔗 2.2.19, Page 53) will be sent to all modules.
	The set communication interface will be opened or closed.
	A search will be made for available communication interfaces.
	Various <i>basic settings</i> (🔗 4.3.4, Page 159) can be changed.

4.3.2 Menu

- File
 - Load: Load EEPROM configuration from available file.
 - Save: The EEPROM content of all initialized modules will be saved.
 - Print: The EEPROM content of all initialized modules will be printed out.
 - Finish: The application will be closed.
- View
 - Display bus details: A detailed message on the status of the bus system will be displayed under "General information".
 - Set bookmark: A marker will be set in the output windows.
 - Delete output data: The content in all output windows will be deleted.
 - Module ... : The view of the selected module will be activated (module window).
- Module
 - Initialization with ID: A module with a known ID will be initialized
 - Deinitialization via ID: An already initialized module will be deinitialized.

- Search bus: The entire bus will be searched for active modules.
- Find diagnosis: search diagnosis-modules
- All fast stop: A *fast stop command* ([☞ 2.2.19, Page 53](#)) will be sent to all initialized modules.
- Settings
 - Create/close link: The set communication interface will be opened or closed.
 - Open communication: An automatic search is made for installed communication interfaces, and one of these is to be selected manually.
 - Language: The current language for "Motion Tool Schunk" can be changed. The change of language applies after the application has been restarted.
 - Settings: Various *basic settings* ([☞ 4.3.4, Page 159](#)) can be changed.
- Extras
 - CRC: A *Tool* ([☞ 4.3.5, Page 161](#)) for calculating the CRC16 checksum.
 - Number Converter: A *Tool* ([☞ 4.3.5, Page 161](#)) for mutually converting several number formats. (*Data Format* ([☞ 1.4.1, Page 11](#)))
 - Program Editor: A *Tool* ([☞ 4.3.5, Page 161](#)) for creating *module programs* ([☞ 2.4.4, Page 68](#)).
- ?
 - Motion Control: This documentation.
 - Help: "Motion Tool Schunk" help.
 - About: Here you can find further information about the program.

4.3.3 Output window



Output window

In the *output window*, all information relating to the individually connected modules is featured in plain text under the "*General information*" tab. Every single module is highlighted in color. The incoming data on the respective bus system are displayed in their raw form under the "*Received data*" tab. The "*Outgoing data*" tab shows the raw data sent.

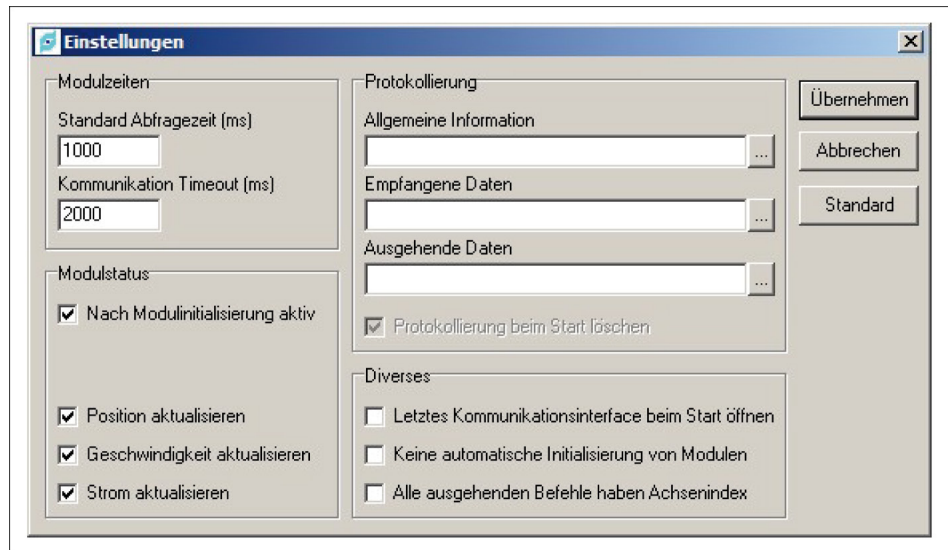
NOTE

"Received" or "outgoing" data can be a great help to you when creating your own applications. The succession of bytes which have to be sent on the respective bus system and the associated response from the module are easy to read off.

All displayed data can be copied to other applications ("Ctrl-C, Ctrl-V"). The individual protocol elements ([👉 1.4.2, Page 14](#)) are **highlighted in color**. **Status bar**

The current communication properties are displayed in the *Status bar* ([👉 4.3, Page 155](#)). Status and settings for the communication interface, number, and ID of the initialized modules.

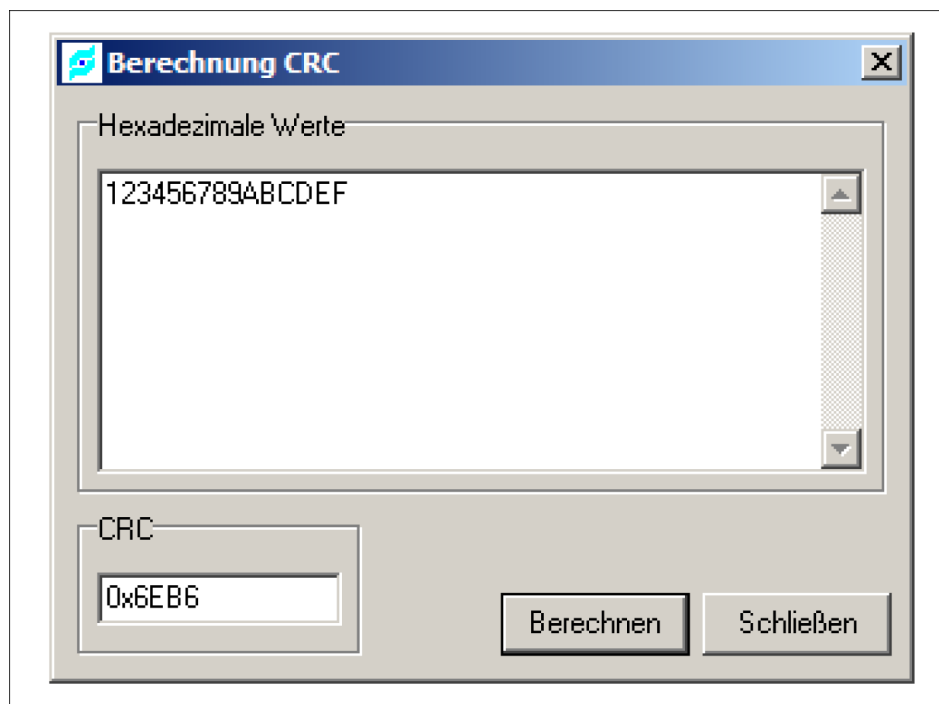
4.3.4 Basic settings



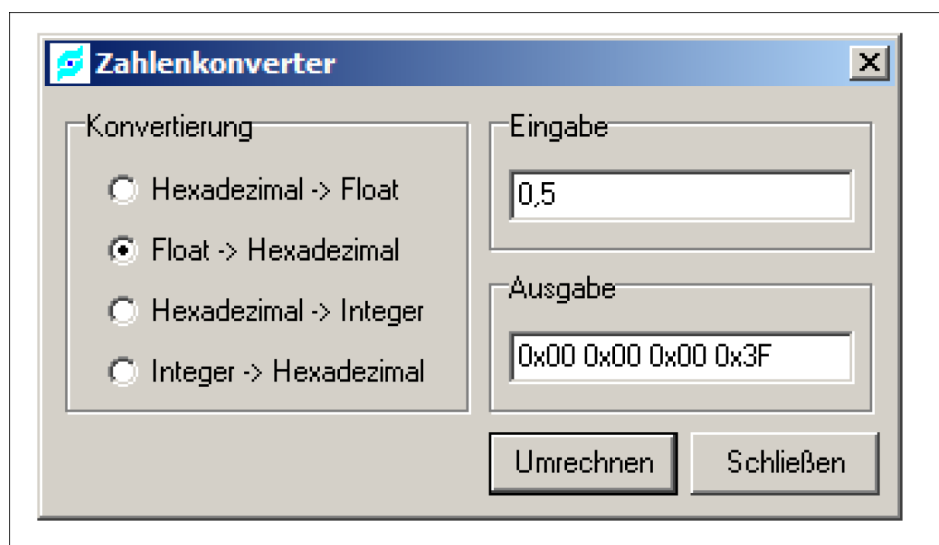
- Module times
 - Standard query time (ms): When initializing a module, a *GET STATE* ([2.6.1, Page 84](#)) is sent with the set cycle time if the box is checked.
 - Communication timeout (ms): Minimum time which the respective interface waits for a response from the module. A "Timeout" is generated if the response is not received and after this period of time has elapsed.
- Module status
 - Active after module initialization: *GET STATE* ([2.6.1, Page 84](#)) is sent automatically after a module has been initialized if the box is checked.
 - Update position: The position is requested with *GET STATE* ([2.6.1, Page 84](#)).
 - Update speed: The speed is requested with *GET STATE* ([2.6.1, Page 84](#)).
 - Update current: The current is requested with *GET STATE* ([2.6.1, Page 84](#)).
- Logging
 - General information: Name of the file for saving the outputs under "General information".
 - Received data: Name of the file for saving the outputs under "Received data".

- Outgoing data: Name of the file for saving the outputs under "Outgoing data".
- Delete logging at start: All previous logging is deleted when the application is restarted.
- Miscellaneous
 - Open last communication interface when starting: The communication setting used last is set during the restart, and communication will be started.
 - No automatic initialization of modules: If a module is found via the bus search, then no module window will be initialized. The module is only initialized manually .
 - All outgoing commands have axis index: All commands sent to the module always have the axis index in the message. This means that "0" is also sent if no axis index is defined.

