*phy*LOGIC™
Command Reference for the *phy*MOTION™ Controller

TRANSLATION OF THE GERMAN ORIGINAL MANUAL

Version	Modification
7	New instructions for the touch panel, BT5 AM terminal, AIOM and AIM modules
8	Supplement: axes status inquiry (chap. 6.17)
9	New: write instructions via serial interface (chap. 6.4)
10	New instructions, P45
11	Axis status command; Control pulses exit
12	BiSS encoder

Phytron GmbH

Industriestraße 12

82194 Gröbenzell, Germany

Tel.: +49(0)8142/503-0

Fax: +49(0)8142/503-190

In this manual "*phy***LOGIC**™ *Command reference for the phy***MOTION**™ *Controller"* (http://www.phytron.de/*phy*MOTION) are the descriptions of the commands and programming for the *phy***MOTION**™ stepper motor controller.

This manual is supplementary to the "*phy***MOTION**™ *Modular Multi-axis Controller for Stepper Motors"* manual.

Every possible care has been taken to ensure the accuracy of this technical manual. All information contained in this manual is correct to the best of our knowledge and belief but cannot be guaranteed. Furthermore we reserve the right to make improvements and enhancements to the manual and / or the devices described herein without prior notification.

We appreciate suggestions and criticisms for further improvement.

Email address: doku@phytron.de

Questions about the use of the product described in the manual that you cannot find answered here, please contact your representative of Phytron (http://www.phytron.de/) in your local agencies.

1 Information

This manual:

Read this manual very carefully before mounting, installing and operating the device and if necessary further manuals related to this manual.

- Please pay special attention to instructions that are marked as follows:

DANGER – Serious injury!	*Indicates a high risk of serious injury or death!*
DANGER – Serious injury from electric shock!	*Indicates a high risk of serious injury or death from electric shock!*
WARNING – Serious injury possible!	*Indicates a possible risk of serious injury or death!*
WARNING – Serious injury from electric shock!	*Indicates a possible risk of serious injury or death from electric shock!*
CAUTION – Possible injury!	*Indicates a possible risk of personal injury.*
CAUTION – Possible damage!	*Indicates a possible risk of damage to equipment.*
CAUTION – Possible damage due to ESD!	*Refers to a possible risk of equipment damage from electrostatic discharge.*
”Any heading“	*Refers to an important paragraph in the manual.*

Observe the following safety instructions!

Qualified personnel

WARNING – Serious injury possible!

Serious personal injury or serious damage to the machine and drives could be caused by insufficiently trained personnel!

Without proper training and qualifications damage to devices and injury might result!

- Design, installation and operation of systems may only be performed by qualified and trained personnel.
- These persons should be able to recognize and handle risks emerging from electrical, mechanical or electronic system parts.
- The qualified personnel must know the content of this manual and be able to understand all documents belonging to the product. Safety instructions are to be provided.
- The trained personnel must know all valid standards, regulations and rules for the prevention of accidents, which are necessary for working with the product.

Safety Instructions

CAUTION – Possible damage!

Malfunctions are possible while programming the instruction codes – e.g. sudden running of a connected motor, braking etc.

- Please test the program flow step by step.

CAUTION – Possible damage!

For each application, the functional reliability of software products by external factors such as voltage differences or hardware failure, etc. is affected.

- To prevent damage due to system error, the user should take appropriate safety measures. These include back-up and shut-down mechanisms.

CAUTION – Possible damage!

Each end user system is customised and differs from the testing platform. Therefore the user or application designer is responsible for verifying and validating the suitability of the application.

- The suitability of the device's use must be tested and validated.

i

CAUTION – Possible damage!

Some modules are set to a default value on delivery. So, e.g., the motor current must be set to the corresponding value (see the motor data from the motor manufacturer). Connected components like motors can be damaged by incorrectly set values.

- Please check before starting, if the parameters are correct.

2 Contents

3 Introduction

“Minilog“ or “*phy***LOGIC**™“?

If you use a MCC, OMC or TMC controller, you continue to use the familiar syntax of the Minilog commands.

*phy***LOGIC**™ is based on the instruction set Minilog and has been extended by commands that support particularly complex multi-axis and contains some different commands.
If you use a *phy***MOTION**™ controller, the new *phy***LOGIC**™ syntax is necessary.

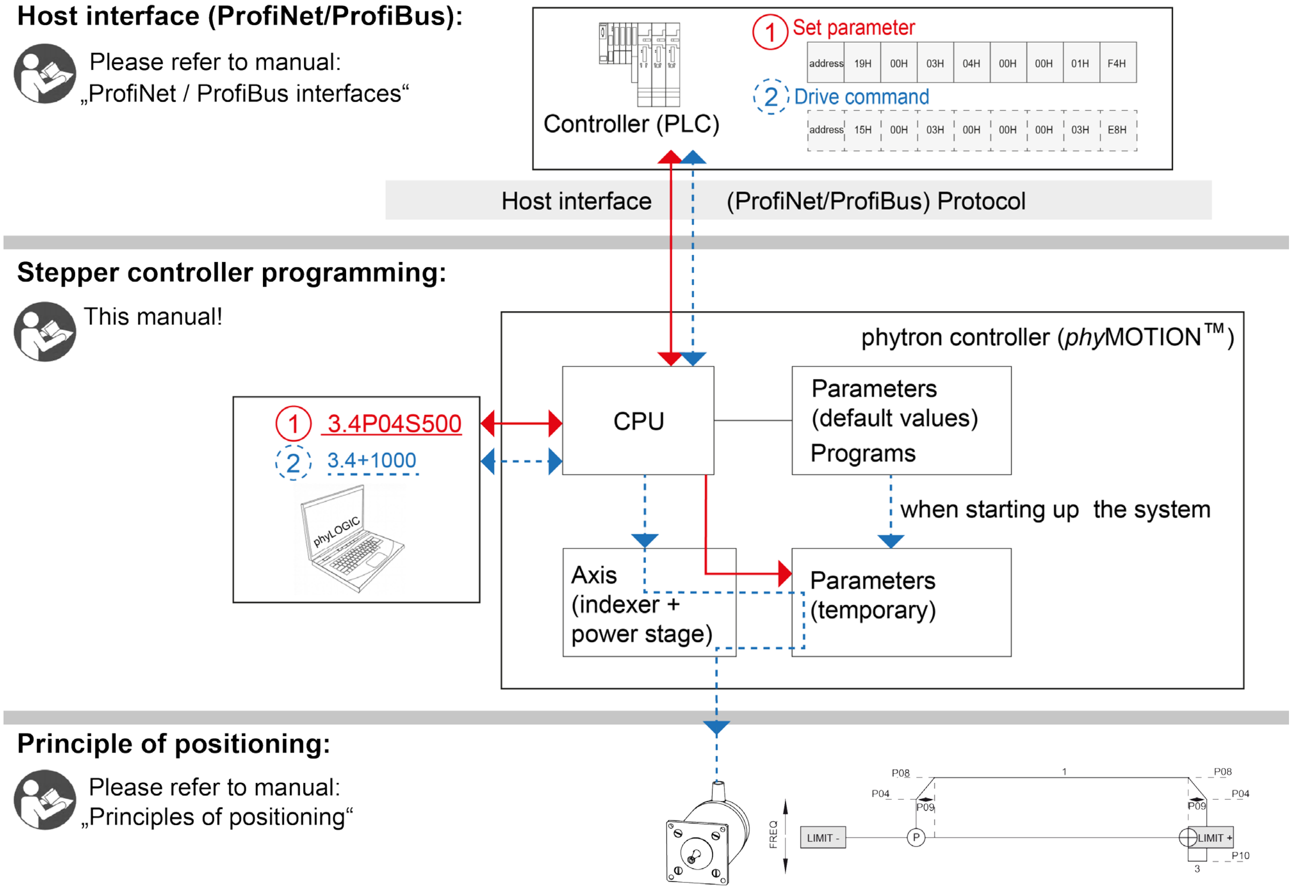
Minilog-Comm knows only the Minilog syntax, but the new *phy***LOGIC**™-ToolBox software contains both the old syntax and the new *phy***LOGIC**™ syntax.

*phy***LOGIC**™ instructions can easily be sent to the controller with phytron’s programming software (*phy***LOGIC**™ Toolbox) via USB, embedded into other protocols like Ethernet or into interface protocols like ProfiBus / ProfiNet.

You can parameterise your commands (e.g. a driving command) per axis either just the first time you set up your system, or adjust the parameters temporarily before sending a driving command.

Example: For “relative run” you can set: step resolution (P45), run current (P41), run frequency (P14), start stop frequency (P04 always “0” with the I1AM01), ramp (P15), recovery time (P16), boost (P17), boost current (P42), current delay time (P43), etc.

Use this illustration to find the adequate manual for your programming task:



Each of our programmable controllers comes along with pre set parameters (default values), which are automatically loaded into the temporary memory of each axis while starting the device. These parameters can be changed during your program is executed to optimise your motion tasks at any time.

If you want your controller to wake up with a new set of parameters, you have to explicitly store them in the non volatile storage of the main CPU unit by using a certain command.

CAUTION – Possible damage!

Some modules are set to a default value on delivery. So, e.g., the motor current must be set to the corresponding value (see the motor data from the motor manufacturer). Connected components like motors can be damaged by incorrectly set values.

- Please check before starting, if the parameters are correct.

This Manual

*You find the complete instruction reference to the phy***LOGIC**™ *in this manual.*

Further Manual

An overview of axis commands and associated parameters, as well as schematic representations of the driving parameters can be found in the following manual:

„Principles of Positioning for Stepper Motor Controllers“

Further Manual

*How to embed phy**LOGIC**™ instructions into the int*erface protocols ProfiNet and ProfiBus can be found in the following manual:

„*phy***LOGIC**™-ProfiNet/ProfiBus Interface“

Besides, complete sequential programs can be realized with *phy***LOGIC**™: drive instructions, initializing axes, sub programs, jump instructions, reading and setting registers and many other special instructions.

For editing and managing *phy***LOGIC**™ programs, the *phy***LOGIC**™ ToolBox communication software for PC is delivered with the *phy***MOTION**™ controller.

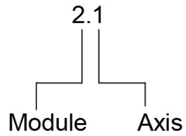
4 Structures

4.1 Structure of the Command Code

2.1rvalue

2.1 — The bold characters represent the instruction code and must be used unchanged.

In this example: **2.1** represents the motion instruction code for relative positioning of the first axis of module 2:



r — Small letter require the input of the characters or values which are described in the column *Meaning*.

In this example: r = running direction + or – .

value — In this example a running distance of 1000 is fed in. The corresponding unit (e.g. steps) of the particular input is defined by **parameters**. For the specific parameters refer to chap. 6.

Example:
2.1+1000

Relative motion instruction for the axis 1 of module 2:
Go 1000 steps to the + direction.

Important:

- **All characters and signs, belonging to one single instruction, must be written without a blank.**
- **The instructions themselves must be separated by a blank.**
- **Leading zeros in an instruction are ignored (Example: the instruction A003.1S is realized as A3.1S)**
- **Instructions, which cannot be used in the program <u>and</u> direct mode, are marked with:**
  1. **Instruction only used in the program (ONLY PROG)**
  2. **Instruction only used in the PC direct mode (ONLY PC)**

Exception: In the instruction group “System Status (ONLY PC)”, chapter 4.13, the first program name must be separated by a blank or (;) from the following alphanumeric part of the instruction code.

4.2 Data and Telegram Format

Data format: No Parity

1 Stop bit

8 Bit ASCII-Code

115 200 Baud

Address: is always 0
exception: with RS485: 0 to F rotary switch

The **send telegram** from PC via RS interface is defined as:

<STX> | Address | Instruction | Separator | Checksum | <ETX>

The **response telegram** (always for address 0-9, A-F) is defined as:

<STX> | ACK | Answer | Separator | Checksum | <ETX> or

<STX> | ACK | Separator | Checksum | <ETX> or

<STX> | NAK | Separator | Checksum |<ETX>

	Meaning
<STX>	<STX> (Start of Text, 02_H): It is exclusively used as the start code for a new telegram
Address	Address of the controller, the range of the address byte is 0 to 9 and A to F (30_H…39_H and 41_H…46_H). Additional the Broadcast[1] address @ (40_H) is used.
Instruction	*phy*LOGIC™ instruction code
Separator	: Colon ($3A_H$) as separator, to distinguish between usable data and checksum.
Checksum	Upper byte of the checksum value (see below for the algorithm to calculate the checksum)
	Lower byte of the checksum value (calculation see below)
<ETX>	(End of Text, 03_H) this code indicates the end of the telegram.
ACK	(Acknowledge 06_H), the instruction has been confirmed.
NAK	(Negative Acknowledge 15_H), the instruction has been negatively confirmed.
Answer	Answer as number or string, e.g. E or N

The checksum CS is defined by summing up all bytes, beginning with the address byte and including the separator (:) in an exclusive-OR-operation:

```
CS = address ⊕ data byte 1 ⊕ data byte 2 . . . ⊕ data byte n ⊕
     separator
```

[1] Broadcast: All axes receive and evaluate the telegram. To avoid bus-conflicts caused by the response of all axes nearly within the same time, the response of the controllers is suppressed by addressing with “@”.

