Description of

the Command Set

Venus

as implemented in the

TANGO Controller

In der Murch 15
35579 Wetzlar
Germany
Tel.: +49/6441/9116-0
www.marzhauser.com

1. Table of Contents

2. Introduction

This manual is intended as technical reference for programmers of any software applications using the Tango controller driven with the so called command set Venus.
The Tango controller is configurable for several different command languages. This manual describes the command set for language "Venus for Tango", which is a superset of the little different interpreter languages Venus-1 and Venus-2. In general the fundamental command construction and functions are compatible to these former versions.

Valid examples:

cal<LF>	calibrate all enabled axes (Venus-1)
2<SP>ncal<LF>	calibrate axis 2 only (Venus-2)
status<LF>	ask for the status of the controller
1.2<SP>3.4<SP>move<LF>	move absolute x=1.2 and y=3.4 (e.g. [mm] other units are definable)

Please do not send more than 255 characters at once to the Tango Controller, as the input buffer will overflow. To avoid this it is recommended to request the status in between and wait for a value to be returned. Another solution is to activate the so called cts handshake (available in Desktop RS232 or USB versions). This will automatically halt the PC transmission for as long as the input buffer is full. The PC COM port then must be opened with hardware handshake on, too. Please refer the ***getcts*** and ***setcts*** commands for further description. Some of the commands are very often used and have abbreviations, to reduce communication time. Example: The command ***setjoyspeed*** is available as identical shortform ***js.***

Important: Security speed limitation!
The Tango controller has a built in security function, which reduces the maximum travel velocity to 10mm/s for as long as no initial ***cal*** and ***rm*** move is executed. This is to preserve the microscope stage from damage that could be caused by moving fast into its end positions. After calibrating the axis into its both limit switches, the travel velocity is not longer limited.
If this feature does not fit your purpose, the stage speed may be increased to up to 100mm/s at own risk. Please refer to the ***getsecvel*** and ***setsecvel*** command for further information.

3. Command and response syntax description

All commands and parameters which are sent to the controller, as well as all feedbacks of the controller, are transferred as a sequence of ASCII characters. The connection may be controlled manually at any time with any usual terminal program. The use of standard ASCII string communication simplifies also the tracing of the communication, since both directions "sent commands" and "received responses" are readable.
The controller does not distinguish between upper and lower case ASCII characters.
All floating point numbers may contain a decimal point but must not contain a comma.
Each single parameter or command may be terminated either with a carriage return <CR> or with a blank <SP>. The usage of <SP> as delimiter for a command line is unusual. Nevertheless this feature is implemented to be compatible with all existing Venus-1 or Venus-2 installations in the field.
The <ETX> is used as special single character command. On reception this character is processed in a high priority manner. It stops automatic moves immediately with a definable deceleration and also clears the receive buffer.

Symbol	Name	Decimal	Hexadecimal	Keyboard	Binary
<ETX>	End of Text	3	0x03	Ctrl-C	00000011
<LF>	Line Feed	10	0x0A		00001010
<CR>	Carriage Return	13	0x0D		00001101
<SP>	Space	32	0x20	Blank	00100000

The syntax for all commands (commands received by the controller) follows either the so called *host mode* or the *terminal mode*, while the response (sent from the controller) is terminated with the sequence <CR><LF> always.

host mode	[parameter]<SP>[axis]<SP>[command]<SP>
terminal mode	[parameter]<SP>[axis]<SP>[command]<CR>
response with one parameter	[parameter]<CR><LF>
response with two parameters	[parameter1]<SP>[parameter2]<CR><LF>

For readability of trace files the terminal mode is recommended.

3.1. mode

The command ***mode*** sets the operation mode of the controller.

```
Syntax:     mode
Parameter:  [0,1] (0 = host mode  or  1 = terminal mode)
Response:   none

Examples:
0 mode (set host mode)
1 mode (set terminal mode)
```

3.2. parameter

Some commands require no parameter, while other require more than one. The following chapters describe each command in detail.
Whenever a command requires an [axis] parameter to specify a certain axis, then the legal range is 1 to 4.
Some commands also allow the [axis] set to 0 or -1 for special access.

[axis] = index number	description
1	1st axis (e.g. name = X)
2	2nd axis (e.g. name = Y)
3	3rd axis (e.g. name = Z)
4	4th axis (e.g. name = A)
0	for access to some axis independent parameters
-1	possible for some commands to query all available axes

4. Stack Commands

This chapter need not to be read but could be useful to whomever it may concern. It describes the structure where the parameters are temporarily stored and all related commands.
Any integer or floating point numbers limited with a <SP> or <CR> are called parameters.
These parameters are written (pushed) onto a stack oriented memory. During command execution the parameters are read automatically from the stack. The sequence of writing and reading is done in the manner "last in, first out". This stack memory area is transparent to the user with the following commands.

4.1. gsp

This command returns the number of actual parameters present on stack. After execution of any Venus commands, the response to this ***gsp*** command should be zero. If this is not the case, this indicates that too many parameters have been sent with the command. This is the so called stack pointer.

4.2. pop

This command removes the last parameter from the stack. The stack pointer is decremented until zero.

4.3. clear

This command clears the stack and removes all parameters. The stack pointer is set to zero.

4.4. nclear

Same function as the clear command. Clearing a single axis stack is not supported.
This command clears the stack and removes all parameters. The stack pointer is set to zero.

5. Cross reference list Venus-1 vs. Venus-2

The following table contains a cross reference list for Venus-1 and Venus-2 commands. The Tango controller provides all these commands. In general the Venus-2 commands are designed to control a single axis, while the Venus-1 was first designed to control up to three axes simultaneously. Valid shorten commands are noted in round brackets.

Venus-1	Venus-2	Description
abort or CtrlC or <ETX>		abort automatic moves
move (m)	nmove (nm)	move absolute
rmove (rm)	nrmove (nrm)	move relative
pos (p)	npos (np)	
setpos (sp)		
getvel (gv)	getnvel (gnv)	get velocity
setvel (sv)	setnvel (snv)	set velocity
getaccel (ga)	getnaccel (gna)	get acceleration
setaccel (sa)	setnaccel (sna)	set acceleration
calibrate (cal)	ncalibrate (ncal)	calibrate
rangemeasure (rm)	nrangemeasure (nrm)	range measure
getlimit	getnlimit	get limit
setlimit	setnlimit	set limit
getcalvel	getncalvel	get calibration velocity
setcalvel	setncalvel	set calibration velocity
getrmvel	getnrmvel	get range measure velocity
setrmvel	setnrmvel	set range measure velocity
getjoyspeed	getnjoyspeed	get manual speed
setjoyspeed (js)	setnjoyspeed (njs)	set manual speed
version	nversion	get version

For Venus-1 style commands the parameters depend on actual value of ***getunit***. For all Venus-2 style commands the unit is either [mm] or [mm/s] or [mm/s²] as usual.

6. Communication Interface Settings

The Tango software provides different hardware connectors for communication. The commands in this chapter depend on the involved communication hardware. The following table shows if the command is working with a certain hardware interface protocol or has no effect.

	getbaud	setbaud	getcts	setcts
PCI-Bus	⃠	⃠	⃠	⃠
USB 2.0	⃠	⃠	✓	✓
RS232	✓	✓	✓	✓

6.1. getbaud

```
Syntax:     getbaud
Parameter:  none
Response:   {9600, 19200, 38400, 57600 or 115200}
Default:    57600
```

This command reads the serial communication transfer rate. The unit is bits per second [Bd]. However this command works only if the serial connection is already established.

6.2. setbaud

```
Syntax:     [value] setbaud
Parameter:  {9600, 19200, 38400, 57600 or 115200}
Response:   none
Example:    57600 setbaud
```

You may change the default baud rate (57600) of the controller to other values required by your application. However this command works only, if the serial connection is already established. This command will change the controllers baud rate only. Refer to the documentation of your controlling device (e.g. a PC) how to change the baud rate there. Next a ***save*** command may be sent to permanently store the new baud rate in the controller. For USB and PCI bus communication this command has no effect.

6.3. getcts

```
Syntax:     getcts
Parameter:  none
Response:   {0 or 1}
Default:    0
```

This command reads the current state of CTS. A "0" indicates CTS hardware handshake is disabled, which is the default. A "1" indicates CTS hardware handshake is enabled.

6.4. setcts

```
Syntax:     [value] setcts
Parameter:  {0 or 1}
Response:   none
Example:    0 setcts
```

This command enables or disables the hardware handshake.
Writing a 0 disables the CTS hardware handshake, which is also the default.
Writing a 1 enables the CTS hardware handshake for the controller (RS232 or USB) interface. You must not forget to open the PC COM port with same settings too. Next a ***save*** command may be sent to permanently store the new CTS hardware handshake in the controller. For PCI bus communication this command has no effect.

7. Controller Informations

The commands ***version***, ***nversion, identify, nidentify*** are provided for backward compatibility only.
The commands ***getserialno, getso, tango*** report details about hardware and firmware versions.

7.1. version

```
Syntax:     version
Parameter:  none
Response:   3.61
```

7.2. nversion

```
Syntax:     [axis] nversion
Parameter:  axis
Response:   3.61
Example:    1 nversion
```

7.3. identify

```
Syntax:     identify
Parameter:  none
Response:   Corvus 1 361 1 0
```

7.4. nidentify

```
Syntax:     [axis] nidentify
Parameter:  axis
Response:   Pegasus 1 143 0 0
Example:    1 nidentify
```

7.5. getserialno

```
Syntax:     getserialno
Parameter:  none
Response:   [YY][WW][V][A][nnn]
Example:    113513017
```

7.6. getstageno

```
Syntax:     getstageno
Parameter:  none
Response:      [JJ][MM][TT][nn]
            or [JJ][MM][TT][nnn]
            or -------- if no stage serial number available
Examples:   getstageno ==> 110831005
            getstageno ==> ---------
```

7.7. tango

```
Syntax:     tango
Parameter:  none
Response:   [version string] [CR,LF] [tango serial number]

Syntax:     Character string including controller type, firmware version,
            build date separated by a comma, and the serial number
            TANGO             Fixed string identifying the Tango controller
            -DT               Desktop version
            -PCI              PCI card version
            -S                Tango short card version (PCI-S, DT-S)
            -MINI             TANGOmini
            -C                Motorized Stage with integrated Controller
            e                 Tango PCI-E card version (PCIe, Dte)
            Version 1.37      Firmware version number
            Aug 12 2008       Firmware build date
            16:39:01          Firmware build time

Example:    TANGO-DT-S, Version 1.37, Aug 12 2008 , 16:39:01
            083513017
```

8. System Configuration

Many parameters can be stored permanently in the Tango Controller, so they are available after each consecutive power on. When stored once, this reduces initialization overhead of the application software. Refer to the "save" command for further information.