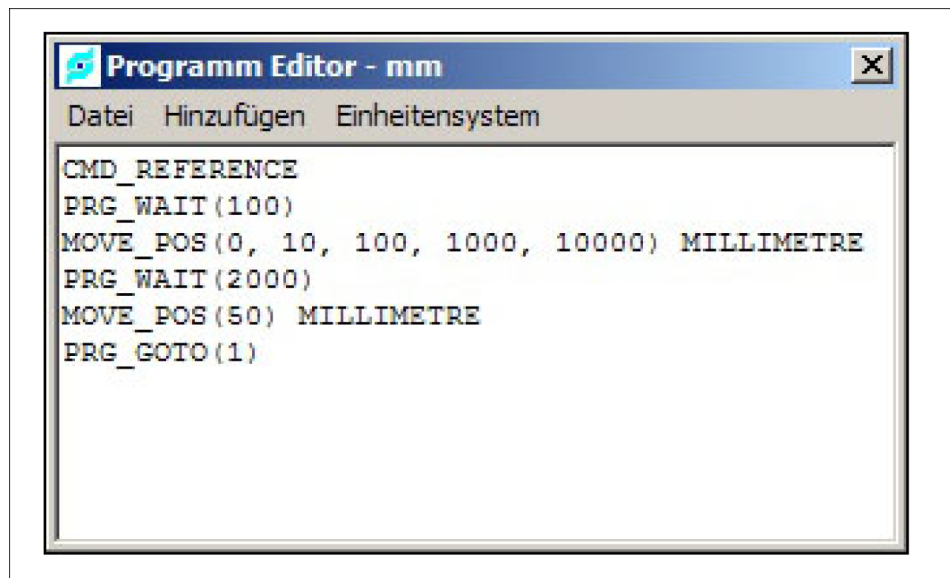
4.3.5 Tools



CRC Calculation



Number Converter



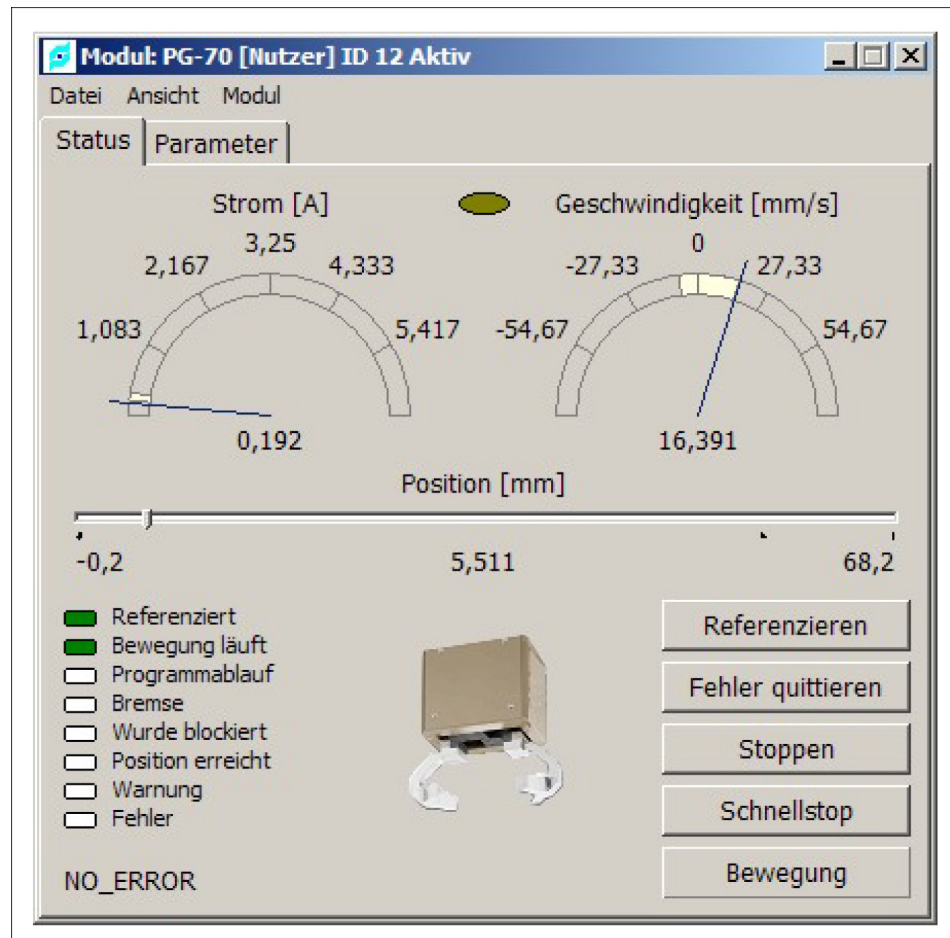
Program Editor

The CRC Calculation ([🔗 4.3.5, Page 161](#)) tool is used to calculate the CRC16 ([🔗 6.2, Page 192](#)) through previously entered hexadecimal numbers.

The Number Converter ([🔗 4.3.5, Page 161](#)) allows you to convert numbers into various formats.

The Program Editor is used to provide support for the internal programming of the modules.

4.4 Module window



Module window

In the module window, all data relevant to the module can be managed and individual commands can be sent to the respective module. The current status of the module can be viewed under the "Status" tab. The status display is updated with every incoming "GET STATE" ([2.6.1, Page 84](#)) message.

NOTE

If status request is deactivated or if this command is deactivated under basic settings ([4.3.4, Page 159](#)), then the display will not be updated. However, when communication takes place via Profibus, the eight indicators on the left-hand side will be updated with every message arriving from the module.

The title line is displayed in the form "Module: <Module type> [<User rights>] ID <Module ID> <Status (active/inactive)>". If a command sent to the module is not answered, then the status display will change from "active" to "inactive".

The switches on the right-hand side can be used to send the most important commands directly to the module. When the "Motion" section has been activated, the window will be extended by the required *parameterization of motion commands*.

Rel. Position		Rel. Position Zeit		Greifen	
Strom	Geschwindigkeit	Position	Position Zeit		
Position [Grad]					
☐ Zyklus	90	Bewegen	Schleife		
Rampenparameter					
☒ Geschwindigkeit [Grad/s]	<input type="range"/>	2400			
☐ Beschleunigung [Grad/s ²]	<input type="range"/>	100000			
☐ Ruck [Grad/s ³]	<input type="range"/>	500000000			
max. zulässiger Strom					
☐ Strom [A]	<input type="range"/>	4,4			

Motion commands

4.4.1 Module states

The following states can be displayed in Status:

- Referenced: The module is referenced.
- Movement running: The module moves at a speed greater than the parameterized *motion threshold* ([↗ 3.2.6, Page 146](#)).
- Program sequence: The internal *program is executed* ([↗ 2.4.4, Page 68](#)).
- Brake: The brake is activated
- Was blocked:

This status *will be activated* ([↗ 2.3.2, Page 55](#)) if:

 - The module is not moving or moves at a speed below the *motion threshold* ([↗ 3.2.6, Page 146](#)), and
 - The target current is reached ($\pm 15\%$), and
 - The *brake timeout* ([↗ 3.2.7, Page 150](#)) has expired.

This status will be deactivated if:

 - A new motion command is triggered, or
 - The target current is reset using the "*SET TARGET CURRENT*" ([↗ 2.2.14, Page 49](#)) command, or
 - The module restarts.
- Position reached: The target position is *reached* ([↗ 2.3.3, Page 56](#)) during a positioning movement.
- Warning: There is a *warning* ([↗ 2.8.1, Page 105](#)).
 - Error: There is an *error* ([↗ 2.8.1, Page 105](#)).

4.4.2 Switch elements

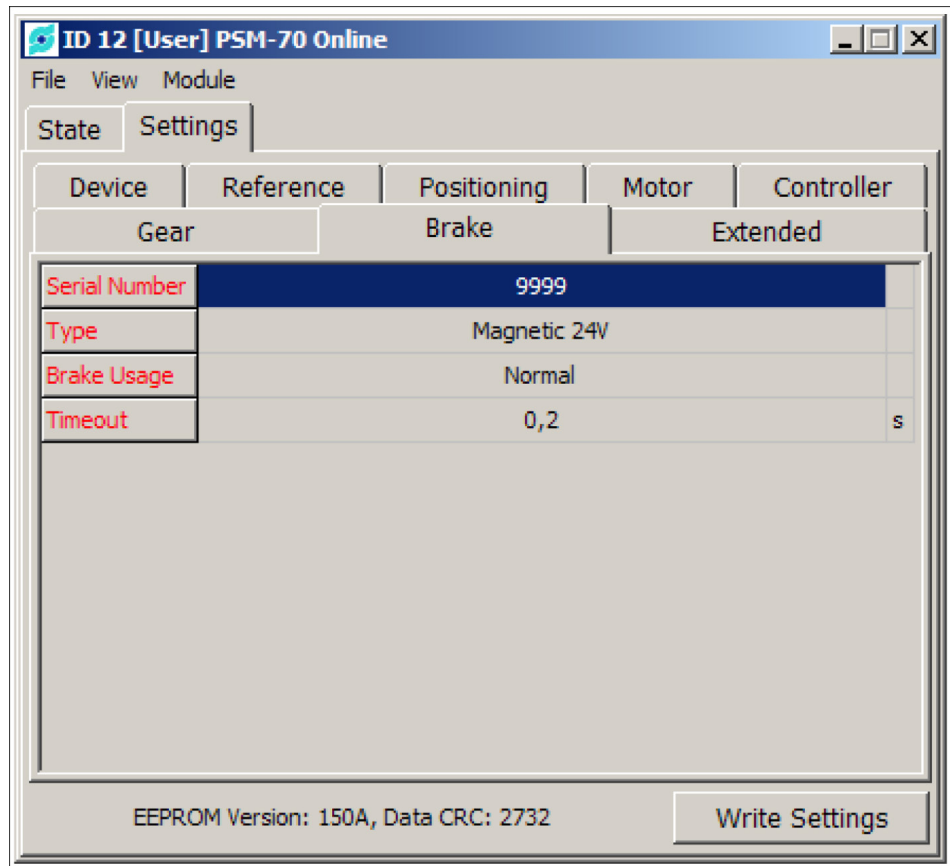
The following commands can be used:

- Reference / Manual reference: A *reference movement* ([↗ 2.2.1, Page 33](#)) will be executed or manual referencing will be started.
- Acknowledge error: A "CMD ACK" ([↗ 2.8.1, Page 105](#)) will be executed.
- Stop: The *Stop* ([↗ 2.2.18, Page 52](#)) command will be sent to the module.
- Fast stop: A *fast stop* ([↗ 2.2.19, Page 53](#)) will be triggered.
- Motion: *The window will be extended by further control elements* (Fig. ([↗ 4.4, Page 163](#))).