The checksum is calculated as one 8-bit binary value (00_H to FF_H). This byte is taken apart in its upper and lower byte (nibbles). After the HEX values of the two nibbles have been transferred to the corresponding two ASCII characters (0 to 9 instead of 0_H to 9_H and A to F instead of A_H to F_H, that means to each nibble 30_H or rather 37_H is mathematically added), the checksum is written in the telegram.

The controller also calculates (Exclusive OR) the checksum of the received data. The telegram will be rejected if a difference to the received checksum is detected, and the error is confirmed by NAK.

If there is no need to validate the contents of the telegram, the checksum monitoring can be set off. Instead of the checksum bytes, **two X** characters will be accepted, e. g.:

```
<STX>| 0 | 1 | . |1 | + | 1 | 0 | 0 | : | X | X | <ETX>
```

5 Programming with *phy*LOGIC™

5.1 Design of *phy*LOGIC™ Programs

- *phy***LOGIC**™ programs consist of up to n program rows which are displayed in the editor.
- The single instructions in each row must be separated by blank characters.
- Do not insert extra blank characters within an instruction.
- The instructions will be executed serially.
- By means of labels jump instructions or subroutines can be defined.
- Parameter and register values should be defined at the beginning of a program.
- Parameter and register numbers may be entered with or without preceding zeros.

  Example: R0001 or R1

5.2 Addressing Mode

For instructions, where at least one register is used as an operator, two addressing modes are available: The **Direct Addressing Mode** and the **Indirect Addressing Mode**. In this programming manual the meaning of the basic instruction is always explained for the **Direct Addressing Mode**. The variations of the **Indirect Addressing Mode** are listed for the sake of completeness. The first named register within an instruction code is always the destination register for the result.

Example for Direct Addressing Mode:

Instruction	***Meaning***
Rnn**BE**m.n**–**m.y	The status of the inputs m.n to m.y is written as a binary value into the register nn. **Example: R1BE1.1–1.8** Status of the inputs 1 to 8 (of module 1) is e.g. **1010 0101**. This binary value is written into the register 1. After the Instruction was carried out, the register content is 165 decimal.

Example for Indirect Addressing Mode:

Instruction	*Meaning*
R[Rnn**]BE**m.n–m.y	**Indirect Addressing Mode:** The status of the inputs m.n to m.y is written as a binary value into the register which is addressed by the register **[Rnn]**. Example: **R1S10 R[R1]BE1.1–1.8** The addressing register **[R1]** is set to 10. The status of the inputs 1 to 8 (of module 1) is e.g. 1010 0101. This binary value is written into the register 10, which was addressed by the register 1. After the instruction was carried out, the content of the register 10 is 165 decimal.

Addressing with Label

In case of jump calls (see chap. 6.11) and subroutine calls (see chap. 6.26) the start or destination row can be set in the instruction code with a label (*la*), which is assigned to this program row. A label is defined between two * and can have up to 6 alphanumeric signs. Max. 512 labels can be used in one program.

Example: *[label name]*

Program name:

Program names [name] in the instruction code can have up to 8 alphanumeric signs.

5.3 Conditional Instructions

The execution of some instructions (e.g. jumps or subroutine calls) can be combined with a condition. Before a conditional jump etc. can be used, the condition byte has to be set, for example by an input request (see chap. 4.4) or a register comparison (see chap. 4.11)

Possible states of the condition byte:

E = Condition fulfilled **N** = Condition not fulfilled

The state of the condition byte remains stored until it is changed again.

All instructions which set no condition delete the condition request.

6 *phy*LOGIC™ Instructions

6.1 AD Converter

Instruction	***Meaning***
ADm.n	Read channel n of the AD converter of the module m. **Response:** **<STX><ACK> value :CS<ETX>** value=0 to 655536 or ±32768
ADm.n**B**z	Read block z of the channel n of the module m (of the AD converter). z = 1 to 64 **Response:** **<STX><ACK> value; value; value;….. :CS<ETX>** 128 „ value “ of each block
ADm.n**R**	Start the recording of measurement of channel n of the AD converter module m **Response:** **<STX><ACK>:CS<ETX>**
ADm.n**S**	Stop the recording of measurement of channel n of the AD converter module m. **Response:** **<STX><ACK> :CS<ETX>**
ADm.n**T**value	Set the channel n of the AD converter module m to its function. value=0 → unipolar voltage value=1 → bipolar voltage value=2 → unipolar current
ADm.n**T**	Read the function of the channel n of the AD converter module m. **Response:** **<STX><ACK> value :CS<ETX>** value=0,1 or 2

> **i** These values are read card based, without specifying the channel:

ADm**V**x **R**yy**=AD**m**V**x	Read the value x from recorded block of the module m. Read the value x from recorded block of the module m and write this value into the register y.
ADm**W** **R**yy**=AD**m**W**	Read number of the recorded values of the module m. Read number of the recorded values of the module m into the register y.

Instruction	***Meaning***
ADm**Z**value	Setting the measurement cycle value → measuring cycle from 1 to 3000 1=33 µS 2=66 µS…. 3000=100 ms
ADm**ZR**	Read time value of the measurement cycle **Response:** **<STX><ACK> value :CS<ETX>** value=1 to 3000

6.2 Outputs

Instruction	***Meaning***
	Set Outputs
Am**.**nz	Set one output. z = S → set z = R → reset m→ number (ID) of the module n, y, x → number (ID) of the output
Am**.**n**==**z	Read status of output n of the module m and set the condition byte. If z=S, output is ON. If z=R, output is OFF.
Am**.**nzm**.**yzm**.**xz	Set/read several outputs. **Example: A1.1S1.2R1.3S** Output **1** and **3** ON, output **2** OFF
	Read Output
AGm**R**	Read the state of the module number (m). (ONLY PC) **Example: AG2R** The 2nd output group is read **Response :** **<STX><ACK> value :CS<ETX>** (ONLY PC) value = 0 to 255

Instruction	***Meaning***
	Set the output group outputs
AGm**S**value	Set the output group m= module number value: 0 to 255 **Example: AG1S170** The 1. output group is set with the information '10101010'
	Read Output Status
AZm**.**n;m**.**y;m**.**x	The state of the outputs (of module m) n, y, x is read. (ONLY PC) **Example: AZ1.1;1.2;1.3** **Response : <STX><ACK>rrr:CS<ETX>** r = 0 Output OFF r = 1 Output ON **Important:** Set a **;** between the output numbers.

6.3 Reset

Instruction	***Meaning***
CR	Reset of the controller by the interface.
CT	The entire display of the operating panel is cleared via the interface..
CTn	Delete a single row n--> row number n=1 to 4 (BT5 AM operating panel)
CTn;m	Delete selected rows. n or m--> row number n=1 to 4; m=1 to 4 (BT5 AM operating panel) **Response : <STX><ACK><ETX><CR><LF>** (ONLY PC)

6.4 Write Instructions via Serial Interface

Instruction	*Meaning*
	Information can be sent via 3 serial interfaces. The writing is sent without formatting. s = 1,2 or 3 → interface name 1 → USB interface 2 → terminal interface 3 → fieldbus, RS and Ethernet interface

Instruction	*Meaning*
Ds= **<**Text**>**	The bracketed expression sent.
Ds=**R**nn	The content of the register nn is sent.
Ds=**R[R**nn**]**	The content of the register which is addressed by register nn is sent.
Ds=m.n**P**mm	Parameter mm of the axis n of the module m is sent. mm = 1 to max. number (ID) of the parameter **Example: D1=5.2P10** The parameter 10 of the axis 2 of the module 5 is carried out via the USB interface.

6.5 DA Converter

Instruction	***Meaning***
DAm**.**n=value	Output value to the channel n of the DA converter module m. value=0 to 65535 or ±32767
DAm**.**n	Read the channel n of the DA converter module m. **Response:** **<STX><ACK> value :CS<ETX>** value=0 to 65535 or ±32767
DAm**.**nTvalue	Set the channel n of the DA converter module m to its function. value=0 → unipolar voltage value=1 → bipolar voltage value=2 → unipolar current
DAm**.**nT	Read the function of the channel n of the DA converter module m. **Response:** **<STX><ACK> value :CS<ETX>** value=0,1 or 2

6.6 Input Requests

Instruction ***Meaning***

Logical AND

E^m**.**nzm**.**yzm**.**xz The inputs n, y, x are tested as AND condition.
Only if the AND condition is fulfilled the condition byte is set.
Otherwise the condition byte is reset.

m → module number
n. y. x → input number
z = S → input ON
z = R → input OFF

Example: E^1.1S1.2R1.3S
The input states 1, 2 and 3 are read out. If input 1 is set, input 2 reset and input 3 set, the AND condition is fulfilled and the condition byte is set.
Now a conditional jump or a conditional call of a subprogram can be carried out.

Response: <STX><ACK> E :CS<ETX> or
<STX><ACK> N :CS<ETX> (ONLY PC)

Logical OR

Evm**.**nzm**.**yzm**.**xz The inputs n, y, x are tested as OR condition.
Only if the OR condition is fulfilled the condition byte is set.
Otherwise the condition byte is reset.

m → module number
n/ y/ x → input number
z = S → input ON
z = R → input OFF

Example: Ev1.1S1.2R1.3S

The input states **1, 2 and 3** are read out. If input 1 is set or input 2 reset or input 3 set, the AND condition is fulfilled and the condition byte is set.
Now a conditional jump or a conditional call of a subprogram can be carried out.

Response: <STX><ACK> E :CS<ETX> or
<STX><ACK> N :CS<ETX> (ONLY PC)

Wait for Condition Fulfilled

Em**.**nz Wait for the preset input condition.
The program stops until the preset input condition is fulfilled. The condition byte is not affected. (ONLY PROG)

Instruction	***Meaning***
Em**.**nzm**.**yz	When reading the status of several inputs, one input after the other is read out (no AND linking). The condition byte is not affected. (ONLY PROG) **Example: E1.1S1.2R1.3S** The status of the inputs 1, 2 and 3 (of module 1) are read. After the input 1 is set, the input 2 is read. After the input 2 is reset, the input 3 is read. After the input 3 set, the reading Instruction is done and the program continues. After the instruction end the inputs1 and 2 can have another state.
Em**.**n**==**z	Read status of the input n of the module m and set the condition byte. If z=S, input is ON. If z=R, input is OFF.
Em**.**n=S;instruction	The instruction is set to the input n of the module m. **Example 1: E1.2=S; R1+10** If input 2 of the module 1 is ON, 10 is added to the register value of R1. **Example 2: E1.2=S;U*test*** If input 2 of the module 1 is ON, the subroutine “test” is called.
ECm**=**n	The input 1 of the module m is set as counter with edge n. m→module number n→edge n=0 rising edge n=1 falling edge
ECm**R**	The input counter 1 of the module m is read. m→module number **Response: <STX><ACK> value:CS<ETX>**
	Read input group
EGm**R**	The input group n is read. (ONLY PC) **Response : <STX><ACK>value:CS<ETX>** value=0 to 255
EZm**.**n;m**.**y;m**.**x	The Status of the inputs n, y, x is read (ONLY PC). **Response: <STX><ACK>nnn:CS<ETX>** n = 0 input is reset n = 1 input is set **Important:** Set a **;** between the input numbers.

6.7 Program Manipulation at Emergency Stop (ONLY PROG)

Instruction	***Meaning***
FN*la*	The program row, at which the program has to be continued in the case of an emergency stop, is defined by a label.

6.8 Set Controller to factory setting

Instruction	***Meaning***
GW*.*	Controller's values and axis parameters are reset and all registers and programs are deleted

6.9 Program Interruption

Instruction	***Meaning***
H	The program waits here until all axes have stopped. (ONLY PROG)

6.10 System Adaption during Program Execution

Instruction ***Meaning***

Number of Axes

IAR
The number of existing axis modules is requested (ONLY PC).
Response: <STX><ACK>x:CS<ETX>

x = 1 to n axis modules

IAn
Read the number of existing axes per module (ONLY PC).
Response: <STX><ACK>a:CS <ETX>

a = 1 to 4 per module

IACn
Read the encoder type (ONLY PC).
n=1 to number of axis modules
Response: <STX><ACK>a:CS <ETX>

a → encoder module type 4Byte
Example: 000A

0	0	0	A
4th axis	3rd axis	2nd axis	1st axis

a=0 → no module
a=A →encoder ECAS01
a=E →encoder ECES01
a=M→encoder ECMS01
a=B→encoder ECBS01

Instruction	***Meaning***

IAEn

Read the power stage type (ONLY PC).
n=1 to number of axis modules
Response: <STX><ACK>a:CS <ETX>

a → power stage type 4 Byte:
Example: 00aa

0	0	a	a
4^{th} axis	3^{rd} axis	2^{nd} axis	1^{st} axis

a=0 → power stage unknow
a=Z → ZMX^{+}
a=a → APS internal
a=I → I1AM01
a=X → MSX 152-120
a=M → MSX 102-120
a=S → MSX 52-120

IATn

Read the temperature module type (ONLY PC).
n=1 to number of axis modules
Response: <STX><ACK>a:CS <ETX>

a → temperature module type 4 Byte
Example: 00PP

0	0	P	P
4^{th} axis	3^{rd} axis	2^{nd} axis	1^{st} axis

a=0 → no module
a=P → PT 100 sensor
a=K → K type

Automatic Start

IBSname

The name of the start program is written into the Auto Start register. If the REMOTE/LOCAL switch is in the LOCAL position the program execution starts here.

