Handbook

hydra

Firmware: 2.203000
GMT: Fri Apr 16 09:36:39 2010

MICOS GmbH 04/2010

Contents

About this documentation

The interpreter language Venus-3 of the controller hydra is described in this manual. Their function and syntax is explained.

The grouping of the commands in function groups improves the overview, an index table and alphabetical command list gives additional assistance.

The mechanisms of the command execution are commented in an introduction. To study this chapter is important as the way of procedure for the correct programming is explained there.

As far as it is necessary to comprehend the correlations, hardware specific peculiarities of the controller are also explained. These attributes are described in greater detail in the manual of the controller.

Introduction to Venus-3

Venus-3 is an interpreter language and combines the languages Venus-1 and Venus-2

Venus-3 commands consist of ASCII-characters which are interpreted in the controller and immediately executed.
A software development surrounding to produce the control programs is not needed.
The commands can be produced by any host and whatever programming language you are using, on condition that there is an access to the RS-232 interface or ethernet interface.
In the simplest way the commands are directly transmitted to the controller via an ASCII terminal.

History

Venus-3 has been developed on the basis of the interpreter language Venus-1 and Venus-2. The fundamental command construction is identical.
The expansion was necessary as the fundamental structures of Venus-1 are designed for controller with at most three linear interpolated axes; but the controller Pegasus supports any number of independent axes (n). Venus-1 commands which structure is designed for operating n-axes are taken over in Venus-2 without any syntax alteration. Venus-3 combines the two languages in most cases.
Special commands for a single axis are expanded with the addition "n" before the command name.
For example: The Venus-1 command cal which has at the same time an effect on three axes has become the device specific command ncal in Venus-2, move has become nmove etc. Some commands need not to be modified, they were already device dependent.

Command syntax

The commands are assembled following this scheme:
[parameter] _ {device index} _ ***command*** _

_ blank, (space) or (SP)

Parameter

The parameter transmits a value without any unit.
For positioning commands i.e. this value is the target coordinate or the relative movement.
If several parameters are needed for one command, they have to be speparated by a blank character.

Device Index

The addressing of the device module is done by the device index. This index is always an integer and selects the device.

Command

The command names the real function. It consists of several ASCII characters, lower and upper case characters are distinguished.

The following letters are allowed for commands:

ASCII-Characters	**a-z A-Z**
Umlauts	not allowed
Numbers	not allowed

Command ending character while transmitting

In the host mode data lines which are transmitted have to be completed with a CR LF character combination.
[parameter] SP {device index} SP ***command*** SP

In the terminal mode is not supported by the hydra controller.

Command ending character while receiving

[1st parameter] SP [2nd parameter] SP [n-parameter] CR LF

Data which is delivered by the controller is always completed with ASCII (CR) and (LF).

Table of important ASCII signs for programming

ASCII Code	Sign	Dez	HEX
CR	Ctrl-M	13	0xD
LF	Ctrl-J	10	0xA
SP		32	0x20
ETX	Ctrl-C	3	0x3

Command execution

For the correct programming it is important to know the internal courses during the execution of the interpreter commands.

The ASCII data transmitted by a host run through the following areas of the controller:

data input interfaces
scanner / stack
interpreter

Data converter

The data from several hardware interfaces are transfered line by line to the scanner input.

Scanner -> Interpreter -> Stack

The line data is read by the scanner and during this checked for parameters, commands and correct device index. The parameters are transmitted to a stack which can accept up to 99 values.
If the scanner separates the line into tokens. Parameter are pushed on the internal parameter stack and the interpreter looks for the command in the different command tables.

Valid commands are immediately executed and the needed parameter are taken from the parameter stack.

Blocking and non blocking commands

Hydra has no more blocking commands. All commands are executed immediately are not waiting that the previously executed command has finished.
Only a new command for asynchronously get the status is added. See command ***ast*** (p. 184)

Producing an automatic status reply message

With the following sequence of instructions an synchronously event can be generated (applies here to the 1. device) :

	Command
1:	10.2 1 ***nmove*** (p. 111)
2:	0 1 ***ast*** (p. 184)

Effect:
An automatic status feedback is produced, after the instruction 10.2 1 ***nmove*** has finished.