Motion commands:

- Speed: A *Geschwindigkeitsfahrt* ([↗ 2.2.9, Page 45](#)) will be carried out.
- Position: A *positioning movement* ([↗ 2.2.3, Page 36](#)) will be carried out. If *Cycle* or *Loop* is activated, then it is possible to move "back and forth" between two freely selectable positions.
- Position time: A time-controlled *positioning movement* ([↗ 2.2.5, Page 40](#)) will be carried out (curved paths possible). If *Cycle* or *Loop* is activated, then it is possible to move "back and forth" between two freely selectable positions.
- Rel. position: A relative *positioning movement* ([↗ 2.2.4, Page 38](#)) will be carried out. If *Cycle* or *Loop* is activated, then it is possible to move "back and forth" between two freely selectable positions.
- Rel. position time: A time-controlled relative *positioning movement* ([↗ 2.2.6, Page 42](#)) will be carried out (curved paths possible). If *Cycle* or *Loop* is activated, then two relative positioning movements will be executed alternately.
- Grip: A *gripping process* ([↗ 2.2.10, Page 46](#)) will be carried out.

4.4.3 Module parameters



EEPROM brake range

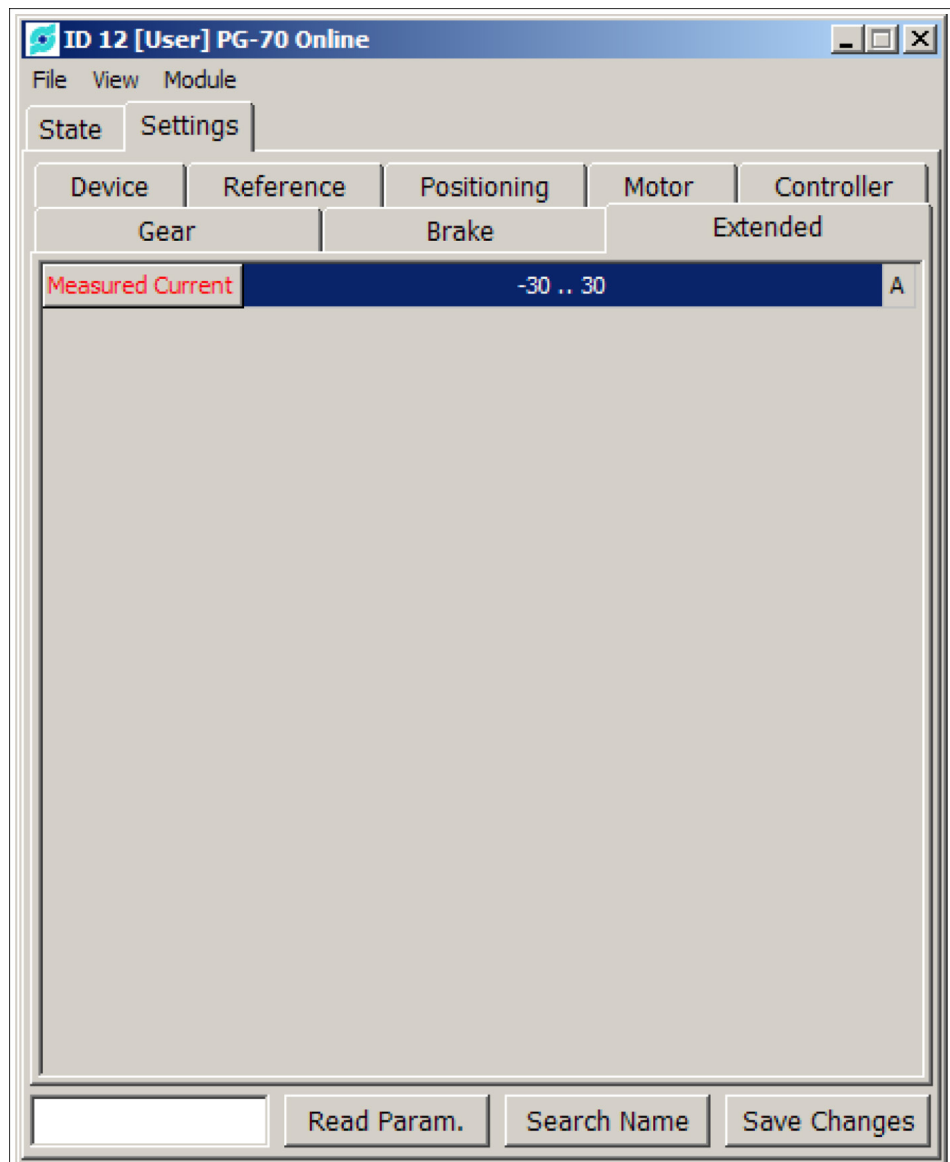
The "Parameter" view is available once the module configuration is received.

The EEPROM entries can be read out and modified under the individual "tabs" (Fig. [4.4.3, Page 167](#)) for the "Parameter" view. Several entries are only enabled for reading depending on the user rights ([1.6, Page 24](#)).

The following tabs are available:

- Device ([3.2.5, Page 135](#))
- Referencing ([3.2.3, Page 123](#))
- Positionierung ([3.2.6, Page 146](#))
- Motor ([3.2.1, Page 117](#))
- Regler ([3.2.4, Page 132](#))
- Getriebe ([3.2.2, Page 123](#))
- Bremse ([3.2.7, Page 150](#))
- Erweitert (Ab Firmware V1.50)

In the "Extended" tab, you can selectively query extended parameters and, allowing for the available rights, change them:



Extended parameterization

- *Read param.* ([👉 2.4.4, Page 68](#)): Loading of parameters based on their parameter ID.
- *Loading all* ([👉 2.4.4, Page 68](#)): Transfer all parameters displayed on the module.
- *Changes* ([👉 2.4.3, Page 64](#)): Saving the changed parameters. All parameters shown in green can be edited. All parameters shown in red cannot be edited.

4.4.4 Menu

The menu can be used for testing all commands available in the module and for making several other settings. Here is a description of the individual menu items:

- File
 - Load EEPROM: Load configuration parameters from a file.
 - Save EEPROM: The displayed configuration parameters will be saved to a file.
 - Print out EEPROM: The displayed configuration parameters will be printed.
 - Print out changes: Only the changed (marked in yellow) configuration parameters will be printed.
 - Deinitialize: The module window will be closed. In the process, the "Fast stop" command will be sent to the relevant module.
- View
 - Up-to-date values: The information last received on the module status will be displayed.
 - Current diagram: The current flow will be shown.
 - Speed diagram: The speed curve will be shown.
 - Position diagram: The dynamics of the change of position will be shown.
 - Combined diagram: The dynamics of the current, speed, and position will be shown.
 - Pause/update diagram: Updating the diagram can be stopped or restarted.
- Module
 - Motion:
 - Current movement: *A current movement will be triggered* ([👉 2.2.8, Page 44](#)).
 - Speed movement: *A speed movement will be triggered* ([👉 2.2.9, Page 45](#)).
 - Positioning movement: *A positioning movement will be triggered* ([👉 2.2.3, Page 36](#)).

- Positioning movement (time): *A time-controlled positioning movement (curved path possible) will be triggered* ([☞ 2.2.5, Page 40](#)).
- Rel. positioning movement: *A relative positioning movement will be triggered* ([☞ 2.2.4, Page 38](#)).
- Rel. positioning movement (time): *A time-controlled relative positioning movement (curved path possible) will be triggered* ([☞ 2.2.6, Page 42](#)).
- Gripping movement: *A gripping movement will be triggered* ([☞ 2.2.10, Page 46](#)).
- Target current: *The target current for the next movement(s) will be set* ([☞ 2.2.14, Page 49](#)).
- Target acceleration: *The target acceleration for the next movement(s) will be set* ([☞ 2.2.12, Page 48](#)).
- Target speed: *The target speed for the next movement(s) will be set* ([☞ 2.2.11, Page 47](#)).
- Target jerk: *The "target jerk" for the next movement(s) will be set* ([☞ 2.2.13, Page 48](#)).
- Target time: *The time for the next positioning command will be set* ([☞ 2.2.15, Page 50](#)).
- Target position: *The position for the next positioning command will be set* ([☞ 2.2.16, Page 50](#)).
- Rel. target position: *The relative position for the next positioning command will be set* ([☞ 2.2.17, Page 51](#)).
- Programming
 - Program to module: *A program can be transferred from a file to a module* ([☞ 2.5.1, Page 71](#)).
 - Program from module: *A program saved in the module can be read out and saved to a file* ([☞ 2.5.2, Page 72](#)).
 - Delete program: *The program memory will be emptied.*
 - Display program: *A program available in the module will be displayed* ([☞ 2.5.2, Page 72](#)).
 - Execute program: *A program will be executed* ([☞ 2.5.3, Page 72](#)).
 - Block selection: *A program block will be executed* ([☞ 2.5.4, Page 73](#)).
 - Execute block 0: *Program block number "0" will be executed* ([☞ 2.5.5, Page 73](#)).