8.1. save

```
Syntax:     save
Parameter:  none
Response:   none
```

This command stores your favourite parameter settings (like acceleration and speed) in a permanent and safe data area (also called non volatile memory). These parameters will be taken by the controller after each consecutive reset or power on as default values.

8.2. restore

```
Syntax:     restore
Parameter:  none
Response:   none
```

This command overwrites the actual controller setting with parameters read from the non volatile memory.

8.3. reset

```
Syntax:     reset
Parameter:  none
Response:   none
```

The controller is forced to perform a software reset. It is a restart similar to power on. Rebooting from reset will take more than 1 second, where the controller is not responding. There is no reply to a software reset. So for knowing if the controller is rebooted and ready, it may be necessary to poll data until it responds again.

8.4. getpowerup

```
Syntax:     getpowerup
Parameter:  none
Response:   current power up mode
Example:    getpowerup (assume response 2 => "calibrate" and "joystick off")
```

The command ***getpowerup*** reads the current power up mode. The response is a bit combination:

Response		Description
	0	No power up functionality, last saved joystick mode is used
bit 2^0	1	Joystick on after power up or reset
bit 2^1	2	Calibration move after power up or reset
bit 2^2	4** (6)	Range Measure move after power up or reset, ** only to be used together with calibration option: cal+rm = 2+4 = 6
bit 2^3	8	Random move, combination with 2+4 required! (= 14)
bit 2^4	16	Performs Calibration and Range Measure, then travels to the zero position
bit 2^5	32	Enables closed loop after power up or reset

8.5. setpowerup

```
Syntax:     [value] setpowerup
Parameter:  required power up mode (refer table above for getpowerup)
Response:   none
Example:    1 setpowerup (enable joystick after power up)
            3 setpowerup (enable joystick an perform calibration after power up)
```

The command ***setpowerup*** causes the controller to automatically perform the desired behaviour on all active axes after power up or reset. Power up instructions may be combined, but not all combinations are allowed or make sense.
For a complete list of options please refer to the ***getpowerup*** description.

8.6. getipreter

```
Syntax:     getipreter
Parameter:  none
Response:   certainly 2 here only
```

This command reads the number of the actual selected interpreter. The response 2 indicates the Venus command set is set active as described in this manual.

8.7. setipreter

```
Syntax:     [value] setipreter
Parameter:  {1 or 2}
Response:   none
Example:    2 setipreter
```

This command selects the required interpreter.

Command	Description
1 setipreter	switch from Venus to native command set (refer separate manual)
!ipreter 2	switch from native to Venus command set (see this manual)

9. Motor Configuration

The controller is adaptable for different motors with the following set of commands.

9.1. gi

```
Syntax:     [axis] gi
Parameter:  none
Response:   motor current in unit Ampere [A]
Example:    2 gi (assume response 0.8 => 0.8 A motor current of axis 2)
```

This command reads the actual motor current in unit Ampere [A].

9.2. si

```
Syntax:     [value] [axis] si
Parameter:  value is motor current in [A]
Response:   none
Example:    0.8 1 si (set motor current 0.8 A for axis 1)
```

This command sets the required motor current. Please read the motor data sheet for details and do not cross the maximum ratings, since this may damage the motor permanently. (This command replaces setumotmin and setumotgrad.)

9.3. getpolepairs

```
Syntax:     [axis] getpolepairs
Parameter:  [axis]
Response:   integer
Example:    2 getpolepairs (reads axis 2 number of motor pole pairs)
```

The command ***getpolepairs*** reads the controller settings for the motor pole pairs of the asked axis.

9.4. setpolepairs

```
Syntax:     [value] [axis] setpolepairs
Parameter:  value is the number of the motor pole pairs (as an integer number)
Response:   none
Example:    50 1 setpolepairs (adapts axis 1 to a motor with 200 steps/rev)
```

The command ***setpolepairs*** adapts the controller to the required number of motor pole pairs for the requested axis. For clarification: A 2 phase stepper motor with 200 steps/rev has 50 pole pairs.

9.5. getmotiondir

```
Syntax:     [axis] getmotiondir
Parameter:  [axis] 1,2,3 or 4
Response:   0 = default motor direction
            1 = reversed motor direction
Example:    2 getmotiondir (read axis 2 direction)
```

The command ***getmotiondir*** reads the motor direction.

9.6. setmotiondir

```
Syntax:     [direction] [axis] setmotiondir
Parameter:  [axis]      1,2,3 or 4
            [direction] 0 = default motor/axis direction
                        1 = reversed motor/axis direction
Response:   none
Example:    0 2 setmotiondir (set axis 2 to default direction)
```

The command ***setmotiondir*** sets the motor direction of the specified axis. The the cal rm assignment is changed automatically.

10. Controller States and Error Messages

The two commands ***geterror*** and ***getstatus*** are useful to determine, if an unusual event has happened or the last command was executed. The response to ***ge*** and ***st*** reflects the actual controller state.

10.1. geterror (ge)

```
Syntax:     geterror or ge
Parameter:  none
Response:   error code
```

The command ***geterror*** reads the current controller error state, which indicates the last occurred error. The error message is deleted after the command is executed. Please refer chapter "Table of Error Numbers" for possible values and description.
This command can also be used as a blocking command in order to wait for completion of a move.

10.2. getnerror (gne)

```
Syntax:     [axis] getnerror or gne
Parameter:  axis
Response:   error code
```

The command ***getnerror*** reads the current controller error state, which indicates the last occurred error. The error message is deleted after the command is executed. Please refer chapter "Table of Error Numbers" for possible values and description.
This command can also be used as a blocking command in order to wait for completion of a move.
Implementation is of limited compatibility, no blocking for multiple axes, no error for a single axis available.

10.3. status (st)

```
Syntax:     status or st
Parameter:  none
Response:   actual controller status (integer in range of {0..511})
```

The command ***status*** reads the current controller status. The replied value reflects the operating state of the controller in a binary coded decimal representation. To decode the states correctly it is necessary to use a bit mask. Greyed status bits are currently not supported.

Bit	Decimal	0 indicates	1 indicates
D0	1	idle / move completed	controller executes a move
D1	2	manual mode (HDI) is not active	manual mode is active
D2	4	button A not pressed	button A pressed
D3	8	no function	no function
D4	16	speed mode is not active	speed mode is active
D5	32	position out of target window	position within the target window
D6	64	stop signal not active or not enabled	axes stopped by stop signal (setinfunc)
D7	128	motor driver is enabled	motor driver is disabled
D8	256	Joystick button not pressed	Joystick button pressed

10.4. nstatus (nst)

```
Syntax:     nstatus or nst
Parameter:  axis
Response:   actual axis status (integer in range of {0..255})
Example:    1 nst
```

The command ***nstatus*** reads the current axis status. The replied value reflects the operating state of the controller axis in a binary coded decimal representation. To decode the states correctly it is necessary to use a bit mask. Greyed status bits are currently not supported.

Bit	Decimal	0 indicates	1 indicates
D0	1	idle / move completed	axis executes a move
D1	2	manual mode (HDI) is not active	manual mode is active
D2	4	no machine error	machine error
D3	8	no function	no function
D4	16	no function	no function
D5	32	position out of target window	position within the target window
D6	64	stop signal not active or not enabled	axes stopped by stop signal (setinfunc)
D7	128	motor driver is enabled	motor driver is disabled

11. General Adjustments

With the following commands the parameters of the controller are widely scalable to the given mechanic construction and to customer requirements. The controller is adaptable to all the requested requirements.

11.1. getunit

```
Syntax:     [axis] getunit
Parameter:  axis
Response:   integer in range of {0..9}
Example:    2 getunit (read actual unit of axis 2)
```

The command ***getunit*** reads the unit of all input and output parameters related to length, e.g. position or move commands. Axis index = 0 (virtual) is used to read the unit of commands, which influence all axes (like setvel, setaccel, setmanaccel)

The possible values for unit are:

Value	unit
0	Microsteps
1	µm
2	mm
3	cm
4	m
5	inch
6	mil (1/1000 inch)
7	0..360°
8	
9	

We recommend to use physical units only. Remember: The unit micro steps is no physical unit, since it is well defined for a certain product only. Nevertheless the Tango is configurable to user defined number of micro steps/revolution (refer ***getusteps***). There is no need to modify existing applications but only to specify the required number of micro steps (refer ***setusteps***).

11.2. setunit

```
Syntax:     [value] [axis] setunit
Parameters: value in range of {0..9} description see table above
Response:   none
Example:    2 1 setunit (set unit [mm] for axis 1)
```

The command ***setunit*** sets the unit of all input and output parameters related to length, e.g. position or move commands. Axis index = 0 (virtual) is used here to specify the unit for commands like ***setvel***, ***setaccel***, ***setmanaccel*** which influence all axes.

11.3. getusteps

```
Syntax:     getusteps
Parameter:  none
Response:   integer [micro steps/revolution]
Example:    getusteps (read actual setting of micro steps)
```

The command ***getusteps*** reads the actual number of micro steps / revolution. The Tango default is 819200 micro steps per revolution (for 1.8° motors). In case you application is designed to work with a different value, you may adapt the Tango to your requirements.