Data reply message to the host

The controller only delivers data if it was requested by the host.

Broadcast commands

The typical Venus-2 command needs the device index for the correct device assignment.

Storing settings

Most settings are storable. So they are not lost after power off. Use command ***nsave*** (p. 62) ***csave*** (p. 61)
to store the configuration parameters of the specified devices.

Example of a typical program to control the hydra

Hydra device configuration

	Command	Description
1:	10 1 ***snv*** (p. 50)	Velocity setting, device-1
2:	5 2 ***snv***	Velocity setting, device-2
3:		
4:	2 1 ***setpitch*** (p. 85)	Pitch setting, device-1
5:	2 2 ***setpitch***	Pitch setting, device-2
6:	100 1 ***sna*** (p. 45)	Acceleration setting, device-1
7:		
8:	1 ***nsave*** (p. 62)	Save all parameters, device-1

	Command	Description
9:	2 ***nsave***	Save all parameters, device-2
10:		
11:	1 ***ncal*** (p. 87)	Move to endswitches, device-1
12:	1 ***nrm*** (p. 105)	(find limits)
13:		
14:	2 ***ncal***	Move to endswitches, device-2
15:		
16:		
17:	15 1 ***nm*** (p. 111)	Positioning absolute, device-1
18:		
19:	1 ***nst*** (p. 183)	Ask for status, device-1
20:	1 ***gne*** (p. 66)	Ask for command decoding error (venus error)
21:	1 ***np*** (p. 190)	Ask for the actual position of device 1
22:		
23:	2.003 2 ***nm***	Positioning absolute, device-2
24:		
25:	2 ***nst***	Ask for status, device-2
26:	2 ***np***	Ask for actual position of device 2

Devices

Controller [Device 0]

This device is the controller itself. Communications state are found here, because these states have effect to all devices. This device has no device number and therefore all commands for the controller are not found at any other device.

Controller

ControllerIO

InputOutput

Interpreter

Motion

Status and position

Axis [Device 1..2]

Controller

Dynamics

Interpreter

Manual operation

Mechanic

Mechanic setup

Motion

Motor

Safety functions

Sensoric

Servo

Status and position

Switches and reference mark

Trigger

Sensor [Device 3]

This device has only measure features. No motor can be connected to this device.

Controller

Interpreter

Sensoric

Datatypes

double

Description

A floating point value with double precision. Values given as integers are converted automatically.

Size: 64 Bit

Range: (+|-) 10^{-308} to 10^{308}

Syntax

Reg.Ex:

```
-?[01234567890]+(.[1234567890]+)?
```

Examples :

100.5
231.321
5352

int

Description

An integer value.

Size: 32 Bit
Range: (+|-) 2147483647

Syntax

Reg.Ex:

```
-?[01234567890]+
```

Examples :

1001
512
-3165

long long

Description

An integer value.

Size: 64 Bit
Range: (+|-) 9223372036854775807

Syntax

Reg.Ex:

```
-?[01234567890]+
```

Examples :

9223372036854775807
512
-3165

machineerrorcodes

Description

The following machine error code are generated.
0 "no machine errors"
11 "emergency stop"
12 "motor overcurrent"
13 "following error"
23 "I²t overflows"
100 "EEPROM checksum error"
101 "no sensor available"

Size: 32 Bit

Syntax

Examples :

sensorstate

Description

state of the sensor
UNDEFINED=-1
isOK=0
SENSORERROR=1
LOWAMPLITUDE=2
LOWQUALTITY=4
NOTMAPPED=8
NOTCONNECTED=16
NOMT=32
POSOUTOFRANGE=64
ILLSUBDEVICE=128
any combination of the above values are possible