Response: <STX><ACK>:CS<ETX> or
<STX><NAK>:CS<ETX> (ONLY PC)

IBC

Program name is deleted out of the Auto Start register.

IBR

The name of the Auto Start program is read (ONLY PC).

Response: <STX><ACK>name:CS<ETX>

ProfiBus Ethercat or CAN Data (for service purposes)

IBER

Read data received by Ethercat and CAN

IBES

Read data written by Ethercat and CAN

Instruction	***Meaning***
IBPRx**;**y	Read data received by ProfiBus and ProfiNet x→start y→end
IBPSx**;**y	Read data sent by ProfiBus and ProfiNet
	Read/Set bau drate (ONLY PC)
ICn**S**baud	Set the baud rate for the interfaces. n = 1 → COM 1 for RS or USB and Bluetooth: baud = Baud rate (9600, 19200, 38400, 57600, 115200 or 230400 Baud) for CAN: baud = Baud rate (125000, 250000, 500000, 1000000 Baud)
ICn**R**	Baud rate setting of the interfaces is read out. n = 1 to 3 1→ USB 2→ Debug 3→ Field bus **Response,** **if n=3 and Field bus type>=3 and <6: <STX><NAK>:CS<ETX>**
	Read Field bus
IC3**T**	Only for field bus! Read interface type for field bus **Response: <STX><ACK>n:CS<ETX>** n = 0→ no module n = 1→ RS 232 or RS 485 are implemented n = 2→ CAN interface n = 3→ ProfiBus n = 4→ ProfiNet n = 5→ Ethernet n = 6→ Bluetooth
IC3**HVS**x	Only for RS bus! (if field bus=1) Set RS bus x=0 → RS 485 4-wire x=1 → RS 485 2-wire

Instruction	***Meaning***
IC3**HVR**	Read RS bus! (if field bus=1) **Response: <STX><ACK>n:<CS><ETX>** n = 0 → RS 485 4-wire n = 1 → RS 485 2-wire
	Set CAN bus (If field bus=2)
IC3**IDBS**x	Set basic address (100_{hex})
IC3**IDBR**	Read basic ID
IC3IDR	Read basic ID and (controller's address switch x 10)
IC3**IDLS**x	Set ID length x=0→ length=11 Bit x=1→ length=29 Bit
IC3IDLR	Read ID length **Response: <STX><ACK>n:CS<ETX>** n = 0→ length =11 Bit n = 1→ length =29 Bit
	Read/Set ProfiBus (if field bus=3)
IDR	Read the controller's address for ProfiBus If **Response: <STX><ACK>(0…125):CS<ETX>**, then the basic setting =1
IDSn	Set the controller's address for ProfiBus n = 1 to 125
	Remote/Local Reversing (ONLY PC)
IFR	The controller is reversed to the Remote function. If a program is running, it is cancelled. If the switch is positioned to Local, the position Remote is simulated. **Response: <STX><ACK>:CS<ETX>**
IFL	The controller is reversed to the function Local, if the Remote/Local switch is on the position Local. If the switch is on Remote position, it is not reversed. **Response: <STX><ACK>:CS<ETX>**
	Module Identification (ONLY PC)
IIPR	Read the IP address of the Ethernet module
IMA	Read number of modules.
IMAn	Read number of the existing axes per module.(ONLY PC). **Response: <STX><ACK>a:CS <ETX>**

Instruction	***Meaning***
	a = 1 to 4 per module
IMAI	Read number of the analogue input modules.
IMAIO	Read number of the analogue input and output modules.
IMAO	Read number of the analogue output modules.
IMDI	Read number of the digital input modules.
IMDIO	Read number of the digital input and output modules.
IMDO	Read number of the digital output modules.
IMn	Check module slot on the module type n =0 to maximum number of modules n=0→MCM module **Response: <STX><ACK>a:CS<ETX>** a=I1AM01 → one axis stepper motor control module (3.5 A) a=I1AM02 → → one axis stepper motor control module (5 A) a=IDXM01 → Indexer module (4 axes) a=DIOM01 → Digital I/O module a=MCM01 → Main Controller module a=AIOM01 → Analogue I/O module a=AIM01 → Analogue input module a=AOM01 → Analogue output module
IMR	Read MAC address
	Static IP Address
IPS192.168.0.10	IP address is set to 192.168.0.10
IPS0.0.0.0	Delete static IP address
	Operating mode
IPT=x	The operating mode of the command type is defined. x=0: *phy***LOGIC**® mode x=1: G-Code mode
IPTR	The operating mode of the command type is read.
	Directory of FLASH
IP	Interrogate the number of programs of the controller.
IPn	Read the n program name of the program list out of the RAM. If no program name exists, the response NAK is shown (ONLY PC). **Response: <STX><ACK>name:CS<ETX>** **Response: <STX><NAK>:CS<ETX>** if no name available
IPM=n	Only for ProfiBus and ProfiNet:

Instruction	***Meaning***
	Basic setting: 1 x 8-byte for MCM module (master) Create number of 8-byte blocks, that remain active after reset n=1 to 4 →1 to 4 x 8-byte
IPMR	Read the number of 8-byte blocks, created for the MCM module
IR	Interrogate the stored register sets of the controller
IRn	Read the n register name of the program list out of the flash. (ONLY PC). **Response: <STX><ACK>name:CS <ETX>** **Response: <STX><NAK>:CS <ETX>** if no name available
ITR	Interrogate the stored text registers of the controller.
ITRn	Read the n text register name of the program list out of the flash (ONLY PC). **Response: <STX><ACK>name:CS <ETX>** **Response: <STX><NAK>:CS <ETX>** if no name available
ITAIOn	Interrogate the analogue module type **Response: <STX><ACK>type:CS <ETX>** n=1 to max. number of the analogue modules type=AIOM01, AIM01 or AOM01
ITIOn	Interrogate the digital module type **Response: <STX><ACK>type:CS <ETX>** n=1 to max. number of the digital modules type=DIOM01
ITIDXn	Interrogate the Indexer module type **Response: <STX><ACK>type:CS <ETX>** n=1 to max. number of the Indexer modules type=IDXM01, I1AM01 or I1AM02
	System name
ISN=name	*phy***MOTION**™ is assigned a name that can be read again
ISN	The system name is read back
	Version Request
IV	Read the number of modules.
IVAIOn	Read the system software status of the n AIOM/AIM or AOM module.
IVM	Read the system software status of the MCM module.
IVIOn	Read the system software status of the n DIOM module.

Instruction	***Meaning***
IVIDXn	Read the system software status of the n I1AM or IDX module.
IV0	Read the software version of the MCM module (Loader, System-Lib, MiniLog system)
IVm	The software version of the module m (loader, system-lib, MiniLog system) is read (ONLY PC). m=1 to max. modules **Response: <STX><ACK>Software Version:CS<ETX>** **Check field bus (Deutschmann)**
IVB	Read the script version

6.11 Interpolation Commands (ONLY for I4XM01)

Linear interpolation

Enter the distance or position in P18 of the axis (i.e.1.1P18S5000 1.2P18S1000)

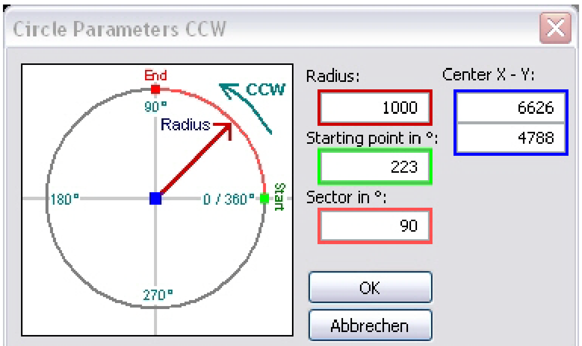
m**I**aw;bw;cw;dw Start the interpolation (ONLY for I4XM01)

m → module number
a,b,c,d → Number of the axis
w=A → drive absolutely
w=R → drive relatively

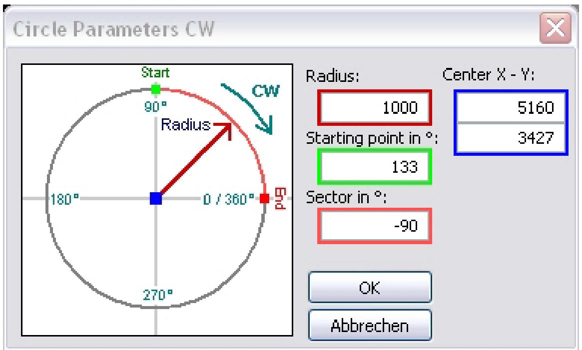
Example: 1I1A;2R → Module 1, Axis 1 absolute, Axis 2 relative

Response: <STX><ACK>:CS<ETX> (ONLY PC)

Circular interpolation



counter-clockwise (CCW)



clockwise (CW)

xKRn — Set the radius n of the circular arc for the Indexer module x,
the unit and the factor of n are defined in P2 and P3 (see chap. 6)

xKSn — Set the starting point n on the circular path for the Indexer module x in degree (°)
n =0 to 360°

xKWn — Set the path (sector) n in degree (°) from the starting point for the Indexer module x
n =0 to 360° (CCW)
n =0 to -360° (CW)

Important: Write these 3 commands in **1** program row!

Example: 1KR100 1KS90 1KW180

x**KG**a;b — Set the axis assignment of the Indexer card x ,

a= Master axis (1,2 or 3)

b=Slave axis (1,2,3 or 4)

x**KT**a:b — Set the divider of axis a and axis b of the Indexer module x (for ellipsis run)

a: divider for axis 1
b: divider for axis 2

6.12 Load Register Set

Instruction	***Meaning***
LRname	Register set name is loaded into the register RAM.
LTRname	Text register set name is loaded into the text register RAM.
LR=Rnn	Load register set from the memory nn→Register number from 1 to 1000
LTR=Rnn	Load text register set from the memory nn→Register number from 1 to 1000

Instruction	***Meaning***
LR=TDx	Get the name from the dropbox x by loading the register x
LTR=TDx	Get the name from the dropbox x by loading the text register x

6.13 Jump Instructions (ONLY PROG)

Instruction	***Meaning***
	Absolute Jump
N*la* **goto** *la*	Absolute jump. Destination program row number: marked by the label *la*.
	Conditional Jump Absolute / E = Condition Fulfilled
NE*la*	Absolute jump. Destination program row: marked by the label ***la***.
	Conditional Jump Absolute / N = Condition Not Fulfilled
NN*la*	Absolute jump. Destination program row number: marked by the label ***la***.

6.14 Password

Instruction	***Meaning***
PA␣	The controller is activated, if there is no password. Then it is not possible to lock the controller.
PAname	The password protected controller is activated.
PSname	This command allocates the controller a password. It has maximum 8 alphanumeric signs.
	Activation Status for Programs, Parameters and Registers
PWSp	Set activation status Programs, parameters and registers can be activated or locked by a password protected controller. p → Activation status for programs, parameters and registers p = 0 all activated p = 1 Program R/W locked p = 2 Parameter R/W locked p = 4 Register R/W locked p is between 0 and 7 Example: PWS**5** Program and register locked (1+4=5)
PWR	Read activation status

Instruction	***Meaning***
	The answer is two digits. **<ACK> > sp:CS <ETX>** s → Activation status of the controller s = 0 Controller locked s = 1 Controller activated p → Activation status for programs, parameters and registers see above
	Example: **PWR** **<ACK> 15:CS <ETX>** s = 1 → Controller activated p = 5 → Program and register locked (1+4=5)

6.15 Ending or Interruption of a Program Call (ONLY PROG)

Instruction	***Meaning***
PE	The actual program is ended and the system waits for another changing of the **REMOTE/LOCAL** switch. If the program was started via computer, the system goes back to the **COMPUTER MODE**.

6.16 Program and Data Management (ONLY PC)

Instruction	***Meaning***
	Delete Programs and Data
QDA*.*	All axis parameters are set to default values.
QDP*.*	All programs, register and text register sets in the flash are deleted.
QDPname	The program *name* is deleted.
QDR	All registers in the RAM are set to zero.
QDRname	The register set *name* is deleted.
QDTRname	The text register set *name* is deleted.
	Program Start
QPname **A**	The program *name* is started.
	Program Stop
QPE	If the QPE Instruction is sent by the computer, it causes a jump back to the initial program level from which the program has been started.

Instruction	***Meaning***

Program Transmission with Read Out

QPname **S**byte

The program *name* is to be transmitted block wise. During the transmission, the whole control sequence must be observed.
name → maximal 16 characters,
byte → number of bytes to be transmitted

1. Computer:

<STX>Controller's addressQPname Sbyte**:CS<ETX>**
The program transmission sequence is started.

2. Controller response:

STX><ACK>O:CS<ETX>
if the program does not exist in the controller, and the controller's RAM capacity is sufficient.