11.4. setusteps

```
Syntax:     [value] setusteps
Parameters: value = required microsteps/rev.
Response:   none
Example:    40000 setusteps
```

The command ***setusteps*** defines the required value for microsteps. The Tango default is 819200 micro steps per revolution (for 1.8° motors). In case you application is designed to work with a different value, you may adapt the Tango to your requirements.

11.5. getdim

```
Syntax:     getdim
Parameter:  none
Response:   integer in range of {1..4}
Example:    getdim (read actual setting of value dimension)
```

The command ***getdim*** reads the actual setting of dimension of all input and output parameters related to length, e.g. position or move commands.

The possible values for dimension are:

Value	expected or replied axes parameters
1	[axis1]
2	[axis1] [axis2]
3	[axis1] [axis2] [axis3]
4	[axis1] [axis2] [axis3] [axis4]

11.6. setdim

```
Syntax:     [value] [axis] setunit
Parameters: value in range of {0..9} description see table above
Response:   none
Example:    2 1 setunit (set unit [mm] for axis 1)
```

The command ***setdim*** defines how many parameters the controller expects or replies for all dimension dependent commands only. The command ***setdim*** does not switch on or off any axes.

11.7. getpdisplay

```
Syntax:     [axis] getpdisplay
Parameter:  [axis]
Response:   [value1] (width of position response)
            [value2] (number of digits after decimal point)
Example:    2 getpdisplay (assume response 9 3 => pos format is "__345.789")
```

The command ***getpdisplay*** returns the actual position format. The response are two integer numbers. The first is the number of digits including the decimal point. The second is the number of digits after the decimal point.

11.8. setpdisplay

```
Syntax:     [value1] [value2] [axis] setpitch
Parameter:  [value1] is the number of all digits
            [value2] is the number of required digits after the decimal point
Response:   none
Example:    10 4 1 setpitch (set format f10.4 for pos response)
```

The command ***setpdisplay*** defines the format of the response to command ***pos***. The first parameter specifies the number of overall required digits including the decimal point. The second parameter specifies the number of digits after the decimal point.

11.9. getpitch

```
Syntax:     [axis] getpitch
Parameter:  [axis]
Response:   floating point number
Example:    2 getpitch (reads the pitch of axis 2)
```

The command ***getpitch*** reads the actual setting of pitch of the asked axis.

11.10. setpitch

```
Syntax:     [value] [axis] setpitch
Parameter:  value is spindle pitch (as a floating point number)
Response:   none
Example:    4.0 1 setpitch (set pitch to 4.0 [mm] for axis 1)
```

The command ***setpitch*** adapts the controller to the required spindle pitch of the requested axis.

11.11. getaccel (ga)

```
Syntax:     getaccel or ga
Parameter:  none
Response:   floating point number
Example:    ga (reads acceleration (identical value for all axes)
```

The command ***getaccel*** reads the general acceleration taken for automatic moves. The answer depends on the actual setting of ***unit***. Assume the command ***0 getunit*** responds with 2 then the unit of acceleration will be in [mm/s²].

11.12. setaccel (sa)

```
Syntax:     [value] setaccel or sa
Parameter:  value is acceleration (as a floating point number)
Response:   none
Example:    0.2 sa (sets acceleration to 0.2 (the unit depends on 0 getunit)
```

The command ***setaccel*** defines the acceleration for automatic moves. It is independent from direction. Any successive move will use this value during calculation. Nevertheless other parameters like e.g. spindle pitch are required to calculate correct. All calculations take care of vectorial move, e.g. the axes will follow a line from start point to end point. They will also reach the destination at the same time.

11.13. getnaccel (gna)

```
Syntax:     [axis] getnaccel or gna
Parameter:  axis
Response:   floating point number in [mm/s²]
Example:    2 gna (reads acceleration of axis 2)
```

The command ***getnaccel*** reads the acceleration of the specified axis taken for automatic moves. The unit is in [mm/s²].

11.14. setnaccel (sna)

```
Syntax:     [value] [axis] setnaccel or sna
Parameter:  value is acceleration (as a floating point number)
Response:   none
Example:    200 2 sna (sets acceleration for axis 2 to 200 [mm/s²]
```

The command ***setnaccel*** defines the acceleration for the specified axis for automatic moves. It is independent from direction. Any successive move will use this value during calculation. Nevertheless other parameters like e.g. spindle pitch are required to calculate correct. All calculations take care of vectorial move, e.g. the axes will follow a line from start point to end point. They will also reach the destination at the same time. When using Venus-2 move commands, the axis are also able to be positioned independent of each other.

11.15. getnaccelfunc

```
Syntax:     [axis] getnaccelfunc
Parameter:  axis
Response:   0 or 1
Example:    2 getnaccelfunc (reads acceleration function of axis 2)
```

The command ***getnaccelfunc*** reads the selected acceleration function of the specified axis for automatic moves.

11.16. setnaccelfunc

```
Syntax:     [value] [axis] setnaccelfunc
Parameter:  0 = linear accelerations
            1 = sin² acceleration (s-curve)
Response:   none
Example:    1 1 sa (sets acceleration for axis 2 to 0.2 [mm/s²]
```

The command ***setnaccelfunc*** defines the acceleration function for the specified axis for automatic moves.

11.17. getvel (gv)

```
Syntax:     getvel or gv
Parameter:  none
Response:   floating point number
Example:    gv (reads the velocity)
```

The command ***getvel*** reads the velocity. The answer depends on the actual setting of ***unit***. Assume command ***0 getunit*** responds with 2 then the unit of velocity will be in [mm/s].

11.18. setvel (sv)

```
Syntax:     [value] setvel or sv
Parameter:  value is the velocity for automatic moves (floating point number)
Response:   none
Example:    20 sv (sets the velocity 20, the unit depends on "0 getunit")
```

The command ***setvel*** defines the velocity for automatic moves. It is independent from direction. Any successive move will use this value. Please make sure the spindle pitch is also set correctly.
The HDI (joystick etc.) velocities must be set using setjoyspeed and setjoybspeed commands.

11.19. getnvel (gnv)

```
Syntax:     [axis] getnvel or gnv
Parameter:  axis
Response:   floating point number in [mm/s]
Example:    2 gnv (reads the velocity of axis 2)
```

The command ***getnvel*** reads the velocity of the selected axis. The `unit is [mm/s].`

11.20. setnvel (snv)

```
Syntax:     [value] [axis] setnvel or snv
Parameter:  value is the velocity for automatic moves (floating point number)
Response:   none
Example:    20 2 snv (sets the velocity 20 mm/s for axis 2
```

The command ***setnvel*** defines the velocity for automatic moves for the selected axis. It is independent from direction. Any successive move will use this value. When using Venus-2 nmove commands, the axis are also able to be positioned independent of each other.

11.21. getsecvel

```
Syntax:     getsecvel
Parameter:  none
Response:   Currently used secure velocity [1 to 100 mm/s]
Example:    getsecvel (read the secure velocity limitation)
```

The command ***getsecvel*** reads the security speed limitation. This value is used as limit for calculations until the axes are calibrated and range measured (***cal***, ***rm***). The velocity unit depends on the ***unit*** state. It prevents the microscope stage from mechanical damage as long as the controller does not know the mechanical limits.

11.22. setsecvel

```
Syntax:     [value] setsecvel
Parameter:  value is the secure speed limitation  as floating point
Response:   none
Example:    20 setsecvel (limit the axes speed to 20 mm/s)
```

The command ***setsecvel*** defines the secure speed limitation until ***cal*** and ***rm*** are executed. The velocity unit depends on the ***unit*** state. It prevents the microscope stage from mechanical damage as long as the controller does not know the mechanical limits.

11.23. getnsecvel

```
Syntax:     [axis] getnsecvel
Parameter:  axis index is 1,2,3,4 or -1 for all axes (depending on getdim)
Response:   Currently used secure velocity of the axis [1 to 100 mm/s] float
Example:    1 getnsecvel (read the secure velocity limitation of the X-axis)
```

The command ***getnsecvel*** reads the security speed limitation of the specified axis. This value is used as limit for calculations until the axes are calibrated and range measured (***cal***, ***rm***). The velocity unit is always mm/s and does not depend on the ***unit*** state. It prevents the microscope stage from mechanical damage as long as the controller does not know the mechanical limits.

11.24. setnsecvel

```
Syntax:     [value] [axis] setnsecvel
Parameter:  value is the secure speed limitation in [mm/s] as floating point
            axis index is 1,2,3 or 4
Response:   none
Example:    5.5 3 setnsecvel (limit the X-axis speed to 5.5 mm/s)
```

The command ***setnsecvel*** defines the secure speed limitation until ***cal*** and ***rm*** are executed of the specified single axis. The velocity unit is always mm/s and does not depend on the ***unit*** state. It prevents the microscope stage from mechanical damage as long as the controller does not know the mechanical limits.

11.25. getaxis

```
Syntax:     [axis] getaxis
Parameter:  axis index is 1,2,3,4 or -1 for all axes
Response:   current state of asked axis
Example:    2 getaxis (read the axis state of axis 2)
```

The command ***getaxis*** reads the current axis state. The response is either 1, 0 or -1.

Response	Description
1	enabled
0	disabled (with motor current on)
-1	disabled (with motor current off)

11.26. setaxis

```
Syntax:     [value] [axis] setaxis
Parameter:  value is either 1,0 or -1 (refer table above)
Response:   none
Example:    1 3 setaxis (enable axis 3)
```

The command ***setaxis*** enables, disables or switches off the selected axis.

12. Limit Switch commands (Hardware and Software)

The command set provides hardware limits as well as definable software limits for each axis. Hardware limits protect the axes.

12.1. getsw

```
Syntax:     [axis] getsw
Parameter:  axis index as usual
Response:   [cal switch (E0)] [rm switch (EE)]
Example:    2 getsw (assume response 0 0 => both are normal open)
```

response	description
0	switch type = normal open
1	switch type = normal close
2	hardware limit switch is ignored

The command ***getsw*** reads the actual settings for the limit switches of the requested axis.