Size: 32 Bit
Range: (+|-) 2147483647

Syntax

Reg.Ex:
-?[01234567890]+

Examples :
1001
512
-3165

sensortypes

Description

The following sensor types are supported.
Possible values are:
nanoStarType = 0
microStarPCSType = 1
betaStarType = 2
betaStarMultiTurnType = 3
sincosStarType = 4
MFStarType = 5
miniStarType = 6
needleStarType = 7

Size: 32 Bit
Range: 0..7

Syntax

Examples :
1 for nanoStarInterface
4 for sincosStarInterface

string

Description

A sequence of characters enclosed by double quotes

Size: variable

Syntax

Reg.Ex:
".+"

Examples :
"This is a test string"
"Hello world"
"ABCD"

Controller

Controller identification

🔒

read-only

Commands

Properties

Type: string

identify

Returns the actual ***Controller identification***.

Syntax

identify

Reply

[identifystring]

Controller name

read-only

Commands

Properties

Type: string

nidentify

Returns the actual ***Controller name***.

Syntax

{device} ***nidentify***

Reply

[identifystring]

Controller version

🔒

read-only

CPU firmware revision.

Commands

version (getversion) 26

Properties

Type: double

version (getversion)

Returns the actual ***Controller version***.

Syntax

version

Reply

[version]

CPU temperature

🔒

read-only

CPU temperature.

Commands

Properties

Type: double
Unit: °C

---

getcputemp

Returns the actual ***CPU temperature***.

Syntax

getcputemp

Reply

[temperature]

Device class

Type of a device specified by device number.

Commands

Properties

<table>
<tr><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>[devicenr]</td><td>int</td><td></td></tr>
<tr><td>[class code]</td><td>int</td><td><table><tr><th>value</th><th>description</th></tr><tr><td>0</td><td>controller device</td></tr><tr><td>1</td><td>axis device</td></tr><tr><td>2</td><td>sensor device</td></tr></table></td></tr>
</table>

getdeviceclass

Returns the ***Device class***.

Syntax

[devicenr] ***getdeviceclass***

Reply

[class code]

Device count

🔒

read-only

Number of devices minus the controller device.

Commands

Properties

Type: int

getaxc

Returns the actual ***Device count***.

Syntax

getaxc

Reply

[axiscount]

Machine error

read-only

Commands

Properties

Name	Type
[devicenr]	int
[machine error]	int

gme (getmerror)

This command pops the last machine error code from the machine error stack. If no errors on the stack then 0 is returned. The command ***merrordecode*** (p. 64) returns the error description of the code in a string.

Syntax

[devicenr] ***gme***

Reply

[machine error]

Examples

	Command	Description
1:	1 ***gme***	Returns the machine error code, 0 if no error messages pending

Parameter stack (Controller)

🔒 read-only

Stack for Venus parameters. Temporarily contains parameter values entered until further processing. Should be empty if no commands pending. The *stackpointer* indicates the number of parameter values currently pending.

Commands

Properties

Type: int

---

gsp

Returns number of parameter values currently pending on ***Parameter stack***.

Syntax

gsp

Reply

[stackpointer]

Examples

Example

Preconditions: Parameter stack is empty.

	Command	Description
1:	***gsp*** (p. 32)	Returns 0.
2:	0 2	Push 2 parameter values on stack without entering a command string.
3:	***gsp***	Returns 2.
4:	***clear*** (p. 33)	Clears stack.
5:	***gsp***	Returns 0 again.

clear

Clears ***Parameter stack***, discarding its content.

Normally, no parameters are left on the stack, unless too many parameter values are entered with a certain command.

Syntax

clear

Reset

Controller hardware reset.

Commands

reset

Initiates ***Reset***.

⚠ Volatile data will be lost. Execute ***save*** or ***csave*** (p. 61) before execution if needed.

Syntax

reset

Serial number

Serial number of controller hardware.

🔒 read-only

Commands

Properties

Type: string

getserialno

Returns ***Serial number***.

Syntax

getserialno

Reply

[number]

Supply voltage

Supply voltage of the motor power stage.

read-only

Commands

Properties

Name	Type
[value]	double
[type]	string

getusupply

Returns the ***Supply voltage***.