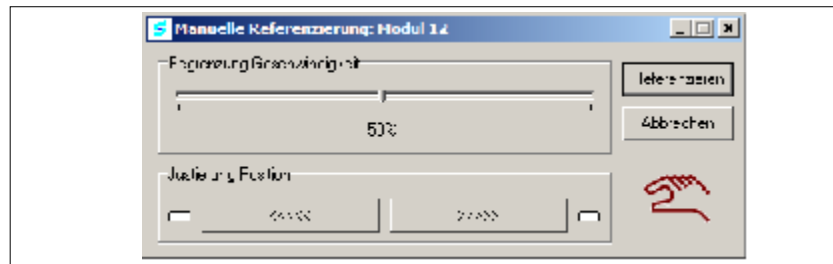
- Execute block 1: *Program block number "1REPLACEME" will be executed* ([☞ 2.5.6, Page 74](#)).
- Execute block 2: *Program block number "2" will be executed* ([☞ 2.5.7, Page 75](#)).
- Execute block 3: *Program block number "3" will be executed* ([☞ 2.5.8, Page 75](#)).
- Execute block 4: *Program block number "4" will be executed* ([☞ 2.5.9, Page 76](#)).
- Execute block 5: *Program block number "5" will be executed* ([☞ 2.5.10, Page 77](#)).
- Execute block 2: *Program block number "2" will be executed* ([☞ 2.5.11, Page 77](#)).
- Execute block 7: *Program block number "7" will be executed* ([☞ 2.5.12, Page 78](#)).
- Execute block 8: *Program block number "8" will be executed* ([☞ 2.5.13, Page 78](#)).
- Execute block 9: *Program block number "9" will be executed* ([☞ 2.5.14, Page 79](#)).
- Execute block 10: *Program block number "10" will be executed* ([☞ 2.5.15, Page 80](#)).
- Execute block 11: *Program block number "11" will be executed* ([☞ 2.5.16, Page 80](#)).
- Execute block 12: *Program block number "12" will be executed* ([☞ 2.5.17, Page 81](#)).
- Execute block 13: *Program block number "13" will be executed* ([☞ 2.5.18, Page 81](#)).
- Execute block 14: *Program block number "14" will be executed* ([☞ 2.5.19, Page 82](#)).
- Execute block 15: *Program block number "15" will be executed* ([☞ 2.5.20, Page 82](#)).
- Communication tests
 - From module: All values: *Transfer test data from module to control unit* ([☞ 2.6.11, Page 98](#)).
 - From module: -1.2345: *Transfer test data from module to control unit* ([☞ 2.6.11, Page 98](#)).
 - From module: 47.11: *Transfer test data from module to control unit* ([☞ 2.6.11, Page 98](#)).
 - From module: 287454020: *Transfer test data from module to control unit* ([☞ 2.6.11, Page 98](#)).

- From module: -1122868: *Transfer test data from module to control unit* ([☞ 2.6.11, Page 98](#)).
- From module: 512: *Transfer test data from module to control unit* ([☞ 2.6.11, Page 98](#)).
- From module: -20482: *Transfer test data from module to control unit* ([☞ 2.6.11, Page 98](#)).
- To module: All values: *Transfer test data from control unit to module* ([☞ 2.6.12, Page 100](#)).
- To module: -1.2345: *Transfer test data from control unit to module* ([☞ 2.6.12, Page 100](#)).
- To module: 47.11: *Transfer test data from control unit to module* ([☞ 2.6.12, Page 100](#)).
- To module: 287454020: *Transfer test data from control unit to module* ([☞ 2.6.12, Page 100](#)).
- To module: -1122868: *Transfer test data from control unit to module* ([☞ 2.6.12, Page 100](#)).
- To module: 512: *Transfer test data from control unit to module* ([☞ 2.6.12, Page 100](#)).
- To module: -20482: *Transfer test data from control unit to module* ([☞ 2.6.12, Page 100](#)).
- Read EEPROM
 - Read EEPROM: *The complete configuration will be read by the module* ([☞ 2.4.2, Page 61](#)).
 - Read ID: *The module ID will be read* ([☞ 2.4.2, Page 61](#)).
 - Read group: *The group identifier will be read* ([☞ 2.4.2, Page 61](#)).
 - Read communication: *The currently set communication mode will be read* ([☞ 2.4.2, Page 61](#)).
 - Read unit system: *The currently set unit system will be read* ([☞ 2.4.2, Page 61](#)).
 - Read serial baudrate: *The currently set serial baudrate will be read* ([☞ 2.4.2, Page 61](#)).
 - Read CAN baudrate: *The currently set CAN baudrate will be read* ([☞ 2.4.2, Page 61](#)).
 - Read max. software limit range: *The currently set upper software limit range will be read* ([☞ 2.4.2, Page 61](#)).
 - Read min. software limit range: *The currently set lower software limit range will be read* ([☞ 2.4.2, Page 61](#)).

- Read nom. current: *The set nominal current will be read* ([☞ 2.4.2, Page 61](#)).
- Read max. current: *The set maximum current will be read* ([☞ 2.4.2, Page 61](#)).
- Read max. speed: *The maximum possible speed will be read out* ([☞ 2.4.2, Page 61](#)).
- Read max. acceleration: *The maximum possible acceleration will be read out* ([☞ 2.4.2, Page 61](#)).
- Read max. jerk: *The maximum possible jerk will be read out* ([☞ 2.4.2, Page 61](#)).
- Read offset phase A: *The offset for current phase A will be read out* ([☞ 2.4.2, Page 61](#)).
- Read offset phase B: *The offset for current phase B will be read out* ([☞ 2.4.2, Page 61](#)).
- Read reference offset: *The offset for referencing will be read out* ([☞ 2.4.2, Page 61](#)).
- Read serial number: *The serial number for the device will be read out* ([☞ 2.4.2, Page 61](#)).
- Read CRC data: *The checksum for the configuration parameters will be read out* ([☞ 2.4.2, Page 61](#)).
- Read order number: *The order number will be read out* ([☞ 2.4.2, Page 61](#)).
- Read board Version: *The version of the module board is read* ([☞ 2.4.2, Page 61](#)).
- Read board Date: *The date of manufacture of the module board is read* ([☞ 2.4.2, Page 61](#)).
- Read extended parameters: *Extended parameters will be read individually* ([☞ 2.4.4, Page 68](#)).
- Write EEPROM
 - Write EEPROM: *The complete configuration will be written* ([☞ 2.4.1, Page 59](#)). Only data enabled for the active user ([☞ 1.6, Page 24](#)) will be written.
 - Write ID: *The module ID will be changed* ([☞ 2.4.1, Page 59](#)). A restart is required.
 - Write group: *The group number will be changed* ([☞ 2.4.1, Page 59](#)). A restart is required.
 - Write communication: *The communication interface will be changed* ([☞ 2.4.1, Page 59](#)). A restart is required.

- Write unit system: The unit system will be converted ([☞ 2.4.1, Page 59](#)). A restart is required.
- Write serial baudrate: The baudrate for serial communication will be changed ([☞ 2.4.1, Page 59](#)). A restart is required.
- Write CAN baudrate: The CAN baudrate will be changed ([☞ 2.4.1, Page 59](#)). A restart is required.
- Write extended parameters: Extended parameters will be written individually ([☞ 2.4.3, Page 64](#)).
- Reference / Start up Nonius: A referencing run will be triggered or Nonius will be started up if the module is manually referenced ([☞ 2.2.1, Page 33](#)).
- Acknowledge error: An error will be acknowledged ([☞ 2.8.1, Page 105](#)).
- Stop: A movement will be stopped ([☞ 2.2.18, Page 52](#)).
- Fast stop: A fast stop will be triggered ([☞ 2.2.19, Page 53](#)).
- REPLACEMEChange user: The current user will be changed ([☞ 2.6.10, Page 98](#)). Password required.
- Query status: The current status of the module will be requested ([☞ 2.6.1, Page 84](#)).
- Error information: Further information about the active error will be read out ([☞ 2.8.1, Page 105](#)).
- Module information: Information (order no., software/hardware version, device type) will be read out ([☞ 2.4.2, Page 61](#)).
- Manual reference: The module will be manually referenced ([☞ 4.4.5, Page 175](#)).
- Digital inputs/outputs: The digital inputs/outputs will be activated or read ([☞ 2.6.4, Page 91](#)).
- Restart: The module will be restarted ([☞ 2.6.3, Page 90](#)).
- Switch off: The module will be disconnected from the bus system ([☞ 2.6.9, Page 97](#)). A restart of the module is required to be able to continue to operate the module.
- Spontaneous message on/off: Spontaneous messages will be switched on or switched off ([☞ 2.3.7, Page 58](#)).
- Start-up assistant: The module will be started up ([☞ 4.4.6, Page 175](#)).
- Update firmware: New firmware can be loaded onto the module. Password required.

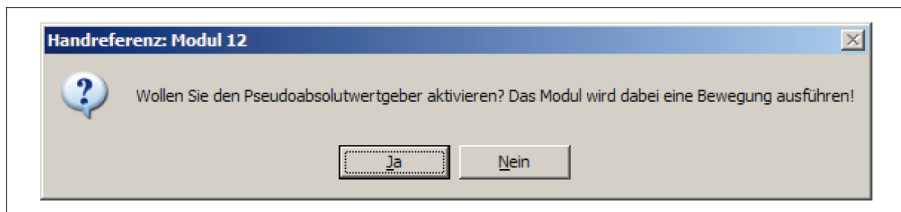
4.4.5 Manual referencing



Manual referencing

If the module is switched to the manual referencing mode ("CMD REFERENCE HAND" ([↗ 2.2.2, Page 34](#)), then it is possible to adjust the module to the required *reference position* ([↗ 3.2.3, Page 123](#)). The maximum speed during the adjustment move (100%) can be set via the maximum *reference speed* ([↗ 3.2.3, Page 123](#)) parameter.

If a module is referenced again using the position system ([↗ 3.2.6, Page 146](#)) "Encoder with index track", which was already manually referenced, then the module will restart after this function has been exited.