<STX><ACK>V:CS<ETX>
K = RAM full
V = existent
If the program exists in the controller and must be overwritten.

How to overwrite:
The response to "V" is: **<STX> J:CS <ETX>** for overwrite or
<STX>N:CS<ETX> for stop (finish)

3. Program transmission:

- Start:
  <STX> Address block 1:CS<ETX>
  Block 1 is 256 byte long and starts with program name <ETB> .The program name must have 8 characters!
- Further blocks (block 1+x) always must have 256 bytes and are embedded in **<STX> Address block 1+x:CS<ETX**

Controller response after each block:

<STX><ACK>:CS<ETX>

Read Program with Request

QPname **R**

The program *name* is to be read from the controller unit. The program is to be read by blocks.

Request and send

1. Computer:
<STX> Address QPname **R:CS <ETX>**
The program *name* is to be read.

2. Controller response:
<STX><ACK>O:CS<ETX>
If the program is available, the controller unit reports the

Instruction	***Meaning***
	character O.
	3. Computer:
	<STX> Address J:CS <ETX> The computer receives the first block of the controller.
	4. Controller response:
	<STX>data program block x:CS<ETX>
	The points 3. and 4. are repeated as long as all blocks are received. The transmission is finished by appending 0x04(EOT) to the last row.
	Example for last row: **<STX>Data last block:CS<EXT>**

6.17 Registers

- The *phy***MOTION**™ controller contains a memory capacity of 1000 used to store variables, called **Registers** within *phy***LOGIC**™ programs.
- The registers are numbered R1 to R1000.
- In each register numbers digits type double can be entered.
- Write value into a register: **R**xx**=**zz
  Read value of a register: **R**xx**R**

  Explanations:

R	instruction code: register
xx	register number
S or **=**	Write (Schreiben)
zz	number (maximum 10 digits)

- Within the program registers can be used for indirect input of positions. Combined with arithmetic calculations registers can be used as counters during program run.
- Indirect Assignment
  The registers allow an indirect assignment. This means that the contents of register points to the address of another register to which can be accessed.
  An application e.g.: compare the numbers:
  Compare the contents of register R999 with the contents of the register content of R999:
  R[R999]==R999.
- For all logic combinations or arithmetic calculations with registers please notice:
  The computed value will always be written into the first register named in the instruction.

Example:	Add the values of two registers
R18+R2	Value of register 2 is added to value of register 18. The result will be stored in register 18.

Compare register values
As the result of a comparison, a condition byte will be set by the program:
E = condition fulfilled,
N = condition not fulfilled.
The status of the condition byte can be used for a conditional jump, subroutine instructions or other operations.

Example: R999==1 NE*one* N*la* Comparison of a register value with a number and conditional jump. If register 999 contains the value 1, jump to label*one*, if not, jump to label *la*.

6.18 Register Instructions

Rxx=value	Write value into register.
Rxx== value	Compare the value of the register and set the condition.
Rxx!= value	Compare the value of the register and set the condition.
Rxx>= value	Compare the value of the register and set the condition.
Rxx<= value	Compare the value of the register and set the condition.
Rxx++	Increase the content of 1
Rxx--	Decrease the content of 1

Instruction	***Meaning***
	Register Value Axes0
Rxx**SMA**	Read number of axis modules.
Rxx**SMA**x	Read number of axes per module. x=module → 1 to n
Rxx**SMAI**	Read number of analogue input modules.
Rxx**SMAIO**	Read number of analogue input and output modules.
Rxx**SMAO**	Read number of analogue output modules.
Rxx**SMDI**	Read number of digital input modules.
Rxx**SMDIO**	Read number of digital input and output modules.
Rxx**SMDO**	Read number of digital output modules.
	Register Value Integer
Rnn.z	The digits after the decimal point of the register nn are deleted without truncation of the value. z = 0 – 6 digits after the decimal point
	Set Outputs with Register Value
Rnn**BA**m**.**n**–**m**.**x	The content of the register nn is set as a binary value to the controller outputs n to x of the module m.

Instruction	***Meaning***
	Load Register with Input Status
Rnn**BE**m.n–m.x	The status of the inputs n to x is written as a binary value into the register nn.
	Example: R1BE1.1–1.8 → Input status: **1010 0101** Result: **165**
	Load Register with Hexadecimal Value
Rnn**BS**value	The register nn is set to the value. The data are fed in hexadecimal
	Example: R1BS1FA
	The register 1 is set to the hexadecimal value 1FA. After the instruction was carried out, the content of the register 1 is 506 decimal.
	Shift Register Bit by Bit
Rnn**BL**m	The content of the register nn is shifted the number of m digits to the left (MSB ←). The right side of the register is filled in with zero. m = 1 to 31 → maximal value of the register content.
	Example: R1S168 R1BL2
	The register 1 is set to the decimal value 168, corresponding to the binary value **10101000**. After the register content was shifted the number of 2 digits to the left, the binary value is **1010100000** which corresponds to the decimal value 672.
	Response: <STX><ACK>:CS<ETX>
Rnn**BR**m	The content of the register nn is shifted the number of m digits to the right (→ LSB). The left side of the register is filled in with zero.
	m = 1 to 31 → maximal value of the register content.
	Example: R1S168 R1BR2
	The register 1 is set to the decimal value 168, corresponding to the binary value **10101000**. After the register content was shifted (R1BL2) the number of 2 digits to the right, the binary value is **101010** which corresponds to the decimal value 42.
	Register Bit Check
Rnn**BT**m	The content of the register nn is regarded as a binary value. The digit in the position m of the binary value is checked. If the corresponding bit has been set, the condition byte is set. Otherwise the condition byte is reset.
	m = 0 to 31 → maximal value of the register content.
	Example: R1S168 R1BT4

Instruction ***Meaning***

The register 1 is set to the decimal value 168, corresponding to the binary value **10101000**. The Instruction R1BT4 checks the 4th digit from the right side (m ← LSB) of the binary value. The condition byte is set, because the 4th digit has the value 1.

Response: for PC: **<STX><ACK> E:CS <ETX>** or
<STX><ACK> N:CS <ETX>

for PROG: condition byte is set

Rnn**B**x**==**1
Rnn**B**x**==**0

The content of the register nn is compared with ‚1‘ or ‚0‘ (equal) and the condition byte is set.

x = 0 to 31 → bit position

Rnn**B**x**!=**1
Rnn**B**x**!=**0

The content of the register nn is compared with ‚1‘ or ‚0‘ (unequal) and the condition byte is set.

x = 0 to 31 → bit position

Logical Register Operations

Logic AND

Rnn**B^**value

A logical **AND** operation is carried out with the content of the register nn and the hexadecimal value.

Example: **R1BS2A8 R1B^1A0**

The register 1 is set to the hexadecimal value 2A8 (= 680 decimal). After the instruction **R1B^1A0** has been carried out, the content of the register 1 is 160 decimal.

	Decimal	Hex	Binary
	680	2A8	1010101000
	416	1A0	0110100000
Result:	160	0A0	0010100000

Rnn**B^R**mm

A logical **AND** operation is carried out with the content of the register nn and the content of register mm.

Logic OR

Rnn**Bv**value

A logical **OR** operation is carried out with the content of the register nn and the hexadecimal value.

Example: **R1BS2A8 R1Bv1A0**

The register 1 is set to the hexadecimal value 2A8 (= 680 decimal). After the instruction **R1Bv1A0** has been carried out, the content of the register 1 is 936 decimal.

Decimal	Hex	Binary
680	2A8	1010101000

Instruction	***Meaning***			
		416	**1A0**	**0110100000**
	Result:	**936**	**3A8**	**1110101000**

Rnn**BvR**mm

A logical **OR** operation is carried out with the content of the register nn and the content of register mm.

Response: <STX><ACK>:CS<ETX> (ONLY PC)

Logical Exclusive OR

Rnn**BX**value

A logical **XOR** operation is carried out with the content of the register nn and the hexadecimal value.

Example: R1BS2A8 R1BX1A0
The register 1 is set to the hexadecimal value 2A8 (= 680 decimal). After the instruction **R1BX1A0** has been carried out, the content of the register 1 is 776 decimal.

	Decimal	**Hex**	**Binary**
	680	**2A8**	**1010101000**
	416	**1A0**	**0110100000**
Result:	**776**	**308**	**1100001000**

Rnn**BXR**mm

A logical **XOR** operation is carried out with the content of the register nn and the content of register mm.

Compare Register Content with Number Values

Rnn**==**value

The content of register nn is compared with a number (value). The condition byte is set if equality has been detected. Otherwise it is reset.

Rnn**!=** value

The content of register nn is compared with a number (value). The condition byte is set if inequality has been detected. Otherwise it is reset.

Rnn**>** value
Rnn**>=** value

The content of register nn is compared with a number (value). The condition byte is set if the register value is higher or equal. Otherwise it is reset.

Rnn**<** value
Rnn**<=** value

The content of register nn is compared with a number (value). The condition byte is set if the register value is lower or equal. Otherwise it is reset.

Compare Register Content

Rnn**==R**mm

The content of register nn is compared with the content of register mm. The condition byte is set if equality has been detected. Otherwise it is reset.

Instruction	***Meaning***
Rnn**!=R**mm	The content of register nn is compared with the content of register mm. The condition byte is set if inequality has been detected. Otherwise it is reset.
Rnn**>R**mm **R**nn**>=R**mm	The content of register nn is compared with the content of register mm. The condition byte is set if the value of register nn is higher or equal. Otherwise it is reset.
Rnn**<R**mm **R**nn**<=R**mm	The content of register nn is compared with the content of register mm. The condition byte is set if the value of register nn is lower or equal. Otherwise it is reset.
	Response for all relations:
	for PC: **<STX><ACK> E:CS <ETX>** or **<STX><ACK> N:CS <ETX>**
	for PROG: condition byte is set
Rnn**B**x**=1**	Bit x in the register nn is set.
	x = 0 to 31 → bit position
Rnn**B**x**=0**	Bit x in the register nn is reset.
	x = 0 to 31 → bit position
	Arithmetical Register Operations
	Addition
Rnn**+**value	The value is added to the content of register nn.
Rnn**+R**mm	The content of register mm is added to the content of register nn.
Rnn**++**	The content of register nn is increased by „1“.
	Subtraction
Rnn**–**value	The value is subtracted from the content of the register nn.
Rnn**–R**mm	The content of register mm is subtracted from the content of the register nn.
	Multiplication
Rnn*****value	The content of register nn is multiplied by the value.
Rnn***R**mm	The content of the register nn is multiplied by the content of register mm.
	Division
Rnn**:R**mm	The content of register nn is divided by the content of register mm.
Rnn**/R**mm	The content of register nn is divided by the content of register mm.

Instruction	***Meaning***
	Trigonometric Functions
Rnn**SIN** **R**nn**COS** **R**nn**TAN**	Sine, Cosine or Tangent is evaluated from the value of the register nn and the result is written back to the register nn.
Rnn**ASIN** **R**nn**ACOS** **R**nn**ATAN**	The angle of Sine, Cosine or Tangent is evaluated from the value of the register nn and the result is written back to the register nn.
	Square Root
Rnn**QW**	The square root is evaluated from the value of the register nn and written back to the register nn.
	Power Function
Rnn**E**value	The power function is evaluated from the value of the register nn and written back to the register nn.
	Random Number
Rnn**RAND**	The register nn is set with the random number in the range 0 to 4294967296 (2^{32}).
	Read Register
Rnn**R**	The content of register nn is read (ONLY PROG).
	Response: <STX><ACK>value:CS<ETX>
	Response for all arithmetical operations: **<STX><ACK>:CS<ETX>** (ONLY PC)
	Write Register **with Decimal Values:**
Rnn**S**value	Register nn is set to the value.
	with Register Values:
Rnn**SR**mm	Register nn is set to the value of register mm.
	with Parameter Values
Rnn**S**m.a**P**y	Register nn is set to the value of the parameter y of the module m of the axis a.
	with the numerical value of the operator panel input
Rnn**ST**	Register nn is set to the input value of the operator panel BT5 AM
	with the Timer Ticker Value

Instruction	***Meaning***
Rnn**STT**	The register nn is written with the timer ticker value.
	Write Register via Inputs
Rnn**SE**mm–xx.k	A BCD value is written via the inputs mm to xx into the register nn. k = number of digits after the decimal point.
	Example: R1SE1–8.1
	The inputs 1 to 8 have e.g. the status: **1001 0011**. The result is 9.3.
Rnn**SEC**m	The counter value of the input module m is sent to the register n.

6.19 System Status (ONLY PC)

Instruction ***Meaning***

System Status General

S Axes check and request for the number of axes.