Syntax

getusupply

Reply

[value] [type]

Time of day

🔒

read-only

Commands

Properties

Type: string

---

gettime

Returns the actual ***Time of day***.

Syntax

gettime

Reply

[time]

Version

read-only

Same as ***Controller version*** (p. 26) . Needs a specified device index, but returns the same result. Compatibility purpose only.

Commands

Properties

Type: double

nversion (getnversion)

Returns the actual ***Version***.

Syntax

{device} ***nversion***

Reply

[versionnumber]

ControllerIO

Network

storable

TCP/IP network.

Commands

Properties

i	Name	Type
0	[TCP/IP adress]	string
1	[Subnet mask]	string
2	[gateway]	string
3	[nameserver]	string
4	[Timeserver]	string

setnetpara

Configures the ***Network***.

Syntax

[*Value*] [*i*] ***setnetpara***

Examples

Example

Note: Unlike in the table below, all commands have to be entered *in one line*.

	Command	Description
1:	"192.168.129.200" 0 ***setnetpara***	Set IP address.
2:	***save*** (p. 61)	Save all controller parameters.

Result: Controller can be accessed via IP address 192.168.129.200 after reset.

getnetpara

Returns the actual configuration of the ***Network***.

Syntax

[*i*] ***getnetpara***

Reply

[*Value*]

Serial communication

storable

RS232 communication interface.

Setting the baud rate to 0 will switch off the respective RS232 port.

With no USB option built in, port 2 must always be switched off.

Commands

Properties

i	Name	Type	Unit	Description
1	[baudrate port 1]	int	Bit/s	standard RS232 port baud rate
2	[baudrate port 2]	int	Bit/s	optional RS232/USB port baud rate

getbaudrate

Returns the current setting of the ***Serial communication*** baud rate.

Syntax

[*i*] ***getbaudrate***

Reply

[*Value*]

setbaudrate

Defines ***Serial communication*** baud rate settings.

Syntax

[*Value*] [*i*] ***setbaudrate***

Examples

Example

Note: Unlike in the table below, all commands have to be entered *in one line*.

	Command	Description
1:	115200 1 ***setbaudrate***	Set baud rate at standard RS232 port to 115.2 kHz.
2:	0 2 ***setbaudrate***	Switch off optional RS232/USB port.

Dynamics

Acceleration

storable

This states defines how the next move ramps up. The acceleration is normally defined in mm/s^2. Minimum setting is 1 µm/s^2; maximum setting is 500 m/s^2 (50G).

Commands

Properties

Type: double
Unit: mm/s^2

gna (getnaccel)

Returns the current setting of the ***Acceleration***.

Syntax

{device} ***gna***

Reply

[acceleration]

sna (setnaccel)

Sets the ***Acceleration***.

Syntax

[acceleration] {device} ***sna***

Stop deceleration

storable

Deceleration for immediate halt used upon

- touch of either end switch during any move
- ***Move abortion*** (p. 113)
- Ctrl+C shortcut

Commands

Properties

Type: double
Unit: mm/s^2

ssd (setstopdecel)

Sets the ***Stop deceleration***.

Syntax

[stop_deceleration] {device} ***ssd***

gsd (getstopdecel)

Returns the current setting of the ***Stop deceleration***.

Syntax

{device} ***gsd***

Reply

[stop_deceleration]

Velocity

storable

Velocity used by all programmed moves but

- ***Calibration move*** (p. 87)
- ***Range measure move*** (p. 105)
- ***Reference move*** (p. 117)
- ***Vector reference move***

Must range between +10 nm/s and +10 m/s.

Commands

Properties

Type: double

---

gnv (getnvel)

Returns the current setting of the ***Velocity***.

Syntax

{device} ***gnv***

Reply

[velocity]

---

snv (setnvel)

Sets the ***Velocity***.

Syntax

[velocity] {device} ***snv***

InputOutput

Digital input event command

storable

String containing a chain of commands executed upon occurrence of an event (positive level transition) at a specified digital input. String length is up to 255 characters.