12.2. setsw

```
Syntax:     [type] [switch] [axis] setsw
Parameter:  [type] see table above and [switch] see table below
Response:   none
Example:    2 1 3 setsw (ignore rm switch of axis 3)
```

[switch]	description
0	cal switch (E0)
1	rm switch (EE)

The command ***setsw*** sets the required type for the hardware limit switches.

12.3. getswst

```
Syntax:     [axis] getswst
Parameter:  axis index as usual
Response:   [cal switch (E0) status] [rm switch (EE) status]
Example:    2 getswst (assume response 0 1 => rm switch of axis 2 is crossed)
```

The command ***getswst*** reads the actual state of the hardware limit switches.

12.4. getlimit

```
Syntax:     getlimit
Parameter:  none
Response:   [lower limit] [upper limit]<CR><LF>
The number of responses depends on getdim.
Example:    getlimit (read the allowed positioning range)
```

The command ***getlimit*** reads the maximum allowed positioning range. Usually a ***cal*** and ***rm*** sequence is used to detect both hardware limits. If either only one or no limit switch is mounted, then the response will reflect the user definable software limits instead. The number of returned values depends on ***getdim***. The unit of the returned values depends on ***getunit***.

12.5. setlimit

```
Syntax:     [lower1] [upper1] {[lower2] [upper2] {[lower3] [upper3]}}setlimit
Parameter:  lower or upper software limit
Response:   none
Example:    -1000 1000 setlimit (assume getdim = 1 and getunit = 1)
```

The command s***etlimit*** defines the maximum allowed positioning range. This is useful if either only one or no limit switch is mounted. The number of expected values depends on ***getdim***. The expected unit of the transmitted values depends on ***getunit***.

12.6. getnlimit

```
Syntax:     [axis] getnlimit
Parameter:  axis
Response:   [lower limit] [upper limit]<CR><LF>
Example:    2 getnlimit (read the allowed positioning range of axis 2 in [mm])
```

The command ***getnlimit*** reads the maximum allowed positioning range of a single axis. Usually a ***ncal*** and ***nrm*** sequence is used to detect both hardware limits. If either only one or no limit switch is mounted, then the response will reflect the user definable software limits instead. The unit is [mm].

12.7. setnlimit

```
Syntax:     [lower] [upper] [axis] setnlimit
Parameter:  lower software limit, upper software limit, axis
Response:   none
Example:    -1000 1000 2 setnlimit
```

The command ***setnlimit*** defines the maximum allowed positioning range. This is useful if either only one or no limit switch is mounted. The unit is [mm].

13. Calibration and Range Measure commands

After each power on or ***reset*** command the actual position of all axes is assumed as zero. Also the travel speed is limited as defined by ***getsecvel*** until the axes are calibrated and range measured. This prevents mechanical hardware collisions, even if the limit switches are close to the mechanical end.
The user may run a calibration (***cal*** or ***ncal***) followed by a range measure (***rm*** or ***nrm***), if the system is equipped with the corresponding limit switches. The cal position becomes the new zero position. The speed during calibration is definable with ***setcalvel*** and ***setrmvel***.
If ***getcalswdist*** and ***getrmswdist*** are zero, then the cal and rm position is very close aside the limit switches. All end switch with low hysteresis (e.g. light beam switches and hall effect switches) usually require an extra position offset to prevent unintentional stop at the limits. This offset (e.g. 1mm) is user definable with the commands ***setcalswdist*** and ***setrmswdist***.

13.1. calibrate (cal)

```
Syntax:     calibrate or cal
Parameter:  none
Response:   none
```

This command moves all currently enabled axes in negative direction towards lower positions, until the calibration limit switch E0 is detected. It then moves in positive direction out of the switch. If calswdist=0, the axis will stop moving as soon as the limit switch E0 is released, and set the position to 0.0. If calswdist>0, the axes will continue moving until this distance, and then set the position to 0. The final position of each axis is also taken as its lower software limit.

13.2. ncalibrate (ncal)

```
Syntax:     [axis] ncalibrate or ncal
Parameter:  axis 1,2,3,4, or -1 to calibrate all active axes
Response:   none
Example:    2 ncal (calibrate axis 2)
```

This command moves the selected axis in negative direction towards lower positions, until the limit switch E0 is detected. It then moves it in positive direction out of the switch. If calswdist=0, the axis will stop moving as soon as the limit switch E0 is released, and set the position to 0. If calswdist>0, the axis will continue moving until this distance, and then set the position to 0. The final position is also taken as lower software limit.

13.3. rangemeasure (rm)

```
Syntax:     rangemeasure or rm
Parameter:  none
Response:   none
```

This command moves all currently enabled axes in positive direction towards higher positions, until the limit switch EE is detected. It then moves in negative direction towards lower positions. If rmswdist=0, the axes will stop moving, as soon as the limit switch EE is released. If rmswdist>0, the axes will continue moving until this distance. The final position is taken as upper software limit.

13.4. nrangemeasure (nrm)

```
Syntax:     [axis] nrangemeasure or nrm
Parameter:  axis 1,2,3,4, or -1 to rangemeasure all active axes
Response:   none
Example:    2 nrm (range measure axis 2)
```

This command moves the selected axis in positive direction towards higher positions, until the limit switch EE is detected. It then moves it in negative direction towards lower positions. If rmswdist=0, the axis will stop moving as soon as the limit switch EE is released. If rmswdist>0, the axis will continue moving until this distance. The final position is taken as upper software limit.

13.5. getcalswdist

```
Syntax:     [axis] getcalswdist
Parameter:  axis
Response:   actual calibration offset
Example:    2 getcalswdist (get cal offset position of axis 2)
```

This command reads the current calibration offset. The unit depends on the current value of command ***getunit***.

13.6. setcalswdist

```
Syntax:     [value] [axis] setcalswdist
Parameter:  value
Response:   none
Example:    0.3 2 setcalswdist (set 0.3 cal offset of axis 2)
```

This command specifies an extra offset position above the limit switch E0 (towards higher positions) where to zero the axis and take this position as lower software limit. The unit depends on the current value of command ***getunit***.

13.7. getrmswdist

```
Syntax:     [axis] getrmswdist
Parameter:  axis
Response:   actual range measure offset
Example:    2 getrmswdist (get rm offset position of axis 2)
```

This command reads the current range measure offset. The unit depends on the current value of command ***getunit***.

13.8. setrmswdist

```
Syntax:     [value] [axis] setrmswdist
Parameter:  value
Response:   none
Example:    0.3 2 setrmswdist (set 0.3 rm offset at axis 2)
```

This command specifies an extra offset position below the limit switch EE (towards lower positions) where to define the upper software limit. The unit depends on the current value of command ***getunit***.

13.9. getcalvel

```
Syntax:     getcalvel
Parameter:  none
Response:   value1<CR><LF> value2<CR><LF> floating point numbers in [mm/s]
Example:    getcalvel (get cal velocity and cal retract velocity)
```

This command reads the current calibration speeds. Value1 is the speed towards the lower limit switch E0. Value2 is the speed used during retraction from lower limit switch E0. There is only one pair of values for all axes.

13.10. setcalvel

```
Syntax:     [value] [index] setcalvel
Parameter:  floating point number in [mm/s] and index
Response:   none
Example:    10 1 setcalvel 0.5 2 setcalvel
```

This command defines the required calibration speeds. Index=11 sets the speed towards the lower limit switch E0. Index=2 sets the speed used during retraction from lower limit switch E0. There is only one pair of values for all axes.

13.11. getrmvel

```
Syntax:     getrmvel
Parameter:  none
Response:   value1<CR><LF> value2<CR><LF> floating point numbers in [mm/s]
Example:    getrmvel (get rm velocity and rm retract velocity)
```

This command reads the current range measure speeds. Value1 is the speed towards the upper limit switch EE. Value2 is the speed used during retraction from upper limit switch EE. There is only one pair of values for all axes.

13.12. setrmvel

```
Syntax:     [value] [index] setrmvel
Parameter:  floating point number in [mm/s] and index
Response:   none
Example:    10 1 setrmvel 0.5 2 setrmvel
```

This command sets the required range measure speeds. Index=1 sets the speed towards the upper limit switch EE. Index=2 sets the speed used during retraction from upper limit switch EE. There is only one pair of values for all axes.

13.13. getrefvel

```
Syntax:     getrmvel
Parameter:  none
Response:   value<CR><LF> 0.010000<CR><LF> floating point numbers in [mm/s]
Example:    getrmvel (get rm velocity and rm retract velocity)
```

This command reads the current range measure speeds. Value1 is the speed towards the reference mark. Value2 is a dummy value.

13.14. setrefvel

```
Syntax:     [value] [index] setrmvel
Parameter:  floating point number in [mm/s] and index
Response:   none
Example:    10.0 1 setrefvel
```

This command sets the required range measure speeds. Index=1 sets the speed towards the reference mark. Index=2 is unused.

13.15. getncalvel

```
Syntax:     [axis] getncalvel
Parameter:  axis
Response:   value1<CR><LF> value2<CR><LF> floating point numbers in [mm/s]
Example:    2 getncalvel (get cal velocity and cal retract velocity of axis 2)
```

This command reads the current calibration speeds. Value1 is the speed towards the lower limit switch E0. Value2 is the speed taken during retraction from lower limit switch E0.

13.16. setncalvel

```
Syntax:     [value] [index] [axis] setncalvel
Parameter:  floating point number in [mm/s]
Response:   none
Example:    10.0 1 3 setncalvel 0.5 2 3 setncalvel (cal velocities of axis 3)
```

This command defines the required calibration speeds. Index=1 sets the speed towards the lower limit switch E0. Index=2 sets the speed taken during retraction from lower limit switch E0.

13.17. getnrmvel

```
Syntax:     [axis] getnrmvel
Parameter:  axis
Response:   value1<CR><LF> value2<CR><LF> floating point numbers in [mm/s]
Example:    2 getnrmvel (get rm velocity and rm retract velocity of axis 2)
```

This command reads the current speeds used during range measurement. Value1 is the speed towards the upper limit switch EE. Value2 is the speed taken during retraction from upper limit switch EE.