Response:<STX><ACK> n IO :CS <ETX>

n = number of axes

System Status Extended (for I4XM01 or I1AM01)

SEm.n Read axis status in hexadecimal code.

m = module number
n = number of axis

Response:
<STX><ACK>?????$d_Hd_Hd_Hd_Hd_Hd_Hd_Hd_H$:CS<ETX>

The response code is **decimal**. Convert the axis status in the hexadecimal code:

1 = Axis busy
2 = Command invalid
4 = Axis waits for synchronisation
8 = Axis initialised
10 = Axis limit switch +
20 = Axis limit switch –
40 = Axis limit switch center
80 = Axis limit switch software +
100 = Axis limit switch software –
200 = Axis power stage is busy
400 = Axis is in the ramp
800 = Axis internal error
1000 = Axis limit switch error
2000 = Axis power stage error

Instruction	***Meaning***
	4000 = Axis SFI error 8000 = Axis ENDAT error 10000 = Axis is running 20000 = Axis is in recovery time (s. parameter P13 or P16) 40000 = Axis is in stop current delay time (parameter P43) 80000 = Axis is in position 100000 = Axis APS is ready 200000 = Axis is positioning mode 400000 = Axis is in free running mode 800000 = Axis multi F run 1000000 = Axis SYNC allowed
SEC	Reset the status of all modules.
SECm.a	Reset the status of the IDX or I1AM module. m = module number a = axis number
	System Status Axes
SGn	The short status of the axis module is read. n → number of the axis module **Response:** **<STX><ACK>$d_H d_H d_H d_H d_H d_H d_H d_H$:CS<ETX>** The response code is **decimal**. Convert the axis status in the hexadecimal code: **Example:** **SG1** **<STX><ACK>00000005:CS<ETX>** Axis 1 and 3 of the axis module 1 (here: I4X module) are active.

Module	Byte	3		2		1		0	
	Bit	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
	Status	**n.n**	**n.n**	**Pre-register**	**Inter-polation**	**Limit switch –**	**Limit switch +**	**Axis error**	**Axis busy**
I4X	**Axis 1 is set**	0	0	1	1	1	1	1	1
	Axis 2 is set	0	0	2	2	2	2	2	2
	Axis 3 is set	0	0	4	4	4	4	4	4
	Axis 4 is set	0	0	8	8	8	8	8	8
	Total 4 axes	0	0	F	F	F	F	F	F
I1AM	**Axis1 is set**	0	0	0	0	1	1	1	1

SH Axis test with status axes output.

Response : <STX><ACK> E:CS <ETX>, if all axes are stopped.

<STX><ACK>N:CS <ETX>, if all axes are running.

System Status Decimal

ST Read system status as hexadecimal number.

Response : <STX><ACK>value:CS<ETX>
value = number between 0 and FFFF

0 = End of program in the LOCAL mode
1 = Program run
2 = Software Remote
4 = Limit switch of an axis
8 = Power stage failure of an axis
10 = Error programming (wrong instruction code or not allowed)
20 = Terminal activated
40 = Input scan is running
80 = Remote
100 = Axis module not available
200 = Axis not available
400 = I/O module not available
800 = I/O not available
1000 = Internal bus failure during data transfer
2000 = Module error on the internal bus
4000 = AIOM module not available
8000 = AIOM channel not available

Status Reset

STC Status is reset.

6.20 Synchronous Start

Instruction	***Meaning***
S1	Synchronic start der Achsen vorbereiten
S0	Synchronic start der Achsen ausführen
SC	Synchronic start Abbruch

6.21 Store Data into Flash EPROM

Instruction	***Meaning***
	Store programs and axis parameters (ONLY PC)
SA	Axis parameters are stored into the EPROM.

6.22 IO Status

Instruction	***Meaning***
SAIOCm	Reset the status of the AIOM module. m = Module number
SAIORm	Read the status of the AIOM module. m = Module number
SIOCm	Reset the status of the DIOM module. m = Module number
SIORm	Read the status of the DIOM module. m = Module number

6.23 Bluetooth

Instruction	***Meaning***
SB	Bluetooth Initialisation by “phymotion“ and password ”0000“
SB“aaaa“**;**xxxx	Bluetooth Initialisation by “aaaa“and password “xxxx“ xxxx → 0000 to 9999

6.24 Store Register Set

Instruction	***Meaning***
SRname;x**-**y	The register set is stored on the Flash memory with *name*. x = Start register y = End register y,x: 1 to 1000 **Example: SRtest;1-100** The registers 1 to 100 are saved with the name *test* on the Flash.

Instruction	***Meaning***
STRname**;**x**-**y	The text register set is steord on the Flash memory with *name*. x = first text register y = last text register y,x: 1 to 100 à 40 characters
SR=Rnn**;**a**-**e	The register set is stored on the Flash memory under the name from the register nn. a = Start register e = End register nn → from 1 to 1000 **Example: SR=R500;1-100** The registers 1 ro 100 are stored under the name from the register 500 on the Flash memory.
STR=Rnn**;**a**-**e	Text register set is stored on the Flash memory under the name from the register nn. a = Start register e = End register nn → from 1 to 1000

6.25 Touch Panel

6.25.1 Touch Panel Mode (ONLY PROG)

> **i** The touch panel interface can only be designed in the passive mode!
>
> Design elements are: switch, button (32 of each)
> label and other elements (50 of each)
>
> x→number of the design element; x=1 to 32 or 50

Instruction	***Meaning***
TA0	Switch the touch panel from the passive mode to the active mode
TA?	Check the touch panel mode 0 → active 1 → passive
TA=1	The MCM module asks the touch panel to switch to the passive mode.
TA==1	Query: passive mode active?
TA!=1	Query: Active mode active?

6.25.2 Touch Panel – Button

Instruction	***Meaning***
TBA?	Query information of all buttons
TBx**==1** **TB**x**!=1**	Query the state of the button x and set condition x → number of the element (max. 32)
TBCx=xpos;ypos;xsize;ysize;font size;alignment	Definition of the button xpos;ypos: button position in x- and y-direction xsize;ysize; button size in pixel Font size in pixel Alignment of the button label 0 → center 1 → left 2 → right
TBDx	Delete button x

Instruction	*Meaning*
TBEx=n	Allow button function x → button n → function n=0 to 2 0 → button is not visible (no function) 1 → button function is visible (no function) 2 → button function is visible and active
TBFx**“**yyyyyy**“**	Set RGB colour for the button y → RGB colour code (6 digits)
TBTx**“**text**“**	Label button with “text”
TB?x	Read the status of the button x 0 → button is not pressed 1 → button is pressed

6.25.3 Touch Panel – Switch

Instruction	*Meaning*
TCA?	Query information of all switches
TCA=Rxx	Set the status of all switches
TCx**==1** **TC**x**!=1**	Query the state of the switch x and set condition
TCCx**=** xpos;ypos;xsize; ysize;font size;alignment	Definition of the switch x xpos;ypos: switch position in x- and y-direction xsize;ysize; switch size in pixel Font size in pixel Alignment: 0 → center 1 → left 2 → right
TCDx	Delete switch x
TCEx**=**n	Set the function of the switch x x → switch n=0 to 2 0 → switch is not visible (no function) 1 → switch function is visible (no function) 2 → switch function is visible and active

Instruction	***Meaning***
TCFONx**“**yyyyyy**“**	Switch x „pressed“ is set with yyyyyy RGB colour
TCFOFFx**“**yyyyyy**“**	Switch x „not pressed“ is set with yyyyyy RGB colour
TCONx**“**text**“**	Switch x „pressed“ is labeled with “text”
TCOffx**“**text**“**	Switch x „not pressed“ is labeled with“text”
TC?x	The status of the switch x is read 0 → switch is not pressed 1 → switch is pressed
TCSx	Switch x is set to „pressed“
TCRx	Switch x is set to „not pressed“

6.25.4 Touch Panel - Dropbox

Befehl	***Bedeutung***
TDCx**=** xpos;ypos;xsize; ysize;font size;alignment	Definition of the dropbox x xpos;ypos: dropbox position x- and y-direction xsize;ysize; dropbox size in pixel Font size in pixel Alignment: 0 → center 1 → left 2 → right
TDDx	Delete dropbox x
TDGx	Active text x of the drop-down box is read
TDPx	Active position x of the text of the drop-down box is read
TDSx**=**n	Text of the position n of the drop-down box is set as active
TDTx“text“	Label dropbox x with „text“
TDTxIR	Fill dropbox x with directory of the registers
TDTxITR	Fill dropbox x with directory of the text registers
TDGx	Back to dropbox x the active text

6.25.5 Touch Panel – Text Display

Instruction	***Meaning***
TLCx= xpos;ypos;xsize; ysize; font size;alignment	Defnition of the label x xpos;ypos: label position x- and y-direction xsize;ysize; label size in pixel Font size in pixel Alignment: 0 → center 1 → left 2 → right
TLDx	Delete label x
TLFBKx“yyyyyy“	Set RGB colour for background x y → RGB colour code (6 digits)
TLFTXx“yyyyyy“	Set RGB colour for the font x y → RGB colour code (6 digits)
TLTx**“**text**“**	Label label x with “text”‘

6.25.6 Touch Panel – Input Field

Instruction	***Meaning***
TECx= xpos;ypos;xsize; ysize; font size;alignment,function	Definition of the input field x xpos;ypos: input field position in x- and y-direction xsize;ysize; input field size in pixel Font size in pixel Alignment: 0 → center 1 → left 2 → right Function: 0 → alphanumeric 1 → numeric
TEDx	Delete input field x
TETx**“**text**“**	Label input field x with “text”
TE?x	Read the status of the input field x 0 → not pressed 1 → pressed
TEGx	Return the text, written into the input field x.

Instruction	***Meaning***
LR=TEx	Read the input field x and write the value into the register
LTR=TEx	Load the register set with the name of the input field x from the flash

6.25.7 Touch Panel – Image

Instruction	***Meaning***
TICx**=** xpos;ypos;xsize;ysize	Image x definieren xpos;ypos: image position in x- and y-direction (position 0;0 is on the top left) xsize;ysize; image size in pixel
TIDx	Delete image x
TISx“name“	Set image x with content of „name“ Selection of name → LED_GN_OFF LED_GN_ON LED_YE_OFF LED_YE_ON LED_RD_OFF LED_RD_ON

6.25.8 Touch Panel – Progress bar

Instruction	***Meaning***
TPCx**=** xpos;ypos;xsize; ysize;scaling	Definition of the progress bar x xpos;ypos: progress bar position in x- and y-direction xsize;ysize; progress bar in pixel Maximum value: end value of the progress bar
TPDx	Delete progress bar x
TPPx=nn	Set value nn into the progress bar x

6.25.9 Touch Panel - Slider

Instruction	***Meaning***
TSCx**=** xpos;ypos;xsize; ysize;scaling	Definition of the slider x xpos;ypos: slider position in x- and y-direction xsize;ysize; slider size in pixel Maximum value: end value of the progress bar
TSDx	Delete slider x
TSPx=nn	Set value nn into the slider x
TSGx	Read slider value x

6.26 Group Instruction

Instruction	***Meaning***
TG= instruction1|instruction2|...	Send instruction in a instruction group **Example: TG=TBC1=10;10;50;50;16;0|TBF1"00FF00"** Define button 1 and set the button colour.. **Important:** does **not** apply to register values!

6.27 Text Registers

Instruction	***Meaning***
TRn**"**text**"**	Write the text register (text must be within"") n = Number of the text registers
TRn**R**	Read text register
TRn**C**	Delete text register

6.28 Time Loops

Instruction	***Meaning***
Tvalue **TR**nn **TR[R**nn**]**	The value for time loops (value, content of register nn or register [Rnn]) is preset in msec.

Instruction	***Meaning***
	The program waits here until the preset time has run out. **Response: <STX><ACK>:CS<ETX**
TTn**S**value **TT**n**SR**nn **TT**n**SR[R**nn**]**	The timer n is loaded with a time (ms) value (value, content of register nn or register [Rnn]). The timer counts down to zero. The program is not interrupted. n=0 to 10
TTn**==**0	The timer n is compared with the value 0. The condition byte is set if the timer value is zero. Otherwise it is reset. Timer = 0 means: the predetermined time is up. **Response: <STX><ACK>:CS<ETX>**
TTn!=0	The timer n is compared with the value 0. The condition byte is set if the timer value is not equal to zero. Otherwise it is reset. **Response: <STX><ACK>:CS<ETX>**
TTn**>**value **TT**n**>R**nn **TT**n**>R[R**nn**]** **TT**n**<**value **TT**n**<R**nn **TT**n**<R[R**nn**]**	The timer n is compared with „value“, the content of the register nn or register [Rnn]. The condition byte is set if the timer value is higher/lower than „value“, the content of register nn or register [Rnn]. Otherwise it is reset. **Response: <STX><ACK>E:CS<ETX>** or **<STX><ACK>N:CS<ETX>**
TT=y	Definition of the terminal type y; y=0 to 3 0 → no terminal 1 → BT5 2 →Touch panel at the terminal interface 3 → Touch panel at fieldbus interface (ETHS, Bluetooth,…)
TT==1 **TT!=1**	Query of the BT5 terminal status: **Response: <STX><ACK>E:CS<ETX> :** BT5 is active **or <STX><ACK>N:CS<ETX> :** BT5 is not active
TTR	Read the terminal type **Response: <STX><ACK>y:CS<ETX>** y=0 to 3 0 → no terminal 1 → BT5 2 →Touch panel at the terminal interface 3 → Touch panel at fieldbus interface (ETHS, Bluetooth,…)

6.29 Subroutines (ONLY PROG)

Instruction	***Meaning***
	Break Off Subroutine
UA	Break off all subroutines and set stack. The program can be continued with a jump instruction.
	End of Subroutine
UE	The subroutine is finished and the program is continued at the program row where this subroutine has been called.
	Call of Subroutine
U*la*	The subroutine starts at that row which is indicated by the label ***la***. The subroutine is ended by the instruction UE.
	Conditional Subroutine Call
	All instruction variants described above are available for the conditional subroutine call. The instruction call is only completed by the letter “E“ for condition fulfilled or “N“ for condition not fulfilled.
	"E" = Condition fulfilled
UE*la*	see **U***la*, page 54
UN*la*	see **U***la*, page 54

6.30 Axes Instructions

The following axis instructions **can** also be used starting with "**M**":

Examples: **M**m.a**R–** is equivalent to m.a**R–**
Mm.a**MA** is equivalent to m.a**MA**

In **indirect (register)** addressing the axis commands **must** begin with "**M**":

Example: **M**R[x].R[y]**+1000** means:

Content of R[x]= module number

Content of R[y]= axis number

Drive axis R[y] of the module R[x] by 1000 in +direction

Instruction	***Meaning***
m.a**C**	Reset the axis a of module m m= module 1 to n a= axis 1 to 4
	Response: <STX><ACK>:CS<ETX> (ONLY PC)
m.a**GP**	Write the default values of the parameters (axis a of the module m) (without saving!) m= module 1 to n a= axis 1 to 4
	Axis Status Request
m.a**==A**	Axis request within the ramp. The condition byte is set when the axis is within the ramp. Otherwise it is reset.
m.a**==E** m.a**!=E**	Axis request on power stage error. Check (**==**) if a power stage error has occurred or check (**!=**) if the power stage is operating normally. The error message “Failure” is requested.
m.a**==H** m.a**!=H**	Axis request on still stand. Check (**==**) if the axis is in standstill or check (**!=**) if the axis is in motion. The condition byte is set when the condition is fulfilled. Otherwise it is reset.