To be used with inputs at external devices (such as joystick buttons) only. Input indexes 0 through 15 are reserved for onboard controller inputs which do not support this feature.

Commands

Properties

i	Name	Type
16	[joystick 1 button 1]	string
17	[joystick 1 button 2]	string
18	[joystick 1 button 3]	string
19	[joystick 1 button 4]	string
20	[joystick 1 button 5]	string
21	[joystick 1 button 6]	string

getdincmd

Returns currently set ***Digital input event command*** string.

Syntax

[*i*] ***getdincmd***

Reply

[*Value*]

---

setdincmd

Sets the ***Digital input event command***. Use the SPACE character in order to seperate symbols from each other.

⚠ Query commands (commands that imply a Venus interpreter reply such as ***np*** or ***nst***) will be ignored.

Syntax

[*Value*] [*i*] ***setdincmd***

Examples

Example

Task: Configure buttons at a connected joystick.

Note: Unlike in the table below, all commands have to be entered *in one line*.

	Command	Description
1:	"1 1 setcloop 1 16 setdout" 16 ***setdincmd***	Configure button 1 to close position control loop at axis 1 and turn on LED 1.
2:	"0 1 setcloop 0 16 setdout" 17 ***setdincmd***	Configure button 2 to open position control loop at axis 1 and turn off LED 1.
3:	"1 2 setcloop 1 17 setdout" 18 ***setdincmd***	Configure button 3 to close position control loop at axis 2 and turn on LED 2.
4:	"0 2 setcloop 0 17 setdout" 19 ***setdincmd***	Configure button 4 to open position control loop at axis 2 and turn off LED 2.
5:	"1 ncal 2 ncal" 20 ***setdincmd***	Configure button 5 to initiate calibration move at both axes.
6:	"reset" 21 ***setdincmd***	Configure button 6 to reset controller.

Result: On button pressure, the controller will react as specified.

Digital output level

not storable

Level at specified digital output which can either be a controller output or a digital output at a connected device such as a joystick indicator LED. Outputs supported at the time:

output index	destination
0	controller TTL I/O 1 *
1	controller TTL I/O 2 *
2	controller open drain output
3...15	reserved
16...23	joystick LEDs 1...8

* hardware I/O, invariably configured as output

Commands

Properties

Name	Type	Description
[Index]	int	see table above
[Value]	int	0 = off, 1 = on

getdout

Returns currently set ***Digital output level***.

Syntax

[Index] ***getdout***

Reply

[Value]

setdoutfreq

Sets digital output toggle rate. If set to 0, the output will be permanently set as long as the level is set to 1. A value different from 0 will cause the output to turn on and off alternatingly instead. Unit is 250 µs ticks.

This feature is available for direct controller outputs (indexes 0..2) only.

Syntax

[Value] [Index] ***setdoutfreq***

setdout

Sets ***Digital output level*** as specified.

Syntax

[Value] [Index] ***setdout***

Examples

Example

	Command	Description
1:	1 18 ***setdout***	Turn on joystick LED 3.
2:	0 16 ***setdout***	Turn off joystick LED 1.

Emergency switch

storable

On demand, the emergency shut-off switch can be disabled (masked). If there is no need for the shut-off switch function, this allows for running the controller without having to connect an external bridge dummy.

⚠

When disabled, a connected switch will have no effect. However, the setting does not affect any emergency-off feature based on error detection (overcurrent, following error etc.).

Commands

Properties

Type: int

value	enable state
0	disabled
1	enabled

getemsw

Returns the current setting of the ***Emergency switch***.

Syntax

getemsw

Reply

[mask]

setemsw

Sets the ***Emergency switch*** configuration.

Syntax

[mask] ***setemsw***

Interpreter

Configuration storage (Controller)

Storage of the specified device's non-volatile parameters.

Commands

save

Syntax

save

csave

With this command the configuration of the controller and all axes are saved in the flash filesystem.

Syntax

csave

Configuration storage (Axis)

Storage of the specified device's non-volatile parameters.

Commands

---

nsave

Executes ***Configuration storage***.