Activation of pseudo-absolute encoder

NOTE

If the pseudo-absolute encoder ([↗ 1.7, Page 27](#)) was not activated, then the module is not referenced after the restart. If the dialog box does not appear, then the pseudo-absolute encoder is already activated, or an activation is not possible.

4.4.6 Start-up assistant

With the assistant, important configuration parameters are adapted step-by-step and ultimately transferred to the module. The tool is designed mainly for the start-up of PSM and PDU modules which are attached to a linear axis.

4.5 Tips

4.5.1 Supported languages

English and German are supported right now. The language can be changed using the "Settings/Language" menu. The selected language applies after the application has been restarted.

4.5.2 Vector CAN driver

If the application does not find the required library file "vcand32.dll", then please observe:

- Either copy the above named DLL file from the installation directory of Vector CAN into the installation directory of "Motion Tool Schunk",
- or add the directory where the DLL file is located into the Windows PATH variable,
- or copy the DLL file to the Windows system directory "[Windows-Installation]/system32".

4.5.3 Peak CAN driver

If the application does not find the required library file "vcand32.dll", then please observe:

- Either copy the above named DLL file from the installation directory of Vector CAN into the installation directory of "Motion Tool Schunk",
- or add the directory where the DLL file is located into the Windows PATH variable,
- or copy the DLL file to the Windows system directory "[Windows-Installation]/system32".

4.5.4 Softing CAN driver

If the application does not find the required library file "vcand32.dll", then please observe:

- Either copy the above named DLL file from the installation directory of Vector CAN into the installation directory of "Motion Tool Schunk",

- or add the directory where the DLL file is located into the Windows PATH variable,
- or copy the DLL file to the Windows system directory "[Windows-Installation]/system32".

4.5.5 ESD CAN interface

When the interface is searched for active modules without having switched on at least one module in the process, the card switches into the Bus off status and remains this way for a certain period of time.

- Switch on at least one module which is ready for CAN communication.
- Wait a few seconds.
- If no module is signaled, then search through the bus.

4.5.6 Siemens Profibus interface

Communication using Siemens Profibus cards in "Motion Tool Schunk" is relatively slow.

If "Motion Tool Schunk" finds no SMP modules on the bus, despite some existing, then these modules have to be restarted.

4.5.7 Have the module status displayed automatically

In the "Settings" window, activate the "Active after module initialization" option and set the "Standard query time" option to a positive number (approx. 1000 ms as standard).

4.5.8 Open the communication interface when starting

With a permanently available communication interface, it is recommended that the interface used last be opened automatically when starting the application. To do this, activate the "Use last communication interface" option in the "Settings" window.

4.5.9 Data throughput with CAN

The shortest time interval for the *GET STATE* ([🔗 2.6.1, Page 84](#)) command is reached using the CAN interface. This interval is feasible up to 1 ms cycle time. This means that the application can display incoming messages every millisecond without causing an overflow in the internal message buffer.

The shortest time interval for the *GET STATE GET STATE (0x95)* command is reached using the CAN interface. This interval is feasible up to 1 ms cycle time. This means that the application can display incoming messages every millisecond without causing an overflow in the internal message buffer.

4.5.10 Configured modules under Profibus

In order to view a list of modules configured for Profibus, you can execute the entry "Display bus details" in the "View" menu in the main window.

4.5.11 Frequent timeouts in serial communication

Since the serial interface is not a bus system, data collisions may sometimes occur sporadically if several modules are communicating at the same time. The application always attempts to handle these data collisions (commands are repeated) so that communication continues to run without interruption.

The probability of timeouts occurring (thus no response from the module) can be reduced by:

- Increasing the time interval for the *GET STATE* ([🔗 2.6.1, Page 84](#)) command or switching off the status information if it is not being used,
- or/and by increasing the baudrate for the serial interface,
- or/and by slightly increasing the communication timeout in the main window under "Settings",
- or/and by initializing or simultaneously controlling as few modules as possible.
- Increasing the time interval for the *GET STATE GET STATE (0x95)* command or switching off the status information if it is not being used,
- or/and by increasing the baudrate for the serial interface,

- or/and by slightly increasing the communication timeout in the main window under "Settings",
- or/and by initializing or simultaneously controlling as few modules as possible.

4.5.12 Change individual EEPROM parameters

If just individual EEPROM parameters such as the module ID or the communication interface are to be changed, then it is advisable to execute the corresponding *SET CONFIG* ([🔗 2.4.1, Page 59](#)) commands. These can be found in the module window under the "Write module/EEPROM /Write ..." menu.

4.5.13 Do not maximize the main window

The main window should not be maximized since module windows already initialized will remain hidden. However, it is possible to find a hidden module window again using the relevant "View/Module ..." menu option.

4.5.14 "Auto" communication type for the module

If, in a module, the communication type is set to "Auto", the following points must be observed:

- After being switched on, the module does not signal until it detects a communication interface as being active.
- Potentially, Profibus will immediately be detected as being active (providing there is an active host).
- Make sure that there is only one, single physical interface for the communication, otherwise the module would, under certain circumstances, select the "wrong" communication.

4.5.15 No automatic initialization

The "No automatic initialization of modules" option under *Basic settings* ([🔗 4.3.4, Page 159](#)) in the application is used so that the modules found are not automatically initialized. At the same time, no module windows are automatically opened.

This setting is particularly useful if many modules communicating with the serial interface are compulsorily initialized at the same time, which may lead to not all modules being able to be fully initialized.

5 Remarks

5.1 Modules

Description of connections to the module	Since there are a wide variety of <i>Schunk</i> modules with different connection assignments, please refer to the supplied documentation for your module for more on this.
The module cannot reference from every position	The maximum permitted reference current is not enough to move the module. Increase the <i>max. reference current</i> (👉 3.2.3, Page 123).

5.2 Protocol

Fragmentation is not possible	The entire module can be fully operated without data fragmentation being necessary. This applies to all supported communication interfaces.
--------------------------------------	---

5.3 Serial communication

Data collisions are occurring	• Connect up fewer modules to the "serial bus" • Switch off spontaneous messages <i>CMD TOGGLE IMPULSE MESSAGE</i> (👉 2.3.7, Page 58) • Use the "Serial communication without spontaneous message" interface.
Problems arise when several modules are combined	The serial interface is not designed as a bus system. If you need to operate several modules using one interface, then you should switch to a "real" bus system (CAN, Profibus). If too many participants are connected up, this may frequently lead to <i>data collisions</i> (👉 5.3, Page 180). If the serial interface is used as a bus system, only baudrates up to 19200 [kBaud] will be supported.
Which baudrates are supported by the module in serial communication?	1200, 2400, 4800, 9600, 19200, 38400 [kBaud].

5.4 CAN

Which CAN baudrates are supported by the module?	50, 100, 125, 250, 500, and 1000 [kBit/s]
---	---

5.5 Profibus

Is SSA (set slave address) supported?	Yes
Data cannot be transmitted consistently	1 Set D-Len (first byte) to 0x00, populate all data and, finally, set D-Len to the required value. 2 Use the SYNC, FREEZE mechanism.

6 Appendix

6.1 Selected examples

6.1.1 Serial communication

6.1.1.1 CHECK MC PC COMMUNICATION (float)

Check communication from module to control unit using test data.

- 1 *Request test data* ([↗ 2.6.11, Page 98](#)) from the module (float value -1.2345).
- 2 Test data are sent from the module.

	Id	D-Len	Cmd	Data	CRC16
M->S	0x05 0x01	0x03	0xE4	0x01 0x01	0xBD 0xB6
S->M	0x07 0x01	0x07	0xE4	0x19 0x04 0x9E 0xBF 0x01 0x01	0x74 0x37

6.1.1.2 CHECK PC MC COMMUNICATION

Check communication from control unit to module using test data.

- 1 *Send test data* ([↗ 2.6.11, Page 98](#)) to the module.
- 2 The module reports that the test data have been received and reports information on which data were interpreted erroneously (all data OK)

	Id	D-Len	Cmd	Data	CRC16
M->S	0x05 0x01	0x15	0xE5	0x19 0x04 0x9E 0xBF 0xA4 0x70 0x3C 0x42 0x44 0x33 0x22 0x11 0xCC 0xDD 0xEE 0xFF 0x00 0x02 0xFE 0xAF	0x89 0xD7
S->M	0x07 0x01	0x04	0xE5	0x4F 0x4B 0x00	0xB6 0xFA

6.1.1.3 Troubleshooting

An error ("ERROR MOTOR VOLTAGE LOW") ([↗ 2.8.2, Page 107](#)) has occurred.

- 1 Interpret and remove error. Power supply to motor switched off => Switch on power supply again
- 2 Acknowledge error "CMD ACK" ([🔗 2.8.1, Page 105](#))
- 3 "CMD ACK" command is confirmed
- 4 Info message appears "No other errors are present".

	Id	D-Len	Cmd	Data	CRC16
Eve- ry 15[s]	0x03 0x01	0x02	0x88	0x74	0x82 0x1B
M- >S	0x05 0x01	0x01	0x8B		0x10 0xFB
S- >M	0x07 0x01	0x03	0x8B	0x4F 0x4B	0x38 0x1E
Spon- tane- ous	0x07 0x01	0x03	0x8A	0x08 0x00	0x1A 0x19

6.1.1.4 GET STATE 1[s]

The current position value is to be transferred every second.

- 1 Send *Get State* ([🔗 2.6.1, Page 84](#)) 1[s] command, only position.
- 2 A message for the position with current status. (Position 1.011[mm], in motion, no error)
- 3 Further messages cyclically every second.
- 4 Message for the position with current status. (Position 5.054[mm], in motion, no error)

	Id	D-Len	Cmd	Data	CRC16
M- >S	0x05 0x01	0x06	0x95	0x00 0x00 0x80 0x3F 0x01	0x54 0x41
S- >M	0x07 0x01	0x07	0x95	0x36 0x89 0x81 0x3F 0x02 0x00	0xF9 0xBC
S- >M	0x07 0x01	0x07	0x95		
S- >M	0x07 0x01	0x07	0x95	0x76 0xBE 0xA1 0x40 0x02 0x00	0x38 0xA0

6.1.1.5 MOVE POS 10[mm]

The standard command for positioning.