13.18. setnrmvel

```
Syntax:     [value] [index] [axis] setnrmvel
Parameter:  floating point numbers in [mm/s]
Response:   none
Example:    10 1 3 setnrmvel 0.5 2 3 setnrmvel (rm velocities of axis 3)
```

This command sets the required speeds used during range measurement. Index=1 sets the speed towards the upper limit switch EE. Index=2 is the speed taken during retraction from upper limit switch EE.

13.19. getnrefvel

```
Syntax:     [axis] getncalvel
Parameter:  axis
Response:   value1<CR><LF> dummy<CR><LF> floating point numbers in [mm/s]
Example:    2 getnrefvel (get reference search velocity of axis 2)
```

This command reads the current calibration speeds. Value1 is the speed towards the reference mark. Value2 is a dummy value of 0.010000.

13.20. setnrefvel

```
Syntax:     [value] [index] [axis] setncalvel
Parameter:  floating point number in [mm/s]
Response:   none
Example:    10.0 1 3 setnrefvel (reference search velocity of axis 3)
```

This command defines the required calibration speeds. Index=1 sets the speed towards the reference mark. Index=2 is unused.

13.21. setpos (sp)

```
Syntax:     [value1] [value2] [value3] setpos (sp)
Parameter:  origin
Response:   none
Example:    0 0 0 setpos (assume dim = 3)
```

This command sets the actual position as origin. The number of expected arguments depends on ***getdim*** and the unit depends on ***getunit*** however.

14. Move commands

All move commands include an automatic linear interpolation. Axes which are started together are reaching the destination at the same time (vector). Axes can also be started independently from each other. In this case each axis drives with its own parameters and may not reach their target positions at the same time.
Blocking commands are available to generate an automatic position reached reply or execute multiple consecutive moves.

14.1. move (m)

```
Syntax:    [position1] [position2] [position3] move (m)
Parameter: target position
Response:  none
Example:   1.23 4.56 7.89 m  (dim=3: 3 axis absolute move to 1.23,4.56,7.89)
           1.23 4.56 m       (dim=2: 2 axis absolute move to 1.23,4.56,7.89)
           10 10 m ge        (use blocking command for completion reply)
           10 10 m 0 0 r st  (use blocking command for completion reply)
           1 1 m 0 0 r 5 1 m (use blocking command for consecutive moves)
```

This command moves the available axes to the specified absolute position.
The number of required arguments depends on ***getdim***.
The unit depends on the value of ***getunit***.

14.2. nmove (nm)

```
Syntax:    [value] [axis] nmove (nm)
           [value] [mask] nmove (nm)
Parameter: target positon of axis or axes
Response:  none
Example:   Single axis
           0.1 2 nm (absolute move axis 2 to 0.1 mm)
           100 3 nm (absolute move axis 3 to 100 mm)
           Bitmask Synchron move
           100 -5 nm (absolute move axis 1 and 3 to 100 mm)
           0.2 -3 nm (absolute move axis 1 and 2 to 0.2 mm)
```

This command moves the selected axis or axes to the specified absolute position in [mm].
Positive axis values move a single axis: 1,2,3,4
Combined negative values may be used to move multiple axes: -1, -2, -4, -8,
e.g. -3 moves axis 1 and 2, -6 moves axes 2 and 3

14.3. rmove (r)

```
Syntax:    [distance1] [distance2] [distance3] rmove (r)
Parameter: delta position
Response:  none
Example:   0.1 0.2 0.3 rmove (move relative for 3 axes controller: getdim=3)
           10 0 r            (move relative for 2 axes controller: getdim=2)
           0 0 0 r           (blocking command: wait for all 3 axes
                              before next command is executed)
```

This command moves one or more axes relative to their actual position.
The number of arguments depends on ***getdim***.
The unit of the input numbers depends on the current value of ***getunit***.

If called with distance 0, it blocks the command interpreter as long as the axes are travelling.

14.4. nrmove (nr)

```
Syntax:     [value] [axis] nrmove (nr)
            [value] [mask] nrmove (nr)
Parameter:  delta positon of axis
Response:   none
Example:    Single axis
            0.1 2 nrmove (relative move axis 2 by +0.1 mm)
            -10 2 nrmove (relative move axis 2 by -10 mm)
            Bitmask Synchron move
            100 -5 nrmove (relative move axis 1 and 3 by +100 mm)
            0.2 -3 nrmove (relative move axis 1 and 2 by +0.2 mm)
```

This command moves the selected axis or axes relative to their actual position in [mm].
It blocks the command interpreter when an addressed axis is still travelling.
Positive axis values move a single axis: 1,2,3,4
Combined negative values may be used to move multiple axes: -1, -2, -4, -8,
e.g. -3 moves axis 1 and 2, -6 moves axes 2 and 3

14.5. speed

```
Syntax:     [speed] [axis] speed
Parameter:  speed and axis
Response:   none
Example:    0.5 1 speed  (X axis travels forward at 0.5 mm/s)
            -10 2 speed  (Y axis travels backward at 10 mm/s)
              0 1 speed  (stop speed travel of X axis)
```

This command starts a per axis speed move, which travels at the desired speed until stopped.
Speed may be overwritten as desired and will change to the new value with the axis acceleration.
It can be stopped by the ***stopspeed*** or ***abort*** command or individually by setting the speed to zero.

14.6. stopspeed

```
Syntax:     stopspeed
Parameter:  none
Response:   none
Example:    stopspeed
```

This command stops the speed move off all axes.

14.7. randmove

```
Syntax:     randmove
Parameter:  none
Response:   none
Example:    randmove (starts random move on all active axes)
```

Randmove starts an endless random move on all active axes.
The position range for random moves can be set by the ***setlimit*** command or by performing a ***cal*** / ***rm*** move.
The velocity varies randomly from 50% to 100% of the selected axis velocity.

14.8. pos (p)

```
Syntax:     pos or p
Parameter:  none
Response:   Axes positions (depends on state of getunit, getdim, and getselpos)
```

This command reads the current axis positions. The number of values depends on the ***getdim*** command. The unit depends on the ***getunit*** command. The response is either the motor or the encoder position. To display the encoder position there is a ***cal*** or ***ncal*** required as precondition. Please see also ***getselpos*** and ***setselpos*** how to select which position should be reported.

14.9. npos (np)

```
Syntax:     [axis] npos or np
Parameter:  axis 1,2,3, or 4
Response:   Axis position (depends on getselpos)
```

This command reads the current position of the selected axis. The unit is [mm]. The response is either the motor or the encoder position. For encoder position there is a ***cal*** or ***ncal*** required as precondition. Please see also ***getselpos*** and ***setselpos*** how to select which position should be reported.

15. Joystick, Trackball and Handwheel commands

The so called HDI interface detects all manual input devices automatically. You are allowed to unplug, plug and exchange these input devices in a safe manner, e.g. the axes keep their position. So called "hot plug" is also provided: You need not to switch off the controller while changing the input devices.

15.1. getjoystick (gj)

```
Syntax:     getjoystick (gj)
Parameter:  none
Response:   0=disable, 1=enable
Example:    gj => 1 (joystick is enabled)
```

This command reads the manual joystick operation for all axes.

15.2. joystick (j)

```
Syntax:     [parameter] joystick (j)
Parameter:  [0,1] 0=disable, 1=enable
Response:   none
Example:    1 j (enable joystick operation)
```

This command enables or disables the manual joystick operation for all axes.

15.3. getnjoystick (gnj)

```
Syntax:     [axis] getnjoystick (gnj)
Parameter:  axis
Response:   0=disable, 1=enable
Example:    3 gnj => 1 (joystick of axis 3 is enabled)
```

This command reads the manual joystick operation.

15.4. njoystick (nj)

```
Syntax:     [parameter] [axis] njoystick (nj)
Parameter:  [0,1] 0=disable, 1=enable
Response:   none
Example:    0 2 nj (disable joystick operation for axis 2)
```

This command enables or disables the manual joystick operation for the specified axis.

15.5. getjoyassign

```
Syntax:     [axis] getjoyassign
Parameter:  axis
Response:   0,1,2,3,-1,-2,-3
Example:    2 getjoyassign => -2 (joystick direction of axis 2 is inverse)
```

This command reads the manual joystick axis directions.

15.6. setjoyassign

```
Syntax:     [parameter] [axis] setjoyassign
Parameter:  0 = joystick axis disabled
            1 = joystick axis assigned to axis 1, default direction
           -1 = joystick axis assigned to axis 1, inverse direction
            2 = joystick axis assigned to axis 2, default direction
           -2 = joystick axis assigned to axis 2, inverse direction
            3 = joystick axis assigned to axis 3, default direction
           -3 = joystick axis assigned to axis 3, inverse direction
Response:   none
Example:    0 3 setjoyassign (disable joystick operation for axis 3)
           -2 2 setjoyassign (inverse joystick direction for axis 2)
            1 1 setjoyassign (default joystick direction for axis 1)
            2 1 setjoyassign (not supported: assigning joy axis 2 to axis 1)
```

This command sets the direction and enables or disables joystick axes. Assigning joystick axes to a different controller axis is currently not supported.

15.7. getjoyspeed

```
Syntax:     getjoyspeed
Parameter:  none
Response:   actual maximum manual speed (unit as specified with 0 setunit)
Example:    getjoyspeed (returns maximum manual speed (unit as specified))
```

This command reads the actual maximum manual speed.

15.8. setjoyspeed (js)

```
Syntax:     [parameter] setjoyspeed (js)
Parameter:  maximum manual speed (unit as specified with 0 setunit)
Response:   none
Example:    20 js (set maximum manual speed to 20mm/s (assume 0 getunit => 2))
```

This command determines the maximum manual speed. This command affects all axes.

15.9. getjoybspeed

```
Syntax:     getjoybspeed
Parameter:  none
Response:   actual maximum manual speed if button is pressed
Example:    getjoyspeed (returns manual speed (unit as specified))
```

This command reads the actual maximum manual speed if button is pressed.