Syntax

```
{device}  nsave
```

Error decoder

Decoder which generates an error description string from a given error code.

Commands

Properties

Name	Type
[errorcode]	int
[errorstring]	string

errordecode

Returns string generated by ***Error decoder*** from an interpreter error code.

Syntax

[errorcode] ***errordecode***

Reply

[errorstring]

merrordecode

Returns string generated by ***Error decoder*** from a machine error code.

Syntax

[errorcode] ***merrordecode***

Reply

[errorstring]

Interpreter error (Controller)

🔒

read-only

Commands

Properties

Type: int

ge

Returns the actual ***Interpreter error***.

Syntax

ge

Reply

[errorregister]

Interpreter error (Axis)

🔒

read-only

Commands

Properties

Type: int

gne

Returns the actual ***Interpreter error***.

Syntax

{device} ***gne***

Reply

[errorregister]

Parameter stack (Axis)

read-only

Stack for Venus parameters. Temporarily contains parameter values entered until further processing. Should be empty if no commands pending. The *stackpointer* indicates the number of parameter values currently pending.

Commands

Properties

Type: int

---

nclear

Clears ***Parameter stack***, discarding its content. Although this command is axis specific, there is no axis specific parameter stack; therefore execution yields same result as ***clear***. Compatibility purpose only.

Normally, no parameters are left on the stack, unless too many parameter values are entered with a certain command.

Syntax

{device} ***nclear***

---

ngsp

Returns number of parameter values currently pending on ***Parameter stack***. Although this command is axis specific, there is no axis specific parameter stack; therefore execution yields same result as ***gsp***. Compatibility purpose only.

Syntax

{device} ***ngsp***

Reply

[stackpointer]

User doubles

storable

User memory, double type portion. Each single entry stores one floating point value.

Commands

Properties

i	Name	Type
0	[vardbl_0]	double
1	[vardbl_1]	double
2	[vardbl_2]	double
3	[vardbl_3]	double
4	[vardbl_4]	double
5	[vardbl_5]	double
6	[vardbl_6]	double
7	[vardbl_7]	double
8	[vardbl_8]	double
9	[vardbl_9]	double

setvardbl

Sets the the specified ***User doubles*** entry.

Syntax

[*Value*] [*i*] ***setvardbl***

getvardbl

Returns the current setting of the the specified ***User doubles*** entry.

Syntax

[*i*] ***getvardbl***

Reply

[*Value*]

User integers

storable

User memory, integer type portion. Each single entry stores one integer value.

Commands

Properties

i	Name	Type
0	[vardbl_0]	int
1	[vardbl_1]	int
2	[vardbl_2]	int
3	[vardbl_3]	int
4	[vardbl_4]	int
5	[vardbl_5]	int
6	[vardbl_6]	int
7	[vardbl_7]	int
8	[vardbl_8]	int
9	[vardbl_9]	int

setvarint

Sets the the specified ***User integers*** entry.

Syntax

[*Value*] [*i*] ***setvarint***

---

getvarint

Returns the current setting of the the specified ***User integers*** entry.

Syntax

[*i*] ***getvarint***

Reply

[*Value*]

User strings

storable

User memory, string type portion. Each single entry stores one string.

Commands

Properties

i	Name	Type
0	[varstring_0]	string
1	[varstring_1]	string
2	[varstring_2]	string
3	[varstring_3]	string
4	[varstring_4]	string
5	[varstring_5]	string
6	[varstring_6]	string
7	[varstring_7]	string
8	[varstring_8]	string
9	[varstring_9]	string

setvarstring

Sets the the specified ***User strings*** entry.

Syntax

[*Value*] [*i*] ***setvarstring***

---

getvarstring

Returns the current setting of the the specified ***User strings*** entry.

Syntax

[*i*] ***getvarstring***

Reply

[*Value*]

Manual operation

Manual device entry

not storable

Entry which allows for access to a specified parameter of a specified manual device. The target manual device is selected by the *control index*, whereas the *parameter index* allows for choice of one parameter out of the ***Manual device parameters*** (p. 79) set.