- 1 Send move to *position* ([2.2.3, Page 36](#)) 10[mm] command
- 2 "Will reach position in 3.358[s]" message. The movement is started.
- 3 After a certain period of time: "Position 9.9969[mm] reached"

	Id	D-Len	Cmd	Data	CRC16
M->S	0x05 0x01	0x05	0xB0	0x00 0x00 0x20 0x41	0x48 0x80
S->M	0x07 0x01	0x05	0xB0	0xEE 0xEE 0x56 0x40	0x7B 0xE4
Spontaneous	0x07 0x01	0x05	0x94	0xB6 0xF3 0x1F 0x41	0x7E 0xD5

6.1.1.6 Referencing

- 1 Send the *Reference* ([2.2.1, Page 33](#)) command
- 2 Referencing command understood
- 3 After a certain period of time: "Stop at position 5.792[mm]"

	Id	D-Len	Cmd	Data	CRC16
M->S	0x05 0x01	0x01	0x92		0xD1 0x31
S->M	0x07 0x01	0x03	0x92	0x4F 0x4B	0xE9 0xD9
Spontaneous	0x07 0x01	0x05	0x93	0x21 0x56 0xB9 0x40	0x4D 0x22

6.1.2 CAN

6.1.2.1 CHECK MC PC COMMUNICATION (float)

Check communication from module to control unit using *test data* ([2.6.11, Page 98](#)).

- 1 Request *test data* ([2.6.11, Page 98](#)) from the module (float value -1.2345).

⇒ Test data are sent from the module.

	Id	D-Len	Cmd	Data	
M->S	0x501	0x04	0x03	0xE4	0x01 0x01
S->M	0x701	0x08	0x07	0xE4	0x19 0x04 0x9E 0xBF 0x01 0x01

6.1.2.2 CHECK PC MC COMMUNICATION

Check communication from control unit to module using *test data* ([2.6.11, Page 98](#)). Fragmentation is required.

NOTE

Data fragmentation is not absolutely necessary for operating and/or checking the module

- 1 Send first fragment of test data to the module.
- 2 Send second fragment of test data to the module.
- 3 Send third fragment of test data to the module.
- 4 Send last fragment of test data to the module.

⇒ The module reports that the test data have been received and reports information on which data were interpreted erroneously (all data OK).

	Id	DLC	D-Len	Cmd	Data
M->S	0x501	0x08	0x15	0x84 0xE5	0x19 0x04 0x9B 0xBF 0xA4
M->S	0x501	0x08	0x0F	0x85	0x70 0x3C 0x42 0x44 0x33 0x22

	Id	DLC	D-Len	Cmd	Data
M->S	0x501	0x08	0x09	0x85	0x11 0xCC 0xDD 0xEE 0xFF 0x00
M->S	0x501	0x05	0x03	0x86	0x02 0xFE 0xAF
S->M	0x701	0x05	0xE4	0x4F 0x4B 0x00	

6.1.2.3 Troubleshooting

An error ("*ERROR MOTOR VOLTAGE LOW*") ([👉 2.8.2, Page 107](#)) has occurred.

Interpret and remove error. Power supply to motor switched off
=> Switch on power supply again.

- 1 Acknowledge error "CMD ACK" ([👉 2.8.1, Page 105](#))
 - 2 "CMD ACK" command is confirmed
- ⇒ Info message appears "No other errors are present".

	Id	DLC	D-Len	Cmd	Data
Every 15[s]	0x301	0x03	0x02	0x88	0x74
M->S	0x501	0x02	0x01	0x8B	
S->M	0x701	0x04	0x03	0x8B	0x4F 0x4B
Spontaneous	0x701	0x04	0x03	0x8A	0x08 0x00

6.1.2.4 GET STATE 1[s]

The current position value is to be transferred every second.

- 1 Send *Get State* ([👉 2.6.1, Page 84](#)) 1[s] command, only position.
- 2 A message for the position with current status. (Position 1.011[mm], in motion, no error)
- 3 Further messages cyclically every second.
- 4 Message for the position with current status. (Position 5.054[mm], in motion, no error)

	Id	DLC	D-Len	Cmd	Data
M->S	0x501	0x07	0x06	0x95	0x00 0x00 0x80 0x3F 0x01
S->M	0x701	0x08	0x07	0x95	0x36 0x89 0x81 0x3F 0x02 0x00
S->M	0x701	0x08	0x07	0x95	
S->M	0x701	0x08	0x05	0x94	0x76 0xBE 0xA1 0x40 0x02 0x00

6.1.2.5 MOVE POS 10[mm]

The standard command for *positioning* ([↗ 2.2.3, Page 36](#)).

- 1 Send move to position 10[mm] command
- 2 "Will reach position in 3.358[s]" message. The movement is started.
- 3 After a certain period of time: "Position 9.9969[mm] reached"

	Id	DLC	D-Len	Cmd	Data
M->S	0x501	0x06	0x05	0xB0	0x00 0x00 0x20 0x41
S->M	0x701	0x06	0x05	0xB0	0xEE 0xEE 0x56 0x40
Spontaneous	0x701	0x06	0x05	0x94	0xB6 0xF3 0x1F 0x41

6.1.2.6 Referencing

- 1 Send the *Reference* ([↗ 2.2.1, Page 33](#)) command
- 2 Referencing command understood
- 3 After a certain period of time: "Stop at position 5.792[mm]"

	Id	DLC	D-Len	Cmd	Data
M->S	0x501	0x02	0x01	0x92	
S->M	0x701	0x04	0x03	0x92	0x4F 0x4B
Spontaneous	0x701	0x06	0x05	0x93	0x21 0x56 0xB9 0x40

6.1.3 Profibus

6.1.3.1 CHECK MC PC COMMUNICATION (float)

Check communication from module to control unit using *test data* ([☞ 2.6.11, Page 98](#)).

- 1 Request test data ([☞ 2.6.11, Page 98](#)) from the module (float value -1.2345).
- 2 Test data are sent from the module. The "*MsgCount*" ([☞ 1.4.5, Page 18](#)) is increased by one.

	D-Len	Cmd	Data	State/MsgCount
M->S	0x03	0xE4	0x01 0x01 0x?? 0x?? 0x?? 0x??	
S->M	0x07	0xE4	0x19 0x04 0x9E 0xBF 0x01 0x01 0x?? 0x?? 0x?? 0x?? 0x?? 0x??	0x20 0x05

6.1.3.2 CHECK PC MC COMMUNICATION

Check communication from control unit to module using *test data* ([☞ 2.6.11, Page 98](#)). Fragmentation is required.

NOTE

Data fragmentation is not absolutely necessary for operating and/or checking the module.

- 1 Send first fragment of test data to the module.
 - 2 Wait for *confirmation* ([☞ 2.7.1, Page 102](#)).
 - 3 Send second fragment of test data to the module.
 - 4 Wait for *confirmation* ([☞ 2.7.1, Page 102](#)).
 - 5 Send third fragment of test data to the module.
 - 6 Wait for *confirmation* ([☞ 2.7.1, Page 102](#)).
 - 7 Send last fragment of test data to the module.
- ⇒ The module reports that the test data have been received and reports information on which data were interpreted erroneously (all data OK). The "*MsgCount*" ([☞ 1.4.5, Page 18](#)) is increased by one.

	D-Len	Cmd	Data	State/MsgCount
M->S	0x15	0x84	0xE5 0x19 0x04 0x9E 0xBF 0xA4	
S->M	0x02	0x87	0x15 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x??	0x20 0x06
M->S	0x0F	0x85	0x70 0x3C 0x42 0x44 0x33 0x22	
S->M	0x02	0x87	0x0F 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x??	0x20 0x06
M->S	0x09	0x85	0x11 0xCC 0xDD 0xEE 0xFF 0x00	
S->M	0x02	0x87	0x09 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x??	0x20 0x06
M->S	0x03	0x86	0x02 0xFE 0xAF 0x?? 0x?? 0x??	
S->M	0x04	0xE5	0x4F 0x4B 0x00 0x?? 0x?? 0x?? 0x?? 0x?? 0x??	0x20 0x06

6.1.3.3 Troubleshooting

An error ("*ERROR MOTOR VOLTAGE LOW*") ([👉 2.8.2, Page 107](#)) has occurred.

- 1 Interpret error (extended diagnosis is supported) and remove. Switch off power supply to motor => Switch on power supply again.
 - 2 Acknowledge error "*CMD ACK*" ([👉 2.8.1, Page 105](#)).
 - 3 "*CMD ACK*" command is confirmed. The "*MsgCount*" ([👉 1.4.5, Page 18](#)) is increased by one.
- ⇒ Info message appears "No other errors are present".

	D-Len	Cmd	Data	State/MsgCount
S->M	0x02	0x88	0x74 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x??	0x30 0x03
M->S	0x01	0x8B	0x?? 0x?? 0x?? 0x?? 0x?? 0x??	
S->M	0x03	0x8B	0x4F 0x4B 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x??	0x20 0x04
Spontaneous	0x03	0x8A	0x08 0x00 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x??	0x20 0x04

6.1.3.4 GET STATE 1[s]

The current position value is to be transferred every second.

- 1 Send *Get State* ([↗ 2.6.1, Page 84](#)) 1[s] command, only position.
- 2 A message for the position with current status. (Position 1.011[mm], in motion, no error) The "*MsgCount*" ([↗ 1.4.5, Page 18](#)) is increased by one.
- 3 Further messages cyclically every second.
- 4 Message for the position with current status. (Position 5.054[mm], in motion, no error)

	D-Len	Cmd	Data	State/MsgCount
M->S	0x07	0x95	0x00 0x00 0x80 0x3F 0x01 0x??	
S->M	0x08	0x95	0x36 0x89 0x81 0x3F 0x02 0x00 0x?? 0x?? 0x?? 0x?? 0x?? 0x??	0x02 0x03
S->M	0x08	0x95		0x02 0x03
S->M	0x08	0x95	0x76 0xBE 0xA1 0x40 0x02 0x00 0x?? 0x?? 0x?? 0x?? 0x?? 0x??	0x02 0x03

6.1.3.5 MOVE POS 10[mm]

The standard command for *positioning* ([↗ 2.2.3, Page 36](#)).