15.10. setjoybspeed

```
Syntax:     [parameter] setjoybspeed
Parameter:  maximum manual speed (unit as specified with 0 setunit)
Response:   none
Example:    2 setjoybspeed (set manual speed to 2mm/s (assume 0 getunit => 2))
```

This command determines the maximum manual speed if button is pressed.

15.11. getjoysticktype

Whenever a HDI device is connected to the controller, it is automatically detected. This command is implemented for compatibility purpose only.

15.12. setjoysticktype

Whenever a HDI device is connected to the controller, it is automatically detected. This command is implemented for compatibility purpose only.

15.13. getnjoyspeed

```
Syntax:     [axis] getnjoyspeed
Parameter:  axis
Response:   actual maximum manual speed of requested axis in [mm/s]
Example:    1 getnjoyspeed (responds maximum manual speed of axis 1 in mm/s)
```

This command reads the actual maximum manual speed of a single axis.

15.14. setnjoyspeed (njs)

```
Syntax:     [parameter] [axis] setnjoyspeed (njs)
Parameter:  maximum manual speed of requested axis in mm/s
Response:   none
Example:    20 2 njs (set maximum manual speed of axis 2 to 20mm/s)
```

This command determines the maximum manual speed of a single axis.

15.15. getkey

```
Syntax:     getkey
Parameter:  none
Response:   [F1] [F2] [F3] [F4] (0=not pressed or 1 = pressed)
Example:    response 1 0 0 1 indicates F1 and F4 were pressed since last query.
```

This command reads, if one or more of the buttons on the joystick panel were pressed since the last getkey command. The keys are named F1 to F4.

15.16. getwheelratio

```
Syntax:     [axis] getwheelratio
Parameter:  axis 1,2,3
Response:   travel distance per handwheel knob revolution in [mm]
Example:    2 getwheelratio (returns distance per rev. of Y axis)
```

This command reads the handwheel travel distance per knob revolution. Also refer to getwheelbratio, which reads the alternate travel distance which can be selected by a button on the device.

15.17. setwheelratio

```
Syntax:     [distance] [axis] setwheelratio
Parameter:  axis 1,2,3 and distance in mm
Response:   none
Example:    1 14.4 setwheelratio (set X travel distance to 14.4mm/revolution)
```

This command sets, the handwheel travel distance per knob revolution. Also refer to setwheelbratio, which sets the alternate travel distance which can be selected by a button on the device.
The maximum speed of traveling can be limited by the “js” or “njs” instruction.

15.18. getwheelbratio

```
Syntax:     [axis] getwheelbratio
Parameter:  axis 1,2,3
Response:   travel distance per handwheel knob revolution in [mm]
Example:    1 getwheelbratio (returns distance per rev. of X axis)
```

This command reads the handwheel travel distance per knob revolution. Also refer to getwheelratio, which reads the alternate travel distance which can be selected by a button on the device.

15.19. setwheelbratio

```
Syntax:     [axis] [axis] setwheelbratio
Parameter:  axis 1,2,3 and distance in mm
Response:   none
Example:    1 1.4 setwheelbratio (set travel distance to 1.4mm/revolution)
```

This command sets, the handwheel travel distance per knob revolution. Also refer to setwheelratio, which sets the alternate travel distance which can be selected by a button on the device.
The maximum speed of traveling can be limited by the “js” or “njs” instruction.

16. Digital and Analogue I/O

The Tango provides several digital I/O, two analogue outputs (channel 0 and 1) and one analogue input. These are available on the optional auxiliary I/O port.
The analogue output channel 2 is reserved for special purpose. Furthermore the HDI Interface analogue inputs may be read as well, if no HDI-device is connected.

16.1. setdac

```
Syntax:      [parameter] [channel-ID] setdac
Parameter:   [0..100] analogue output in [%] 100% = 10 Volt
Channel-ID:  [0..2] channel number
Response:    none
Example:     53.2 1 setdac (set channel 1 to 53.2% = 5.32 Volt
```

Channel No	Connector	Pin	Signal Name
0	AUX-IO	10	ANOUT0
1	AUX-IO	11	ANOUT1
2	reserved	-	-

This command sets the values for analogue outputs in percent. Fractional numbers may be used, too.

16.2. getaout

```
Syntax:      [channel-ID] getaout
Channel-ID:  [1 or 2] channel number
Response:    [0..10000] analogue output voltage in [mV]
Example:     1 getaout (returns analog output voltage of channel 1 in mV)
```

Channel No	Connector	Pin	Signal Name
1	AUX-IO	10	ANOUT0
2	AUX-IO	11	ANOUT1

This command reads the values for analogue outputs in mV.

16.3. setaout

```
Syntax:      [parameter] [channel-ID] setaout
Parameter:   [0..10000] analogue output voltage in [mV]
Channel-ID:  1 or 2] channel number
Response:    none
Example:     7500 1 setaout (set channel 1 to 7.5V)
```

Channel No	Connector	Pin	Signal Name
1	AUX-IO	10	ANOUT0
2	AUX-IO	11	ANOUT1

This command sets the values for analogue outputs in mV.

16.4. getnout

```
Syntax:     [axis] getnout
Response:   [integer number 0...15] digital signal level bitmask
Example:    1 getnout (returns an integer value representing the output state)
```

Bitmask Number	Output Name
1	OUT 0
2	OUT 1
4	OUT 2
8	OUT 3

This command reads the current output state of the 24V IO extension connector.
The axis parameter must be specified, but is not supported (dummy).
Only available with Tango PCI-E based controllers.

16.5. setnout

```
Syntax:     [bitmask] [axis] setnout
Parameter:  [integer number 0...15] digital signal level bitmask
Response:   none
Example:    10 1 setnout (set OUT1 and OUT3 to +24V)
```

Bitmask Number	Output Name
1	OUT 0
2	OUT 1
4	OUT 2
8	OUT 3

This command sets the output state of the 24V IO extension connector.
The axis parameter must be specified, but is not supported (dummy).
Only available with Tango PCI-E based controllers.

17. Encoder and Closed Loop Commands

The closed loop functionality requires encoder signals as precondition. These signals are automatically validated during each calibration. In case of valid encoder signals the user may use them as feed back for so called closed loop. If the closed loop mode is enabled, the system will use the encoder position as reference. If closed loop mode is disabled, the system will use calculated micro step position as reference.

17.1. getclperiod

```
Syntax:     [axis] getclperiod
Parameter:  axis
Response:   encoder period in [mm]
Example:    3 getclperiod => 0.02 (indicates 20 µm encoder period for axis 3)
```

This command reads the actual encoder period of the requested axis.

17.2. setclperiod

```
Syntax:     [parameter] [axis] setclperiod
Parameter:  [0..1] encoder period in [mm]
Response:   none
Example:    0.5 2 setclperiod (set axis 2 encoder period to 0.5 mm)
```

This command defines the actual encoder period for the requested axis.

17.3. getcloop

```
Syntax:     [axis] getcloop
Parameter:  axis
Response:   status of closed loop (0=OFF and 1=ON)
Example:    1 getcloop => 1 (indicates closed loop is enabled for axis 1)
```

This command reads the actual closed loop state of the requested axis.
A return value of 1 means the closed loop is enabled, but does not indicate an active closed loop:
Closed loop is activated by either a ***cal*** command or after reset/power up, depending on ***setpowerup***.

17.4. setcloop

```
Syntax:     [parameter] [axis] setcloop
Parameter:  [0..1] 0 = disable or 1 = enable the closed loop
Response:   none
Example:    1 2 setcloop (enable closed loop for axis 2)
```

This command enables or disables the closed loop mode for the requested axis.
Closed loop is then activated by either a ***cal*** command or after reset/power up, depending on ***setpowerup***.

17.5. getclfactor

```
Syntax:     [axis] getclfactor
Parameter:  axis
Response:   closed loop factor
Example:    2 getclfactor => 2000 (closed loop factor is 2000 for axis 2)
```

This command reads the closed loop factor of the requested axis.

17.6. setclfactor

```
Syntax:     [parameter] [axis] setclfactor
Parameter:  [100..10000] closed loop factor (independent from resolution)
Response:   none
Example:    1234 2 setclfactor (set closed loop factor 1234 for axis 2)
```

This command defines the required closed loop factor for the requested axis.

17.7. getselpos

This command reads, if the position readings with command ***pos (p)*** are from the actual encoder positions or calculated only.

```
Syntax:     [axis] getselpos
Parameter:  axis
Response:   position source
Example:    1 getselpos => 1 (axis 1 positions are measured encoder positions)
```

17.8. setselpos

```
Syntax:     [parameter] [axis] setselpos
Parameter:  [0,1] encoder position reading is 0=disabled 1=enabled
Response:   none
Example:    1 2 setselpos (enable axis 2 encoder position reading)
```

This command defines, if the position readings with command ***pos (p)*** shall be taken from the actual encoder position or calculated only.

17.9. getencamp

```
Syntax:     [axis] getencamp
Parameter:  axis
            -1 reads the amplitude of all enabled axes (dim)
             1 reads the amplitude of axis 1 (X)
             2 reads the amplitude of axis 2 (Y)
             3 reads the amplitude of axis 3 (Z)
Response:   encoder signal amplitude in percent
Example:    3 getencamp => 63 (axis 3 encoder amplitude at 63%)
           -1 getencamp => 79 81 63 (all encoder amplitudes when getdim=3)
```

This command reads the actual encoder signal amplitude of the requested axis.

17.10. getenc

```
Syntax:     [axis] getenc
Parameter:  axis
Response:   status of encoder (0=OFF and 1=ON)
Example:    2 getenc => 1 (indicates encoder is enabled for axis 2)
```

This command reads the actual encoder state of the requested axis.
Enabling the encoder manually makes it possible to access the measuring system position by ***1 setselpos / pos*** without entering the closed loop.

17.11. setenc

```
Syntax:     [parameter] [axis] setenc
Parameter:  [0,1] 0 = disable or 1 = enable the encoder
Response:   none
Example:    1 2 setenc (enable encoder for axis 2)
```

This command enables or disables the encoder for the requested axis.
Disabling the encoder also disables the closed loop, if active.
Enabling the encoder manually makes it possible to access the measuring system position by ***1 setselpos / pos*** without entering the closed loop.