Instruction	***Meaning***
m.a**==I+** m.a**==I–** m.a**==IC** m.a**==IS+** m.a**==IS–**	Axis request on initiator status. The condition byte is set when the axis has come to a standstill at the initiator or the initiator is not connected. Otherwise it is reset. IC→ Limit switch center IS+→Software limit switch + IS–→Software limit switch – I+ →Limit switch + I–→ Limit switch –
m.a**==M** m.a**!=M**	Axis request on step failure error. Check power stage (**=**), if a **Step failure** has occurred or has not (**#**) occurred. The condition byte is set, when the condition is fulfilled. Otherwise it is reset. This instruction applies only to control units with optional **Encoder** board.
m.a**==N** m.a**!=N**	Axis request on emergency stop. Check (**==**) if the axis has come to a standstill (or not (**!=**)) at an emergency switch. The condition byte is set when the condition is fulfilled. Otherwise it is reset. **Response: <STX><ACK>E:CS<ETX>** or **<STX><ACK>N:CS<ETX>** (ONLY PC)
	Wait until Set Point is reached
m.a**>**value m.a**>R**nn m.a**>R[R**nn**]**	The axis m.a is positioned and the program waits until the value of the counter m.aP20 is higher than the preset value (value, content of register nn or register [Rnn]). If the m.aP20 value is higher or the axis has come to a standstill the program is continued. **Example: m.aP20S0 m.aP14S2000 m.a+10000** **m.a>5000 m.aP14S1000** **m.a>10000 m.aS m.aP14S2000** The axis is to be moved 10000 steps with 2000 Hz. After 5000 steps, the frequency is lowered to 1000 Hz and is set to 2000 Hz again after the standstill of the axis. At the instruction **m.a>5000** the program is stopped and will be continued after the position 5000 is reached or the axis has been stopped by an emergency stop.
m.a**<**value m.a**<R**nn m.a**<R[R**nn**]**	The axis m.a is positioned and the program waits until the value of the counter m.aP20 is lower than the preset value (value, content of register nn or register [Rnn]). If the m.aP20 value is lower or the axis has come to a standstill the program is continued. **Response: <STX><ACK>:CS<ETX>** (ONLY PC), if the axis has come to a standstill or the position condition is fulfilled. Otherwise the program waits.

Instruction	***Meaning***
	Switching Power Stages **Reset**
m.a**C**	The power stage of axis a of the module m is reset.
	Activate
m.a**MA**	The power stage of axis m.a is activated.
	Deactivate
m.a**MD**	The power stage of axis m.a is deactivated.
	Axis parameter
m.a**P**mm**R**	The parameter mm of axis m.a is read out.
	Response : <STX><ACK>value:CS<ETX> mm = Parameter ID (ONLY PC)
m.a**P**mm**S**value m.a**P**mm**SR**nn or m.a**P**mm**=**value m.a**P**mm**=R**nn m.a**P**mm**=R[R**nn**]**	The parameter mm of axis m.a is loaded with the preset value (value, the content of register nn or register [Rnn]). mm = Parameter ID
	Initialisation/Reference Search Run
	To initialize an axis; a reference search run has to be carried out. The initiators, also called limit switches, serve as reference point. The axis moves to an initiator. When the initiator signal is identified, the motor stops and moves as long in the opposite direction until there is no more initiator signal. In case of initiator offset setting the offset distance is run and the axis is stopped. This point is called MØP (mechanical zero point) or reference point.
m.a**R–**	The axis moves to the initiator of the — direction.
m.a**R–C**	The axis moves via the — initiator to the center initiator.
m.a**RC–**	The axis moves in — direction to the center initiator where the half of the distance is damped, the other half is free (see manual: Principles of Positioning chap. 5.4).
m.a**R+**	The axis moves to the initiator of the + direction.
m.a**R+C**	The axis moves to the center initiator via the + direction.
m.a**RC+**	The axis moves to the center initiator in + direction where the half of the distance is damped, the other half is free (see manual: Principles of Positioning chap. 5.4).

Instruction	***Meaning***
m.a**R–I**	The axis moves in — direction and stops with the zero pulse of the incremental encoder. Only incremental, no SSI, ENdat and BiSS encoders!
m.a**R+I**	The axis moves in + direction and stops with the zero pulse of the incremental encoder. Only incremental, no SSI, ENdat and BiSS encoders!
m.a**R–^I**	The axis moves to the initiator of the – direction. After the offset distance the axis moves again until the zero impulse of the **I**ncremental encoder stops the axis. Only incremental, no SSI, ENdat and BiSS encoders!
m.a**R+^I**	The axis moves to the initiator of the + direction. After the offset distance the axis moves again until the zero impulse of the **I**ncremental encoder stops the axis. Only incremental, no SSI, ENdat and BiSS encoders!
m.a**R-C^I**	The axis moves via the – direction to the center initiator. After the offset distance the axis moves again until the zero impulse of the **I**ncremental encoder stops the axis. Only incremental, no SSI, ENdat and BiSS encoders!
m.a**R+C^I**	The axis moves via the +direction to the center initiator. After the offset distance the axis moves again until the zero impulse of the **I**ncremental encoder stops the axis. Only incremental, no SSI, ENdat and BiSS encoders!
m.a**RC–^I**	The axis moves to the center initiator in — direction. After the offset the axis moves until the zero impulse of the encoder Only incremental, no SSI, ENdat and BiSS encoders!
m.a**RC+^I**	The axis moves to the center initiator in + direction. After the offset the axis moves until the zero impulse of the encoder Only incremental, no SSI, ENdat and BiSS encoders!
m.a**RCW**	The axis moves to the center initiator counterwise.
m.a**RCCW**	The axis moves to the center initiator counterclockwise.
m.a**RCW^I**	The axis moves to the center initiator counterwise. After the offset the axis moves until the zero impulse of the encoder Only incremental, no SSI, ENdat and BiSS encoders!
m.a**RCCW^I**	The axis moves to the center initiator counterclockwise. After the offset the axis moves until the zero impulse of the encoder Only incremental, no SSI, ENdat and BiSS encoders!
	Free Running
m.a**L**r	The axis is started and runs as long as it is stopped by the instruction m.aS or by a limit switch. r = + or – running direction

Instruction	***Meaning***
	Relative Positioning
m**.**arvalue m**.**ar**R**nn m**.**ar**R[R**nn**]**	The axis runs the distance relatively which is preset by value, the content of register Rnn or register [Rnn]. r = + or – running direction
	with stop instruction via input
m**.**arvalue**vE**m**.**nz m**.**ar**R**nn**vE**m**.**nz m**.**ar**R[R**nn**]vE**m**.**nz	The axis runs relatively the distance with the start/stop frequency P4 which is preset by value, the content of Rnn or register [Rnn]. The axis stops prematurely if the input nn gets the status z or a limit switch stops the positioning. r = + or – running direction z = S → input set z = R → input reset
m**.**arvalue**vvE**m**.**nz m**.**ar**R**nn**vvE**m**.**nz m**.**ar**R[R**nn**]vvE**m**.**nz	The axis runs relatively with P14 (ramp) the distance which is preset by value, the content of Rnn or register [Rnn]. The axis stops prematurely if the input nn gets the status z or a limit switch stops the positioning. r = + or – running direction z = S → input set z = R → input reset
	Absolute Positioning Related to the MØP
m**.**a**A**rvalue m**.**a**A**r**R**nn m**.**a**A**r**R[R**nn**]**	The axis runs, in relation to the mechanical zero point MØP (m.aP20) to the absolute position, which is preset by value, the content of Rnn or register [Rnn]. r = + or – running direction
	with stop instruction via input
m**.**a**A**rvalue**vE**m**.**nz m**.**a**A**r**R**nn**vE**m**.**nz m**.**a**A**r**R[R**nn**]vE**m**.**nz	The axis runs with start/stop frequency (P4) to the absolute position, which is preset by value, the content of Rnn or register [Rnn]. The axis stops prematurely if the input nn gets the status z or a limit switch stops axis run. r = + or – running direction z = S → input set z = R → input reset

Instruction	***Meaning***
m.a**A**rvalue**vvE**m.nz m.a**A**r**R**nn**vvE**m.nz m.a**A**r**R[R**nn**]vvE**m.nz	The axis runs with P14 (ramp) to the absolute position, which is preset by value, the content of Rnn or register [Rnn]. The axis stops prematurely if the input nn gets the status z or a limit switch stops axis run. r = + or – running direction z = S → input set z = R → input reset
	Axis Stop
m.a**S**	All running instructions are cut off. The axis stops with the preset ramp.
m.a**SN**	The axis stops with the preset emergency stop ramp (parameter P7).
	Axis control pulses (only for I4X)
	(values are stored in the MCM and are active after switching on)
m.a**TB**=value	The control pulses width for the external control pulses is set.
m.a**TBR**	The value of the control pulses width is read.
m.a**TE=**n	The external control pulses output of an axis is assigned and activated. n = 0 → off n = 1 → on Important: for temporary control pulses output is valid: n=0
m.a**TER=**n	The selected axis is read.
m.a**TD=**value	The control pulses divider for the external control pulses is set.
m.a**TDR**	The value of the divider is read.
m.avalue^**C**divider	Axis a is relatively moved and the control pulses are sent to the external control pulses. After positioning the external control pulses are switched off. value → distance in increments or mm divider = 1/(n+1) * control $\text{pulses}_{\text{input}}$ 0: 1 / (0+1) = 1 1: 1 / (1+1) = 1/2 2: 1 / (2+1) = 1/3 3: 1 / (3+1) = 1/4 4: 1 / (4+1) = 1/5 5: 1 / (5+1) = 1/6 etc.

6.31 Operator Panel BT5 AM Instructions

> **i** The functions can also be activated via interface, if the operator panel BT5 AM is connected.

6.31.1 Functions in the Sequence Program

Instruction	***Meaning***
TF=y	Call up the display function y y = HA (manual mode) y = PAR (parameter) y = REG (register) y = INI (initiator status) y = DI (digital input) y = DO (digital output) y = AI (analogue input) y = AO (analogue output) y = LREG (load register set from memory) y = LTREG (load text register set from memory)

6.31.2 View Data by the Operator Panel

Instruction	***Meaning***
TPx;p=**“**ASCII text**“**	View „ASCIItext“ in row x at position p x→row number x=1 to 4 p→position within a row p=1 to 20
TPx;p**M**n.m**P**yy	View the axis parameter Pyy of the axis n of the module m in row x at position p y→parameter number m → module number n → axis number x → row number x=1 to 4 p→ position within a row p=1 to 20
TPx;p**=R**nn	View contents of register n in row x at position p nn → register number x → row number x=1 to 4 p→ position within a row p=1 to 20

Instruction	***Meaning***
TPx;p**=R**nn**;**s	View content of register n with s decimal places in line x at position p x → line number x=1 to 4 p→ position within a line p=1 to 20 nn → register number s → number of decimal places **Response: <STX><ACK> :CS<ETX>**
TPx;p**=R**nn**;**s+"text"	Add „text" to the content of register n with s decimal places and view in line x at position p x → line number x=1 to 4 p→ position within a line p=1 to 20 nn → register number s → number of decimal places **Response: <STX><ACK> :CS<ETX>+"text"**

6.31.3 Clear Display

Instruction	***Meaning***
CT	The entire operator panel display is deleted via the interface.
CTn	Delete a single row. n--> row number n=1 to 4
CTn;m	Delete selected rows n or m-->row number n=1 to 4; m=1 to 4 **Response: <STX><ACK> :CS<ETX>**

6.31.4 Enter the Register

Instruction	***Meaning***
Rnn**ST**	Enter the register nn with the input of the operator panel BT5 AM. **Response: <STX><ACK> :CS<ETX>**

6.31.5 Key Query of the Operator Panel BT5 AM (also PC)

Instruction ***Meaning***

Condition key querys

#vFn When the function key n is pressed, the condition byte is set. If the button is not pressed, then the condition byte is reset.

n = function key F1 to F6

#vnmx When the function key n or m or x is pressed, the condition byte is set. If the button is not pressed, then the condition byte is reset.

n, m, x = 0 to 9 (key 0 ro 9)
n, m, x = L (key CURSOR LEFT)
n, m, x = R (key CURSOR RIGHT)
n, m, x = U (key CURSOR UP)
n, m, x = D (key CURSOR DOWN)
n, m, x = H (key CURSOR HOME)
n, m, x = B (key SCROLL)
n, m, x = C (key CLEAR)
n, m, x = E (key ENTER)
n, m, x = P (key PRINT)
n, m, x = ? (key ?)
n, m, x = + (key +)
n, m, x = – (key –)
n, m, x = . (key .)