- 1 Send move to position 10[mm] command
- 2 "Will reach position in 3.358[s]" message. The movement is started. The "MsgCount" ([👉 1.4.5, Page 18](#)) is increased by one.
- 3 After a certain period of time: "Position 9.9969[mm] reached"

	D-Len	Cmd	Data	State/MsgCount
M->S	0x05	0xB0	0x00 0x00 0x20 0x41 0x?? 0x??	
S->M	0x05	0xB0	0xEE 0xEE 0x56 0x40 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x??	0x01 0x02
Spontaneous	0x05	0x94	0xB6 0xF3 0x1F 0x41 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x??	0x61 0x02

6.1.3.6 Referencing

- 1 Send the *Reference* ([👉 2.2.1, Page 33](#)) command
- 2 Referencing command understood. The "MsgCount" 15 is increased by one.
- 3 After a certain period of time: "Stop at position 5.792[mm]"

	D-Len	Cmd	Data	State/MsgCount
M->S	0x01	0x92	0x?? 0x?? 0x?? 0x?? 0x?? 0x??	
S->M	0x03	0x92	0x4F 0x4B 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x?? 0x??	0x00 0x01
Spontaneous	0x05	0x93	0x21 0x56 0xB9 0x40	0x61 0x01

6.2 CRC16 calculation for serial communication

The CRC16 algorithm shown here is available in this form in the firmware. Further algorithms can be found on the internet which also carry out a CRC16 calculation and do without a table.

```

UInt16 CRC16(UInt16 crc, UInt8 data)
{
const UInt16 tbl[256] = {
0x0000, 0xC0C1, 0xC181, 0x0140, 0xC301, 0x03C0, 0x0280,
0xC241,
0xC601, 0x06C0, 0x0780, 0xC741, 0x0500, 0xC5C1, 0xC481,
0x0440,
0xCC01, 0x0CC0, 0x0D80, 0xCD41, 0x0F00, 0xCFC1, 0xCE81,
0x0E40,
0x0A00, 0xCAC1, 0xCB81, 0x0B40, 0xC901, 0x09C0, 0x0880,
0xC841,
0xD801, 0x18C0, 0x1980, 0xD941, 0x1B00, 0xDBC1, 0xDA81,
0x1A40,
0x1E00, 0xDEC1, 0xDF81, 0x1F40, 0xDD01, 0x1DC0, 0x1C80,
0xDC41,
0x1400, 0xD4C1, 0xD581, 0x1540, 0xD701, 0x17C0, 0x1680,
0xD641,
0xD201, 0x12C0, 0x1380, 0xD341, 0x1100, 0xD1C1, 0xD081,
0x1040,
0xF001, 0x30C0, 0x3180, 0xF141, 0x3300, 0xF3C1, 0xF281,
0x3240,
0x3600, 0xF6C1, 0xF781, 0x3740, 0xF501, 0x35C0, 0x3480,
0xF441,
0x3C00, 0xFCC1, 0xFD81, 0x3D40, 0xFF01, 0x3FC0, 0x3E80,
0xFE41,
0xFA01, 0x3AC0, 0x3B80, 0xFB41, 0x3900, 0xF9C1, 0xF881,
0x3840,
0x2800, 0xE8C1, 0xE981, 0x2940, 0xEB01, 0x2BC0, 0x2A80,
0xEA41,
0xEE01, 0x2EC0, 0x2F80, 0xEF41, 0x2D00, 0xEDC1, 0xEC81,
0x2C40,
0xE401, 0x24C0, 0x2580, 0xE541, 0x2700, 0xE7C1, 0xE681,
0x2640,
0x2200, 0xE2C1, 0xE381, 0x2340, 0xE101, 0x21C0, 0x2080,
0xE041,
0xA001, 0x60C0, 0x6180, 0xA141, 0x6300, 0xA3C1, 0xA281,
0x6240,
0x6600, 0xA6C1, 0xA781, 0x6740, 0xA501, 0x65C0, 0x6480,
0xA441,
0x6C00, 0xACC1, 0xAD81, 0x6D40, 0xAF01, 0x6FC0, 0x6E80,
0xAE41,
0xAA01, 0x6AC0, 0x6B80, 0xAB41, 0x6900, 0xA9C1, 0xA881,
0x6840,
0x7800, 0xB8C1, 0xB981, 0x7940, 0xBB01, 0x7BC0, 0x7A80,
0xBA41,
0xBE01, 0x7EC0, 0x7F80, 0xBF41, 0x7D00, 0xBDC1, 0xBC81,

```

```

0x7C40,
0xB401, 0x74C0, 0x7580, 0xB541, 0x7700, 0xB7C1, 0xB681,
0x7640,
0x7200, 0xB2C1, 0xB381, 0x7340, 0xB101, 0x71C0, 0x7080,
0xB041,
0x5000, 0x90C1, 0x9181, 0x5140, 0x9301, 0x53C0, 0x5280,
0x9241,
0x9601, 0x56C0, 0x5780, 0x9741, 0x5500, 0x95C1, 0x9481,
0x5440,
0x9C01, 0x5CC0, 0x5D80, 0x9D41, 0x5F00, 0x9FC1, 0x9E81,
0x5E40,
0x5A00, 0x9AC1, 0x9B81, 0x5B40, 0x9901, 0x59C0, 0x5880,
0x9841,
0x8801, 0x48C0, 0x4980, 0x8941, 0x4B00, 0x8BC1, 0x8A81,
0x4A40,
0x4E00, 0x8EC1, 0x8F81, 0x4F40, 0x8D01, 0x4DC0, 0x4C80,
0x8C41,
0x4400, 0x84C1, 0x8581, 0x4540, 0x8701, 0x47C0, 0x4680,
0x8641,
0x8201, 0x42C0, 0x4380, 0x8341, 0x4100, 0x81C1, 0x8081,
0x4040};

return ((crc & 0xFF00) >> 8) ^ tbl[(crc & 0x00FF) ^ (data &
0x00FF)];
}

```

6.3 Command

Hex	Dec	Name [Parameter]	Page
0x80	128	GET CONFIG [<i>Parameter</i>]	(👉 2.4.2, Page 61)
0x81	129	SET CONFIG [<i>Parameter</i>]	(👉 2.4.1, Page 59)
0x82	130	GET CONFIG EXT [<i>Parameter</i>]	(👉 2.4.4, Page 68)
0x83	131	SET CONFIG EXT [<i>Parameter</i>]	(👉 2.4.3, Page 64)
0x84	132	FRAG START	(👉 2.7.2, Page 102)
0x85	133	FRAG MIDDLE	(👉 2.7.3, Page 102)
0x86	134	FRAG END	(👉 2.7.4, Page 104)
0x87	135	FRAG ACK [<i>D-Len</i>]	(👉 2.7.1, Page 102)
0x88	136	CMD ERROR [<i>Error code</i>]	(👉 2.8.1, Page 105)
0x89	137	CMD WARNING [<i>Error code</i>]	(👉 2.8.1, Page 105)
0x8A	138	CMD INFO [<i>Error code</i>]	(👉 2.3.1, Page 54)
0x8B	139	CMD ACK	(👉 2.8.1, Page 105)
0x90	144	CMD FAST STOP	(👉 2.2.19, Page 53)

Hex	Dec	Name [Parameter]	Page
0x91	145	CMD STOP	(👉 2.2.18, Page 52)
0x92	146	CMD REFERENCE	(👉 2.2.1, Page 33)
0x93	147	CMD MOVE BLOCKED [<i>Current position</i>]	(👉 2.3.2, Page 55)
0x94	148	CMD POS REACHED [<i>Current position</i>]	(👉 2.3.3, Page 56)
0x95	149	GET STATE [<i>Time</i>] [<i>Data</i>]	(👉 2.6.1, Page 84)
0x96	150	GET DETAILED ERROR INFO	(👉 2.8.1, Page 105)
0x97	151	CMD REFERENCE HAND	(👉 2.2.2, Page 34)
0x98	152	GET STATE AXIS	(👉 2.6.2, Page 87)
0xA0	160	SET TARGET VEL [<i>Speed</i>]	(👉 2.2.11, Page 47)
0xA1	161	SET TARGET ACC [<i>Acceleration</i>]	(👉 2.2.12, Page 48)
0xA2	162	SET TARGET JERK [<i>Jerk</i>]	(👉 2.2.13, Page 48)
0xA3	163	SET TARGET CUR [<i>Current</i>]	(👉 2.2.14, Page 49)
0xA4	164	SET TARGET TIME [<i>Time</i>]	(👉 2.2.15, Page 50)
0xA6	166	SET TARGET POS [<i>Position</i>]	(👉 2.2.16, Page 50)
0xA7	167	SET TARGET POS REL [<i>Rel. position</i>]	(👉 2.2.17, Page 51)