17.12. getscaleinterface

```
Syntax:     [axis] getscaleinterface
Parameter:  axis 1,2,3
Response:   0 : No encoder interface present
            1 : Quadrature encoder interface (RS422 A/B-TTL)
            2 : Analog sin/cos interface (1Vss or MR depends on hardware)
Example:    2 getscaleinterface (read interface type of axis 2)
```

This command reads the encoder interface type of the specified axis.

17.13. setscaleinterface

```
Syntax:     [type] [axis] setscaleinterface
Parameter:  axis 1,2,3
            type 1 : Quadrature encoder interface (RS422 A/B-TTL)
                 2 : Analog sin/cos interface (1Vss or MR depends on hardware)**
Response:   none
Example:    1 3 setscaleinterface (interface of axis 3 is Quadrature RS422/TTL)
```

This command sets the encoder interface type of the specified axis.
** Analog interface type depends on order. It is always possible and recommended to configure quadrature encoders for an analog interface (here: 2-->1). But switching from quadrature to analog is only possible if analog functionality is configured by manufacturer.

17.14. getclwindow

```
Syntax:     [axis] getclwindow
Parameter:  axis
Response:   [window in mm] 0 [status] [signal]
            status = 0: status (st,nst) info bit D5 disabled
                   = 1: status (st,nst) info bit D5 enabled
            signal    : dummy value
Example:    2 getclwindow => 0.0010 0 1 0
```

This command reads the closed loop target window and status settings.

17.15. setclwindow

```
Syntax:     [window in mm] 0 [status] [signal] [axis] setclwindow
Parameter:  window = 0.0001 .. 1.0 mm closed loop position window
            status = 0: status, nstatus info bit D5 disabled
                   = 1: status, nstatus info bit D5 enabled
            signal    : dummy value
Response:   none
Example:    0.0001 0 1 0 1 setclwindow
           (axis 1 has a 100nm window, D5 status output of in window state)
            0.0020 0 0 0 2 setclwindow
           (axis 2 has a 2µm window, no D5 status output)
```

This command defines the closed loop position window and status settings for the requested axis.

17.16. getclwintime

```
Syntax:     [axis] getclwintime
Parameter:  axis
Response:   [time in seconds]
Example:    1 getclwintime => 0.100
```

This command reads how long the closed loop position must be in the target window.

17.17. setclwintime

```
Syntax:     [time in seconds] [axis] setclwintime
Parameter:  axis, time [0 … 1.000 s]
Response:   none
Example:    0.25 1 setclwintime
           (closed loop of axis 1 must remain in the position window for 250 ms)
```

This command defines how long the closed loop position must be in the target window.

18. Trigger Commands

The Tango controller can generate either position dependent or periodic signals to trigger external components, e.g. cameras, illumination, laser etc. Depending on the model, Tango controllers can generate up to two trigger signals. Here the second trigger output may be used to generate a precise delay (refer to settrout command). This feature is not available with all Tango controllers.
When using encoder signals as trigger position source, please make sure that the encoders are enabled and used by the Tango before executing settrpara. Usually encoders are activated after executing the calibrate command.

18.1. gettr

```
Syntax:     [axis] gettr
Parameter:  axis
Response:   trigger mode for specified axis
            0 = Trigger disabled
            1 = Scan-Trigger (positions)
            2 = Free running trigger (periodic signal)
Example:    1 gettr => 0 (trigger for axis 1 is disabled)
```

This command reads the trigger mode of the specified axis.

18.2. settr

```
Syntax:     [trigger mode] [axis] settr
Parameter:  trigger mode: 0 = Trigger disabled
                          1 = Scan-Trigger (positions)
                          2 = Free running trigger (periodic signal)
            axis
Response:   none
Example:    2 1 settr (set trigger mode to permanent frequency output)
```

This command defines the trigger mode for the specified axis.

18.3. gettrout

```
Syntax:     gettrout
Parameter:  none
Response:   trigger output mode:
            AUX-I/O Output |STANDARD|PREC.WIDTH2|PREC.DELAY2|PREC.FREQUENCY2|
            ---------------+--------+-----------+-----------+---------------+
            [no]           |    0   |    (4)    |    (8)    |      (12)     |
            ---------------+--------+-----------+-----------+---------------+
            TRIGGER_OUT    |    1   |    (5)    |    (9)    |      (13)     |
            ---------------+--------+-----------+-----------+---------------+
            TAKT_OUT       |    2   |     6     |    10     |       14      |
            ---------------+--------+-----------+-----------+---------------+
            TRIGGER+TAKT   |    3   |     7     |    11 **  |       15      |
            ---------------+--------+-----------+-----------+---------------+
            ** required for precise delay of secondary trigger output
Example:    1 gettrout => 11
            (both trigger outputs are used, output 2 is in precise delay mode)
            1 gettrout => 1
            (default trigger output mode on most Tango controllers)
```

This command reads the trigger output mode.

18.4. settrout

```
Syntax:     [trigger output mode] settrout
Parameter:  trigger output modes:
            AUX-I/O Output |STANDARD|PREC.WIDTH2|PREC.DELAY2|PREC.FREQUENCY2|
            ---------------+--------+-----------+-----------+---------------+
            [no]           |    0   |    (4)    |    (8)    |      (12)     |
            ---------------+--------+-----------+-----------+---------------+
            TRIGGER_OUT    |    1   |    (5)    |    (9)    |      (13)     |
            ---------------+--------+-----------+-----------+---------------+
            TAKT_OUT       |    2   |     6     |    10     |       14      |
            ---------------+--------+-----------+-----------+---------------+
            TRIGGER+TAKT   |    3   |     7     |    11 **  |       15      |
            ---------------+--------+-----------+-----------+---------------+
            ** required for precise delay of secondary trigger output
Response:   none
Example:    1 settrout (default mode for Tango controllers)
            3 settrout (secondary trigger output does the same as main trigger)
           11 settrout (secondary trigger signal after trdelay time)
```

This command defines the trigger output mode. Modes 4..7 and 12..15 are currently not available.

18.5. gettrpol

```
Syntax:     gettrpol
Parameter:  none
Response:   trigger polarity
            0 = falling edge, active low
            1 = rising  edge, active high (default)
Example:    gettrpol => 1
```

This command reads the trigger output polarity.

18.6. settrpol

```
Syntax:     [polarity] settrpol
Parameter:  0 = falling edge, active low
            1 = rising  edge, active high (default)
Response:   none
Example:    1 settrpol (set trigger polarity to active high)
```

This command defines the trigger output polarity.

18.7. gettrlen

```
Syntax:     gettrlen
Parameter:  none
Response:   trigger pulse length in Milliseconds [ms]
            0.000 ~ 25000.000
Example:    gettrlen => 0.040 (trigger pulse length is 40µs)
```

This command reads the trigger output signal length.

18.8. settrlen

```
Syntax:     [trigger pulse length] settrlen
Parameter:  0.00 ~ 25000 (in 0.04 ms steps)
Response:   none
Example:    0.04 settrlen (set trigger pulse length to 40µs)
            10 settrlen (set trigger pulse length to 10ms)
            0 settrlen (set pulse to shortest possible length)
```

This command defines the trigger output signal length.

18.9. gettrcount

```
Syntax:     gettrcount
Parameter:  none
Response:   Number of generated trigger events
            (since trigger enabled or resetted)
Example:    gettrcount => 100 (100 trigger pulses have been generated)
```

This command reads the trigger event counter.

18.10. settrcount

```
Syntax:     [count] settrcount
Parameter:  new trigger counter value, typically 0
            0 ~ 2147483647
Response:   none
Example:    0 settrcount (reset trigger counter)
```

This command sets the trigger event counter to the specified count.

18.11. gettrselpos

```
Syntax:     gettrselpos
Parameter:  none
Response:   Trigger source:
            0 = Motor position
            1 = Encoder position
Example:    gettrselpos => 0 (trigger uses internal motor position)
```

This command reads the trigger position source.

18.12. settrselpos

```
Syntax:     [position source] settrselpos
Parameter:  Trigger source: 0 = Motor position
                            1 = Encoder position (only with analogue encoders)
Response:   none
Example:    1 settrselpos (trigger uses encoder position)
```

This command defines the trigger position source.

IMPORTANT: When using encoder position (option 1), please make sure that the encoders are enabled and used by the Tango before executing settrpara. Else the Tango will use the motor position even if trselpos is set to 1.
Usually encoders are activated after executing the calibrate command.
Encoder position trigger is only available with analogue sin/cos encoders. In case of A/B TTL encoders, the trigger source 0 (motor position) must be used.

18.13. gettrdelay

```
Syntax:     [axis**] gettrdelay
Parameter:  axis
Response:   trigger signal delay for secondary trigger output in [µs]
Example:    1 gettrdelay => 1.5 (trigger delay is 1.5 µs)
```

This command reads the delay of the secondary trigger signal.
** The axis parameter must be used for compatibility, but is ignored internally (one delay applies to all axes).

18.14. settrdelay

```
Syntax:     [delay] [axis**] settrdelay
Parameter:  [0.0 .. 32500000.0]
            trigger signal delay for secondary trigger output in [µs]
Response:   none
Example:    10.5 1 settrdelay (set trigger signal delay to 10.5 µs)
```

This command defines the delay of the secondary trigger signal which is used, when settrout option 11 is set.
This function is only available with Tango PCI-E based controllers.
** The axis parameter must be used for compatibility, but is ignored internally (same delay for all axes).

18.15. gettrf

```
Syntax:     [axis] gettrf
Parameter:  axis
Response:   trigger frequency in [Hz]
Example:    1 gettrf => 10.000 (trigger frequency is 10 Hz)
```

This command reads the trigger frequency of periodic signal (for settr mode 2).
The axis parameter must be used for compatibility, but is ignored internally (one delay applies to all axes).