Example: *wait* #vH1? NN*wait*
The keyboard of the panel is queried until
the key **H, 1** or **?** is pressed. The condition byte is reset, if the keys **HOME,1** or **?** are pressed.
It applies to the instruction **NN*wait*** : jumping to the row label *wait*

Response : <STX><ACK> E:CS<ETX> or
<STX><ACK> N:CS<ETX> (ONLY PC)

Important: The INPUT-key is not defined for a query.

6.31.6 Programmed Key Query (*phy*LOGICTM Instructions based on "C")

Condition comparison: == equal

!= unequal

#Fn and **#**n can be used in the C-queries.

#Fn == 1 // query function key n on condition equal

#Fn != 1 // query function key n on condition unequal

#n == 1 // query key n on condition equal

#n != 1 // query key n on condition unequal

Examples:

while(#F1!=1) {} // wait until the function key F1 is pressed

while(#F1==1) {} // wait until the function key F1 is released

7 *phy*LOGIC™ Commands based upon „C“

CAUTION – Programming error!

Malfunctions are possible by false instruction construction.

- The EXECUTION of an instruction must always be written **between** braces: i.e. do**{**execution**}** while (condition)

7.1 “if“ Command

If (condition) {execution}

If (condition && condition) {execution}

If (condition) {execution} else {execution}

If (condition) {execution} else if (condition) {execution}

Condition:	Register, inputs, outputs, axis status, timer interrogation
Condition comparison:	== equal
	!= unequal
	<= less equal
	>= greater equal
	> greater
	< less
Operation	&& AND operation
	|| OR operation

Examples: if(R10>100} { A1.1S}

if (R10>100 && R10<200)

{ A1.1S} else { A1.2S1}

if (R10>100 && R10<200)

{ A1.2S} else if (R10<100{ A1.1S}

7.2 “while“ Command

while (condition) {execution}

while (condition && condition) {execution}

Condition:	Register, inputs, outputs, axis status, timer interrogation
Condition comparison:	== equal
	!= unequal
	<= less equal
	>= greater equal
	> greater
	< less
Operation:	&& AND operation
	|| OR operation

Examples:

```
while(M1.1!=H) {}          // wait until the motors stops
while(M1.1!=H || M1.2!=H)
        { A1.1S A1.2S}
```

7.3 "do while" Command

do{execution} while (condition)
do{execution} while (condition && condition)

Condition:	Register, inputs, outputs, axis status, timer interrogation
Condition comparison:	== equal
	!= unequal
	<= less equal
	>= greater equal
	> greater
	< less
Operation:	&& AND operation
	|| OR operation

Example:

```
do {
        if(E1.1==S){              // input 1is set ?
          break;                  // if yes quit "while" loop
                  }
   } while(M1.1!=H)               // wait until the motors stops
       M1.1S                      // stop axis 1

   while(M1.1!=H || M1.2!=H)
   {A1.1S A1.2S}
```

7.4 “for“ Command

for (Value Initialising;value comparison;value manipulation){ execution }

for (Value Initialising;value comparison && value comparison;value manipulation){ execution}

Value:	Register
Initialising:	Rnn=value (value=number)
Comparison:	Rnn<xxx (xxx = number or register)
Manipulation:	Rnn++ all functions, which change the register content
Condition comparison:	== equal
	!= unequal
	<= less equal
	>= greater equal
	> greater
	< less
Operation:	&& AND operation
	|| OR operation

Example:

```
for (R1=1; R1<9;R1++)      // set register to 1; compare R1<9 ;R1+1
    { A1.R1 T100}          // switch on output 1 to 8
```

7.5 “break“ Command

„break“ aborts the loop in a “while”, “do while” and “for” loop.

7.6 “continue“ Command

„continue“ in a “while” or “do-while” loop jumps to the loop comparison.

„continue“ in a “for“ loop jumps to the manipulation value and then to the comparison value.

7.7 “goto“ Command

goto*label* : jump instruction to “label“

7.8 “switch“ Command

switch(Rnn){case x:break:default:}

„switch“ in the sequence program: query if the value of the register Rnn corresponds to x.

break: the case instruction is terminated.

default: the Rnn register is set to default.

Condition:	Register query
Condition comparison:	== equal

Example:

```
switch (R100){
        case 1:
                R1=1
                R101=1
                U*Test2*
```

```
    case 2:
        R1=2
        R101=6
        U*Test2*
        break;
    case 3:
        R1=3
        break
    case 4:
        R1=4
    default:
        break
}
```

8 List of *phy*LOGIC^{TM} Instructions

9 Parameters

For operating a stepper motor controller several presettings as speed, acceleration ramps or waiting time are required. These presettings are called **Parameters**.

Default parameters are stored which can be used in several applications at delivery. You can read and edit these parameters with *phy***LOGIC**™ToolBox.

Several counters are also contained in the list of parameters, which will be continuously actualized by the program. The counters can be read and some of them can be edited, too.

- For each axis separate parameters have to be set. Insert a module and axis number to mark the axis in front of the parameter number.

  Example: m.aP15 is the acceleration ramp value for axis m.a.

- Parameters (e.g. speeds) may be modified several times within a program, too.
- Parameter values can be entered or read.
- P49 can only be read.
- P19 to P22 are counters. They will be actualized by the program during axis movement.
- P27 to P54 are special parameters for the *phy***MOTION**™.
- Current values (P40 to P42) and P45 only apply to the INTERNAL power stages or power stages, which are connected via a bus:

	Supply	**Power stage module(s)**	**P45**
*phy***MOTION**™	EXTERNAL	INAM, I1AM01, I1AM02,…	as described in chap.9.1
		EXAM	without function; external power stage is set via DIP or rotary switch
	INTERNAL	integrated (MSX+, ZMX+,…)	invalid; Distribution of step resolution (0 to 15) according to the power stage table (see power stage manual)

9.1 List of Parameters

No.	Meaning	Default
P01	Type of movement (free run, relative / absolute, reference run) 0 = Rotational movement (ignoring limit switches) 1 = Hardware limit switches are monitored for XY tables or other linear systems, 2 limit switches: Mechanical zero and limit direction – Limit direction + 2 = Software limit switches are monitored 3 = Hardware and software limit switches are monitored	0
P02	Measuring units of movement: only used for displaying 1 = step 2 = mm 3 = inch 4 = degree	1
P03	Conversion factor for the thread 1 step corresponds to ... If P03 = 1 (steps) the conversion factor is 1. Computing the conversion factor: $Conversion\,factor = \frac{Thread}{Number\,of\,steps\,per\,revolution}$ <u>Example</u>: 4 mm thread pitch 200-step motor = 400 steps/rev. in the half step mode $Conversion\,factor = \frac{4}{400} = 0.01$	1
P04	Start/stop frequency The start/stop frequency is the maximum frequency to start or stop the motor without ramp. At higher frequencies, step losses or motor stop would be the result of a start or stop without ramp. The start/stop frequency depends on various factors: type of motor, load, mechanical system, power stage. The frequency is programmed in Hz.	400
P05 **P06**	not used	

No.	Meaning	Default
P07	Emergency stop ramp Input for I1AM0x: in 4000 Hz/s steps I4XM01: in 1 Hz/s steps	100 000
P08	f_{max} MØP (mechanical zero point) Run frequency during initializing (referencing) Enter in Hz (integer value) I1AM0x: 40 000 maximum I4XM01: 4 000 000 maximum	4000
P09	Ramp MØP Ramp during initializing, associated to parameter P08 Input for I1AM0x: in 4000 Hz/s steps I4XM01: in 1 Hz/s steps	4000
P10	f_{min} MØP Run frequency for leaving the limit switch range Enter in Hz	400
P11	MØP offset for limit switch direction + (away from “LIMIT+” switch, towards “LIMIT–” switch) Distance between reference point MØP and limit switch activation Unit: is defined in parameter P02 P11>=0	0
P12	MØP offset for limit switch direction – (away from “LIMIT–” switch, towards “LIMIT+” switch) Distance between reference point MØP and limit switch activation Unit: is defined in parameter P02 P12>=0	0
P13	Recovery time MØP Time lapse during initialization Enter in msec	20
P14	f_{max} Run frequency during program operation Enter in Hz (integer value) I1AM0x: 40 000 maximum I4XM01: 4 000 000 maximum	4000

No.	Meaning	Default
P15	Ramp for run frequency (P14) Input for I1AM0x: in 4000 Hz/s steps I4XM01: in 1 Hz/s steps	4000
P16	Recovery time position Time lapse after positioning Input in msec	20
P17	Boost (current is defined in P42) 0 = off 1 = on during motor run 2 = on during acceleration and deceleration ramp <u>Remarks:</u> The boost current is set in parameter P42 for internal power stages. You can select with parameter P17 in which situation the controller switches to boost current. P17 = 1 means, the boost current always is switched on during motor run. During motor standstill the controller switches to stop current.	0
P18	Internally used for linear interpolation	
P19	Encoder deviation MØP counter	
P20	Mechanical zero counter This counter contains the number of steps referred to the mechanical zero (MØP). If the axis reaches the MØP, P20 will be set to zero.	0
P21	Absolute counter Encoder, multi turn and also for single turn. The value of P22 is extended to P21 by software. The encoder counters have a fixed resolution, e.g. 10 bit (for single-turn encoders: the resolution is bits per turn), then the read value repeats. A saw tooth profile of the the numerical values is produced during a continuous motor running. This course is "straightened" by software. P20 and P21 will be scaled to the same value per revolution by P3 and P39 and are therefore directly comparable, see P36.	0

No.	Meaning	Default
P22	Encoder counter Indicates the true absolute encoder position. Is only set for A/B encoders to zero (after reset), the absolute encoder remains the value.	0
P23	Software Limit Switch (Axial limitation pos. direction +) If the distance is reached, the run in + direction is aborted. 0 = no limitation	0
P24	Software Limit Switch (Axial limitation neg. direction –) If the distance is reached, the run in – direction is aborted. 0 = no limitation	0
P25	Compensation for play Indicates the distance, the target position in the selected direction is passed over and afterwards is started in reverse direction. 0 = no compensation for play	0
P26	The data transfer rate is set by P26 (**ONLY** for SSI encoder), by which the encoder is read. The transfer rate is dependent on the length of the cable by which the encoder is connected to the device. The shorter the cable, the encoder can more quickly be read. Data transfer rate 1 to 10 (= 100 to 1000 kHz) 1 = 100 kHz 2 = 200 kHz 3 = 300 kHz 4 = 400 kHz 5 = 500 kHz 6 = 600 kHz 7 = 700 kHz 8 = 800 kHz 9 = 900 kHz 10 = 1000 kHz	1

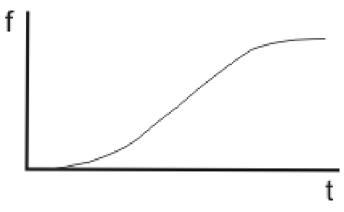
<table>
<tr><th>No.</th><th>Meaning</th><th>Default</th></tr>
<tr><td>P27</td><td>Limit switch type
NCC: normally closed contact
NOC: normally open contact
<table>
<tr><th></th><th>LIMIT–</th><th>Center/Ref</th><th>LIMIT+</th></tr>
<tr><td>0</td><td>NCC</td><td>NCC</td><td>NCC</td></tr>
<tr><td>1</td><td>NCC</td><td>NCC</td><td>NOC</td></tr>
<tr><td>2</td><td>NOC</td><td>NCC</td><td>NCC</td></tr>
<tr><td>3</td><td>NOC</td><td>NCC</td><td>NOC</td></tr>
<tr><td>4</td><td>NCC</td><td>NOC</td><td>NCC</td></tr>
<tr><td>5</td><td>NCC</td><td>NOC</td><td>NOC</td></tr>
<tr><td>6</td><td>NOC</td><td>NOC</td><td>NCC</td></tr>
<tr><td>7</td><td>NOC</td><td>NOC</td><td>NOC</td></tr>
</table></td><td>0</td></tr>
<tr><td>P28</td><td>Axis options
0 = Power stage is deactivated after power on
1 = Power stage is activated after power on</td><td>0</td></tr>
<tr><td colspan="3">P29 not used</td></tr>
<tr><td>P30</td><td>For I4XM01 only!
Frequency band setting
0 = manual
1 = automatic
<u>Remark</u>:
It is recommended to work with the automatic setting mode.
For each run frequency (P14) and ramp (P15) the controller automatically selects suitable settings.</td><td>1</td></tr>
<tr><td>P31</td><td>For I4XM01 only!
Frequency and ramp predivider (only if P30 = 0, manual)
This parameter changes the predivider which supplies the hardware (frequency generated) with a clock of 20 MHz derived.</td><td>3</td></tr>
</table>