Hex	Dec	Name [Parameter]	Page
0xB0	176	MOVE POS [<i>Position</i>] [<i>Speed</i>] [<i>Acceleration</i>] [<i>Current</i>] [<i>Jerk</i>]	(👉 2.2.3, Page 36)
0xB1	177	MOVE POS TIME [<i>Position</i>] [<i>Speed</i>] [<i>Acceleration</i>] [<i>Current</i>] [<i>Time</i>]	(👉 2.2.5, Page 40)
0xB3	179	MOVE CUR [<i>Current</i>]	(👉 2.2.8, Page 44)
0xB5	181	MOVE VEL [<i>Speed</i>]	(👉 2.2.9, Page 45)
0xB7	183	MOVE GRIP [<i>Current</i>]	(👉 2.2.10, Page 46)
0xB8	184	MOVE POS REL [<i>Change of position</i>] [<i>Speed</i>] [<i>Acceleration</i>] [<i>Current</i>] [<i>Jerk</i>]	(👉 2.2.4, Page 38)
0xB9	185	MOVE POS TIME REL [<i>Change of position</i>] [<i>Speed</i>] [<i>Acceleration</i>] [<i>Current</i>] [<i>Time</i>]	(👉 2.2.6, Page 42)
0xBA	186	MOVE POS LOOP [<i>Position</i>] [<i>Speed</i>] [<i>Acceleration</i>] [<i>Current</i>] [<i>Jerk</i>]	(👉 2.2.7, Page 44)
0xC0	192	SET PHRASE [<i>Data frame</i>]	(👉 2.5.1, Page 71)
0xC1	193	EXE PHRASE [<i>Phrase number</i>]	(👉 2.5.4, Page 73)
0xC2	194	GET PHRASES	(👉 2.5.2, Page 72)
0xC3	195	PRG GOTO [<i>Phrase number</i>]	(👉 2.5.21, Page 83)

Hex	Dec	Name [Parameter]	Page
0xC4	196	PRG WAIT [<i>Time</i>]	(👉 2.5.22, Page 83)
0xCF	207	PRG EXE [<i>Program number</i>]	(👉 2.5.3, Page 72)

Hex	Dec	Name [Parameter]	Page
0xD0	208	EXE PHRASE0	(👉 2.5.5, Page 73)
0xD1	209	EXE PHRASE1	(👉 2.5.6, Page 74)
0xD2	210	EXE PHRASE2	(👉 2.5.7, Page 75)
0xD3	211	EXE PHRASE3	(👉 2.5.8, Page 75)
0xD4	212	EXE PHRASE4	(👉 2.5.9, Page 76)
0xD5	213	EXE PHRASE5	(👉 2.5.10, Page 77)
0xD6	214	EXE PHRASE6	(👉 2.5.11, Page 77)
0xD7	215	EXE PHRASE7	(👉 2.5.12, Page 78)
0xD8	216	EXE PHRASE8	(👉 2.5.13, Page 78)
0xD9	217	EXE PHRASE9	(👉 2.5.14, Page 79)
0xDA	218	EXE PHRASE10	(👉 2.5.15, Page 80)
0xDB	219	EXE PHRASE11	(👉 2.5.16, Page 80)
0xDC	220	EXE PHRASE12	(👉 2.5.17, Page 81)
0xDD	221	EXE PHRASE13	(👉 2.5.18, Page 81)
0xDE	222	EXE PHRASE14	(👉 2.5.19, Page 82)
0xDF	223	EXE PHRASE15	(👉 2.5.20, Page 82)
0xE0	224	CMD REBOOT	(👉 2.6.3, Page 90)
0xE1	225	CMD DIO [<i>Out</i>]	(👉 2.6.4, Page 91)
0xE2	226	FLASH MODE [<i>Password</i>]	(👉 2.6.8, Page 96)
0xE3	227	CHANGE USER [<i>Password</i>]	(👉 2.6.10, Page 98)
0xE4	228	CHECK MC PC COMMUNICATION [<i>Test data</i>] [<i>ParamCode</i>]	(👉 2.6.11, Page 98)
0xE5	229	CHECK PC MC COMMUNICATION [<i>Test data</i>] [<i>ParamCode</i>]	(👉 2.6.12, Page 100)
0xE6	230	CMD DISCONNECT [<i>Password</i>]	(👉 2.6.9, Page 97)
0xE7	231	CMD TOGGLE IMPULSE MESSAGE	(👉 2.3.7, Page 58)
0xE9	233	CMD GET DIO [<i>In</i>]	(👉 2.6.6, Page 93)
0xEA	234	CMD SET DIO [<i>Out</i>]	(👉 2.6.7, Page 94)
0xEB	235	CMD DIO AXIS [<i>Out</i>]	(👉 2.6.5, Page 92)

6.4 Information codes and error codes

Hex	Dec	Name	Page
0x0001	1	INFO BOOT	(👉 2.8.2, Page 107)
0x03	3	INFO NO RIGHTS	(👉 2.8.2, Page 107)
0x04	4	INFO UNKNOWN COMMAND	(👉 2.8.2, Page 107)
0x05	5	INFO FAILED	(👉 2.8.2, Page 107)
0x06	6	NOT REFERENCED	(👉 2.8.2, Page 107)
0x0007	7	INFO SEARCH SINE VECTOR	(👉 2.8.2, Page 107)
0x0008	8	INFO NO ERROR	(👉 2.8.2, Page 107)
0x09	9	INFO COMMUNICATION ERROR	(👉 2.8.2, Page 107)
0x10	16	INFO TIMEOUT	(👉 2.8.2, Page 107)
0x11	17	INFO UNKNOWN AXIS INDEX	(👉 2.8.2, Page 107)
0x16	22	INFO WRONG BAUDRATE	(👉 2.8.2, Page 107)
0x19	25	INFO CHECKSUM	(👉 2.8.2, Page 107)
0x1D	29	INFO MESSAGE LENGTH	(👉 2.8.2, Page 107)
0x1E	30	INFO WRONG PARAMETER	(👉 2.8.2, Page 107)
0x70	112	ERROR TEMP LOW	(👉 2.8.2, Page 107)
0x71	113	ERROR TEMP HIGH	(👉 2.8.2, Page 107)
0x72	114	ERROR LOGIC LOW	(👉 2.8.2, Page 107)
0x73	115	ERROR LOGIC HIGH	(👉 2.8.2, Page 107)
0x74	116	ERROR MOTOR VOLTAGE LOW	(👉 2.8.2, Page 107)
0x75	117	ERROR MOTOR VOLTAGE HIGH	(👉 2.8.2, Page 107)
0x76	118	ERROR CABLE BREAK	(👉 2.8.2, Page 107)
0x82	130	ERROR OVERSHOOT	(👉 2.8.2, Page 107)
0xC8	200	ERROR WRONG RAMP TYPE	(👉 2.8.2, Page 107)
0xD2	210	ERROR CONFIG MEMORY	(👉 2.8.2, Page 107)
0xD3	211	ERROR PROGRAM MEMORY	(👉 2.8.2, Page 107)
0xD4	212	ERROR INVALID PHRASE	(👉 2.8.2, Page 107)
0xD5	213	ERROR SOFT LOW	(👉 2.8.2, Page 107)
0xD6	214	ERROR SOFT HIGH	(👉 2.8.2, Page 107)
0xD8	216	ERROR SERVICE	(👉 2.8.2, Page 107)
0xD9	217	ERROR FAST STOP	(👉 2.8.2, Page 107)
0xDA	218	ERROR TOW	(👉 2.8.2, Page 107)
0xDB	219	ERROR VPC3	(👉 2.8.2, Page 107)

Hex	Dec	Name	Page
0xDC	220	ERROR FRAGMENTATION	(👉 2.8.2, Page 107)
0xDD	221	ERROR COMMUTATION	(👉 2.8.2, Page 107)
0xDE	222	ERROR CURRENT	(👉 2.8.2, Page 107)
0xDF	223	ERROR I2T	(👉 2.8.2, Page 107)
0xE0	224	ERROR INITIALIZE	(👉 2.8.2, Page 107)
0xE1	225	ERROR INTERNAL	(👉 2.8.2, Page 107)
0xE4	228	ERROR TOO FAST	(👉 2.8.2, Page 107)
0xEB	229	ERROR RESOLVER CHECK FAILED	(👉 2.8.2, Page 107)
0xEC	236	ERROR MATH	(👉 2.8.2, Page 107)

6.5 Supported hardware

Bus system	Card	Auto detect	Remark
Serial communication	PC internal	Yes	if "real" RS232 is available
Serial communication	USB-RS232 converter	Yes/No	Depending on PC configuration and manufacturer
CAN	Vector Informatik CANcardXL PCMCIA	Yes	
CAN	Vector Informatik all CAN cards	unknown	
CAN	esd CAN USB mini	Yes	
CAN	esd CAN PCI/331	Yes	
CAN	esd EtherCAN/2	Yes	
CAN	esd, other CAN cards	unknown	
CAN	Peak P-CAN USB	Yes	
CAN	IXXAT CAN iPC-320/PCI (VCI driver, version 3)	Yes	
CAN	IXXAT, other CAN cards	unknown	
CAN	Softing CAN-ACx-PCI	Yes	
CAN	Softing, other CAN cards	unknown	
Profibus	Hilscher PCMCIA CIF-60	Yes	
Profibus	Hilscher, other cards	unknown	
Profibus	WoodHead Applicom I/O	No	
Profibus	CP5611	Yes	
Profibus	Siemens S7	Yes	

7 Release notes

Version	Date	Remark
1.00	03.09.2007	Created
1.01	29.10.2007	Correction
1.16	31.01.2008	Adaption to Firmware V1.10
1.18	18.03.2008	Typing errors corrected
1.20	18.06.2008	Adaption to Firmware V1.20
1.22	29.07.2008	Adaption to Firmware V1.22
1.23	05.08.2008	Correction of the MD-SE parameter descriptions
1.24	02.09.2008	Correction
1.25	08.10.2008	Adaption to Firmware V1.23
1.26	19.11.2008	Adaption to Firmware V1.24
1.30	08.05.2009	Adaption to Firmware V1.30
1.40	10.12.2009	Adaption to Firmware V1.40
1.41	08.03.2010	Adaption to Firmware V1.41
1.43	31.05.2010	Axis index functionality
1.50	13.01.2011	Adaption to Firmware V1.50
1.55	30.03.2012	Adaption to Firmware V1.55
1.56	19.06.2012	Adaption to Firmware V1.56
1.56	10.11.2014	Correction
1.59	19.06.2014	Adaption to Firmware V1.59
1.60	14.09.2014	Adaption to Firmware V1.60