18.16. settrf

```
Syntax:     [frequency] [axis] settrf
Parameter:  [0.010 .. 12500.000]
            trigger frequency in [Hz]
Response:   none
Example:    1000 1 settrf (set trigger frequency to 1kHz)
```

This command defines the trigger frequency of periodic signal (for settr mode 2).
The axis parameter must be used for compatibility, but is ignored internally (same delay for all axes).

18.17. gettrpara

```
Syntax:     [axis] gettrpara
Parameter:  axis
Response:   [startpos] [endpos] [count]
            trigger parameter for specified axis
Example:    1 gettrpara => 10.0000 90.0000 5
           (5 equidistant trigger points from 10 to 90 mm: 10, 30, 50, ...)
```

This command reads the trigger position settings (for settr mode 1).

18.18. settrpara

```
Syntax:     [startpos] [endpos] [count] [axis] settrpara
Parameter:  startpos = position where first trigger is generated
            endpos   = position where last trigger is generated
            count    = number of equidistant trigger points **
            axis     = which axis to be used
Response:   none
Example:    10 110 11 1 settrpara (11 trigger positions in X from 10 to 110 mm)
            110 10 11 1 settrpara (same, but in reverse direction)
            5.553 7.553 21 2 settrpara
            (21 trigger positions in Y, trigger distance is 0.1µm)
```

This command defines equidistant trigger positions from a start- to an endpoint (when in settr mode 1).
The axis position must be in front of the trigger start position in order to charge the trigger.
Trigger direction is unidirectional. As long as the settr mode 1 is not changed, the trigger will recharge after returning to a position in front of the trigger start position.
As the first trigger is released at the start position, please consider that e.g. 10 to 20 will require 11 counts if a 1mm distance is required.

** The number of trigger points is limited. Depending on the firmware and controller type a maximum of 2047, 8191 or 10000 triggers is possible. The then may return with an error.

19. Wait Commands

19.1. waittime (wt)

```
Syntax:     [time] [unit] wt
Parameter:  time,
            unit: 0 = ticks of 0.25ms    (time =     1 ... 65000 ticks = 16.25s)
                  1 = seconds            (time = 0.001 ...   240 s)
Response:   --
Example:    1 1 waittime (1 Sekunde warten)
            2.5 1 wt     (2.5 Sekunden warten)
            2 0 wt       (0.5 ms warten)
```

This command prevents execution of the following commands for the specified time.
Two units are available: ticks (in 0.25ms) and seconds (resolution down to 1ms).

20. Safety Commands

20.1. abort (a)

```
Syntax:     abort
            or a
            or Ctrl+C (0x03 hex)
Parameter:  none
Response:   none
Example:    a
```

This command stops all axes. May not work with blocking commands.
In such case it is recommended to to send the abort character “Ctrl-C” (hex 0x03), which works independent of the command interpreter and also aborts blocking commands.

20.2. nabort

```
Syntax:     [axis] nabort
Parameter:  axis 1,2,3, or 4
Response:   none
Example:    2 nabort
```

This command stops individual axes. May not work with blocking commands.
In such case it is recommended to to send the abort character “Ctrl-C” (hex 0x03), which also aborts blocking commands but stops all axes.

20.3. getinfunc

```
Syntax:     [input] [axis] getinfunc
Parameter:  input = Digital input to which the functionality is assigned
            axis  = Axis to which the functionality is assigned
Response:   current functionality of the selected input (refer to setinfunc)
Example:    1 3 getinfunc ==> 3 (digital input #1 of Z-axis has functionality 3)
```

This command reads the stop signal configuration.

20.4. setinfunc

```
Syntax:      [function] [input] [axis] setinfunc
             [function] 0 0 setinfunc ***
Parameter:   *** The Tango AUX-I/O stop input is addressed by sending axis and
                 input as 0 (zeros). It is always assigned to all axes.

             input    = Digital input to which the functionality is assigned
             axis     = Axis to which the functionality is assigned
             function =

             -1 = clears a latched (sticky) stop condition

              0 = Stop function disabled
              1 = not available
              2 = not available
              3 = not available

             20 = stoppol  0 active low  \ stopped as long as signal applied
             21 = stoppol  1 active high / joystick operation remains enabled

             22 = stoppol  2 active low  \ stopped as long as signal applied
             23 = stoppol  3 active high / joystick operation disabled

             24 = stoppol  4 active low  \ stop latched until "-1 0 0 setinfunc"
             25 = stoppol  5 active high / joystick operation disabled

             28 = like 20, waits until any active move has completed
             29 = like 21, waits until any active move has completed

             30 = like 22, waits until any active move has completed
             31 = like 23, waits until any active move has completed

             32 = like 24, waits until any active move has completed
             33 = like 25, waits until any active move has completed

Response:    --
Example:     0 5 2 setinfunc  (disable stop function of Y-axis digital input #5)
             21 0 0 setinfunc (Tango: set the stop input to stoppol 1 mode)
             -1 0 0 setinfunc (Tango: clear a latched stop condition)
```

This command configures the stop signal functionality of the specified digital input and axis.

21. Dummy Instructions

Dummy instructions are implemented for compatibility only. They do not affect the controllers behaviour.

21.1. getmotorpara

```
Syntax:     [index] [axis] getmotorpara
Parameter:  index 1,2 or 3
            axis  1,2,3 or 4
Response:   index=3: [I²t limit] [current I²t value] as integer
Example:    3 1 getmotorpara (returns X-axis values, e.g. 40000 92)
            2 1 getmotorpara (returns X-axis values, e.g. 0.000000 0)
```

21.2. setmotorpara

```
Syntax:     [value] 3 [axis] setmotorpara
Parameter:  maximum I²t value
            axis index 1,2,3 or 4
Response:   --
Example:    40000 3 1 setmotorpara (set X-axis I²t limit to 40000)
```

21.3. getclpara

```
Syntax:     [axis] getclpara
Parameter:  axis index 1,2,3 or 4
Response:   [P] [I] [D] as floating point numbers
Example:    1 getclpara (returns 0 2 0.15)
```

21.4. setclpara

```
Syntax:     [P] [I] [D] [16383] [SP5] [SP6] [dpos] [ivel] [cutoff] [SP10] [np] [axis] setclpara
Parameter:  Closed loop parameter or subset of parameter defined by [np]
            np is 1 to 10, depending on the number of parameters sent
            axis index 1,2,3 or 4
Response:   --
Example:    0 5.3 0.001 3 1 setclpara (only set PID parameters of X-axis)
```

21.5. getsp

```
Syntax:     [SP index] [axis] getsp
Parameter:  SP Parameter index 1 to 10 as integer,
            axis 1,2 or 3
Response:   Specified parameter as float or integer, depending on parameter
Example:    2 1 getsp (returns closed loop parameter 2 = I param. for X-axis)
```

21.6. setsp

```
Syntax:     [value] [SP index] [axis] setsp
Parameter:  value as float or integer, depending on indexed parameter
            SP Parameter index 1 to 10 as integer,
            axis 1,2,3 or 4
Response:   --
Example:    0.15 3 2 setsp (set closed loop D parameter of Y-axis to 0.15)
```

22. Table of Error Numbers

The response to command ***geterror (ge)*** is an error number. The following table contains a description of these error numbers and a description of the possible root cause.

Error Number	Description
0	no error
1001	wrong parameter type
1002	not enough parameters for the command on the stack
1003	invalid parameter
1004	move stopped working range should run over
1007	invalid parameter value
1008	stack is empty (or contains not enough parameters)
1009	stack is full
1015	parameter outside allowed limits
1020	arc tan
1027	
1029	
1100	division through zero
1200	non volatile memory write error
1234	error during calibration
2000	unknown command

Please refer command ***geterror (ge)*** for further details.

23. Document Revision History

No.	Revision	Date	Changes	Remarks
01		17. June 2008		birthday
	A	23. February 2010	Description corrected: setcalvel, setrmvel, setncalvel, setnrmvel Added commands: setrefvel, getrefvel, setnrefvel, getnrefvel	Firmware 1.51
	B	07. April 2010	Extended description of geterror	
	C	20. July 2010	New MW Logo	
	D	22. July 2010	Added commands: getwheelratio, setwheelratio, getwheelbratio, setwheelbratio,	Firmware 1.521
	E	29. July 2010	Description of status (st) corrected, Added commands nstatus (nst), setclwindow, getclwindow, setclwintime, getclwintime, getnaccelfunc, setnaccelfunc, nclear, getjoystick (gj), njoystick (nj), getnjoystick (gnj), getnerror (gne)	Firmware 1.522
	F	26. August 2010	Added commands getaout, setaout, changed document title from “MST Dokument” to “Tango Instruction Set Venus”	Firmware 1.528
	G	07. October 2010	Added trigger commands: gettr,settr, gettrout,settrout, gettrselpos,settrselpos, gettrdelay,settrdelay, settrf,getttrf, gettrpara,settrpara	Firmware 1.534
	H	28. October 2010	setpowerup example corrected, getpowerup description nmove, nrmove with bitmask, extended powerup, randmove	Firmware 1.543
	I	01. December 2010	Added commands: getjoyassign, setjoyassign, tango	Firmware 1.561
	J	10. February 2011	Improved description of trigger functionality	
	K	15. February 2011	gettrpol, settrpol, gettrlen, settrlen, gettrcount, settrcount	Firmware 1.5682
	L	17. February 2011	Added waittime (wt), improved description of move	Firmware 1.5691
		04. March 2011	Added speed, stopspeed, nabort, improved abort description	Firmware 1.56
	M	08. March 2011	Added setenc, getenc	Firmware 1.57
			Added getencamp	Firmware 1.57
		10. March 2011	Added getscaleinterface, setscaleinterface	Firmware 1.57
		12. August 2011	Added getnsecvel, setnsecvel, Added dummy instructions: getmotorpara, setmotorpara, getclpara, setclpara, getsp, setsp	Firmware 1.57
		01. Sept. 2011	Added getinfunc, setinfunc, getstageno getserialno now reports Tango S/N	Firmware 1.57
		11. October 2011	Added getmotiondir, setmotiondir	Firmware 1.57
		14. December 2011	Added setnout, getnout	Firmware 1.57