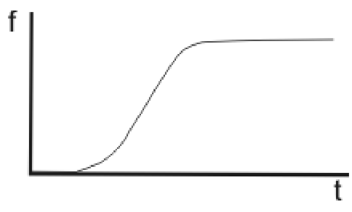
<table>
<tr><th>No.</th><th>Meaning</th><th>Default</th></tr>
<tr><td></td><td>
<table>
<tr><td>P31</td><td>Run frequency</td><td>resolution</td><td>predivider</td></tr>
<tr><td>0</td><td>1 Hz … 8 kHz</td><td>⅛ Hz</td><td>2440</td></tr>
<tr><td>1</td><td>1 Hz … 16 kHz</td><td>¼ Hz</td><td>1220</td></tr>
<tr><td>2</td><td>1 Hz … 32 kHz</td><td>½ Hz</td><td>609</td></tr>
<tr><td>3</td><td>1 Hz … 65 kHz</td><td>1 Hz</td><td>304</td></tr>
<tr><td>4</td><td>2 Hz … 130 kHz</td><td>2 Hz</td><td>152</td></tr>
<tr><td>5</td><td>4 Hz … 260 kHz</td><td>4 Hz</td><td>75</td></tr>
<tr><td>6</td><td>8 Hz … 520 kHz</td><td>8 Hz</td><td>37</td></tr>
<tr><td>7</td><td>16 Hz … 1 MHz</td><td>16 Hz</td><td>18</td></tr>
<tr><td>8</td><td>32 Hz … 2 MHz</td><td>32 Hz</td><td>9</td></tr>
<tr><td>9</td><td>64 Hz … 4 MHz</td><td>64 Hz</td><td>4</td></tr>
</table>
The parameter can be used for individual settings when automatic frequency band setting for the specific application is not appropriate.
</td><td></td></tr>
<tr><td>P32</td><td>Positioning ramp shape

0 = s-shape
1 = linear ramp

<u>Remark:</u> The s-shape ramp can be modified with P33 parameter.</td><td>1</td></tr>
<tr><td>P33</td><td>Arc value setting for s-shape ramp

Values: OMC: 1 to 8191
TMC: 1 to 32767



P33: low value


P33: high value</td><td>1</td></tr>
</table>

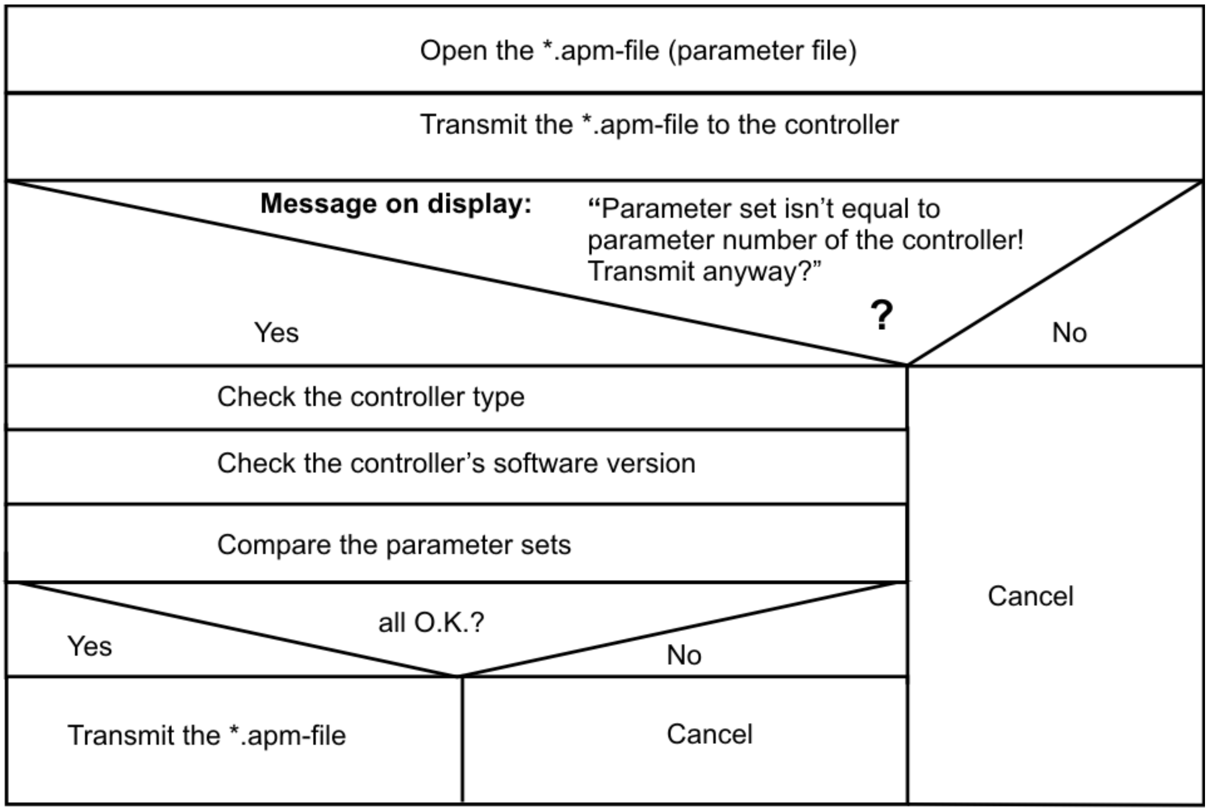
No.	Meaning	Default
P34	Encoder type 0 = no encoder 1 = incremental 5.0 V 2 = incremental 5.5 V 3 = serial interface SSI binary Code 5.0 V 4 = serial interface SSI binary Code 5.5 V 5 = serial interface SSI Gray Code 5.0 V 6 = serial interface SSI Gray Code 5.5 V 7 = EnDat 5 V 8 = EnDat 5.5 V 9 = resolver 10 = LVDT 4-wire 11 = LVDT 5/6-wire 12 = BiSS 5,0 V 13 = BiSS 24,0 V	0
P35	Encoder resolution for SSI and EnDat encoder Enter max. encoder resolution in Bit (max. 48 Bit) Special feature EnDat: if the parameter is set to zero, the controller uses the resolution which is read from the connected instrument.	10
P36	Encoder function This parameter specifies the use of P21 as a pure counter or whether its value is continuously compared with the value of the P20 counter, if the counter values vary too much, the motion is aborted with an error message. 0 = counter 1 = counter+SFI	0
P37	Encoder tolerance for SFI Enter tolerance value for SFI evaluation Input: tolerance value for SFI-evaluation in the selected resolution (P3 * P20). If P21 is used for step failure indication the scale of the counter P20 * P3 must be equal to the scale of the counter P21 * P39 and P21 must be set to zero after initialization of the scaling (or can be set to the same value as P20). e.g. scaling to 360°/rev.: Motor 200 steps per revolution, 1/20 step, →P3 = 360 / 200 / 20 = 0.09, encoder 10 bit / rev. → P39 = 360 / 2^{10} = 0.3515625	0

No.	Meaning	Default
P38	Encoder preferential direction of rotation 0 = + (positive) 1 = – (negative)	0
P39	Encoder conversion factor 1 increment corresponds to ... Computing the conversion factor: $Conversion\,factor = \frac{Thread}{Encoder\,steps\,per\,revolution}$	1
P40	Stop current in 0.01 $A_{r.m.s.}$ steps depending on the power stage I1AM01: 0 to 250 (0 to 2.5 $A_{r.m.s.}$) I1AM02: 0 to 350 (0 to 3.5 $A_{r.m.s.}$) ZMX^+: 0 to 630 (0 to 6.3 $A_{r.m.s.}$) MCD^+: 0 to 63 (0 to 6.3 $A_{r.m.s.}$) APS: 0 to 350 (0 to 3.5 $A_{r.m.s.}$) MSX52: 0 to 280 (0 to 2.8 $A_{r.m.s.}$) MSX102: 0 to 560 (0 to 5.6 $A_{r.m.s.}$) MSX152: 0 to 840 (0 to 8.4 $A_{r.m.s.}$)	2
P41	Run current in 0.01 $A_{r.m.s.}$ steps I1AM01: 0 to 250 (0 to 2.5 $A_{r.m.s.}$) I1AM02: 0 to 350 (0 to 3.5 $A_{r.m.s.}$) ZMX^+: 0 to 630 (0 to 6.3 $A_{r.m.s.}$) MCD^+: 0 to 63 (0 to 6.3 $A_{r.m.s.}$) APS: 0 to 350 (0 to 3.5 $A_{r.m.s.}$) MSX52: 0 to 280 (0 to 2.8 $A_{r.m.s.}$) MSX102: 0 to 560 (0 to 5.6 $A_{r.m.s.}$) MSX152: 0 to 840 (0 to 8.4 $A_{r.m.s.}$)	6
P42	Boost current in 0.01 $A_{r.m.s.}$ steps I1AM01: 0 to 250 (0 to 2.5 $A_{r.m.s.}$) I1AM02: 0 to 350 (0 to 3.5 $A_{r.m.s.}$) ZMX^+: 0 to 630 (0 to 6.3 $A_{r.m.s.}$) MCD^+: 0 to 63 (0 to 6.3 $A_{r.m.s.}$) APS: 0 to 350 (0 to 3.5 $A_{r.m.s.}$) MSX52: 0 to 280 (0 to 2.8 $A_{r.m.s.}$) MSX102: 0 to 560 (0 to 5.6 $A_{r.m.s.}$) MSX152: 0 to 840 (0 to 8.4 $A_{r.m.s.}$)	10
P43	Current hold time in msec	20

No.	Meaning	Default
P44	For I4XM01 only! Origin of the Control pulses for the axis 0 = 1:1 (Input=Output) 1 = from X 2 = from Y 3 = from Z 4 = from U 5 = from external	0
P45	Step resolution 1 to 512 0 = 1/1 step 7 = 1/16 step 1 = 1/2 step 8 = 1/20 step 2 = 1/2.5 step 9 = 1/32 step 3 = 1/4 step 10 = 1/64 step 4 = 1/5 step 11 = 1/128 step 5 = 1/8 step 12 = 1/256 step 6 = 1/10 step 13 = 1/512 step (e.g. APS01) Important: for I1AM: step resolution from 1/1 to 1/128 step P45 only applies to the INTERNAL power stages or power stages, which are connected via a bus (see chap. 9).	3
P46	not used	
P47	not used	
P48	not used	
P49	Power stage temperature in 1/10 °C	(read only)
P50	Divider for Control pulses only for I4XM01 Control pulses $_{Output}$=1/(n+1) * Control pulses $_{Input}$ 0 : 1/(0+1)=1 1: 1/(1+1)= 1/2 2: 1/(2+1) =1/3 3: 1/(3+1)=1/4 4: 1/(4+1)=1/5 5: 1/(5+1)=1/6	n=0
P51	Pulse width: (n+1)*100 ns only for I4XM01 n: 0….255 e.g. n=19: (19+1)*100 ns=2000 ns= 2µs -> F_{max}=1/(2*2 µs)=250 kHz	n=19
P52	Internally used for trigger position.	

No.	Meaning	Default
P53	Power stage monitoring 0 = off 1 = on	1
P54	Motor temperature in 1/10 °C -999999: Temperature module not existent -9999: negative overflow or temperature lower -220 °C at PT100 9999: positive overflow or temperature higher +390 °C at PT100	-999999 (read only)
P55	Motor temperature **warning** in 1/10 °C If the motor warmed up to a defined temperature value, a warning occurs. We recommend to operate the motor until it is cooled again.	0
P56	Motor temperature **shut-off** in 1/10 °C If the motor warmed up to a defined temperature value, the controller switches off and the power stage must be reset.	0
P57	Resolver voltage n=3…10 (Volt)	3
P58	Resolver ratio (ratio of primary to secondary winding) 0=1/8 1=1/4 2=1/2 3=1 4=2	2

9.2 Parameter Set Transmission to the Controller



10 Storing Programs, Parameters and Registers

Programs and parameters can be edited with *phy***LOGIC**™ ToolBox, transferred to the controller and stored. During program run registers and counters can be modified by the program.

Number of programs	512	
Program memory RAM	384 kB	Program is written in this RAM for running and then started.
Flash memory	4 MB	Storage of programs, register and text register sets
Register FRAM	1000	Registers survive a reboot.
Text register FRAM	100	

11 Index