Commands

Properties

Name	Type	Description
[control index]	int	selection index of target manual device
[parameter index]	int	selection index of target parameter (see ***Manual device parameters***, [i] column)
[parameter value]	double	value selected parameter is to be set to

setmanpara

Changes the current setting of one of the ***Manual device parameters*** (p. 79) .

Syntax

[parameter value] [parameter index] [control index] {device} ***setmanpara***

Examples

Example

	Command	Description
1:	100 1 1 1 ***setmanpara***	Sets the *velocity* at the 1st manual device of axis 1 to 100 mm/s.
2:	2 6 1 2 ***setmanpara***	Sets the *input function* at the 1st manual device of axis 2 to cubic progression.

getmanpara

Returns the current setting of one of the ***Manual device parameters***.

Syntax

[parameter value] [parameter index] [control index] {device} ***getmanpara***

The *parameter value* entry is an ineffective dummy here; any value (e.g. 0) will be taken. If it is ommitted, though, proper execution is still granted, but the Venus interpreter will report an ***Interpreter error*** (p. 65) (code 1002). This is due to the specific parameter stack handling.

Reply

[parameter value]

Examples

Example

	Command	Description
1:	0 2 1 1 ***getmanpara***	Returns the *acceleration* setting at the 1st manual device of axis 1.
2:	0 7 2 1 ***getmanpara***	Returns the *mode* setting at the 2nd manual device of axis 1.

Manual device parameters

storable

Properties of specific manual device.

A programmed move unconditionally overrules manual motion.

As for the *mode*, there are 3 different settings:

- 0: disabled
- 1: enabled until next programmed move; disabled afterwards
- 2: enabled until next programmed move and afterwards

Properties

i	Name	Type	Unit	Description
1	[velocity]	double	mm/s	standard manual move velocity
2	[acceleration]	double	mm/s²	manual move acceleration
3	[alt. velocity]	double	mm/s	manual move velocity valid when button pressed - *void since firmware rev. 2.200*
4	[resolution]	int	1/rev	handwheel - encoder steps per revolution
5	[ratio]	double	mm/rev	handwheel - distance covered per revolution
6	[input function]	int	-	joystick - elongation to velocity transfer function (0: linear, 1: square, 3: cubic, etc.)
7	[mode]	int	-	0: disabled, 1: conditionally enabled, 2: enabled
8	[threshold]	double	-	elongation threshold below which generated velocity will be 0; normalized to elongation range (1.0 => full elongation)
9	[direction]	int	-	effective direction (0, 1)

i	Name	Type	Unit	Description
10	[input driver]	int	-	0: joystick 1
11	[input channel]	int	-	joystick - 0: horizontal, 1: vertical

Manual motion control

storable

Integrated activation state of all manual devices assigned to axis device. Allows for bitwise enabling / disabling.

Commands

Properties

Type: int
0: disabled - 1: enabled

setmanctrl

Sets the ***Manual motion control***.

Syntax

[enable state] {device} ***setmanctrl***

getmanctrl

Returns the current setting of the ***Manual motion control***.

Syntax

{device} ***getmanctrl***

Reply

[enable state]

Mechanic

Pitch

storable

Move distance to be covered by one complete motor axis turn.

⚠

To operate the device in closed loop mode, the parameter setting has to provide that delivered move distances equal actually covered move distances. See also ***Motor pole pairs*** (p. 138)

Commands

getpitch.. 84
setpitch.. 85

Properties

Type: double
Unit: mm

---

getpitch

Returns the current setting of the ***Pitch***.

Syntax

{device} ***getpitch***

Reply

[pitch]

---

setpitch

Sets the ***Pitch***.

⚠ Upon change of ***Pitch***, the motor axis may leap to another position. This can be avoided by moving the device to its initial position origin before changing.

Syntax

[pitch] {device} ***setpitch***

Examples

	Command	Description
1:	5 1 ***setpitch***	Sets pitch at device 1 to 5 mm per turn.
2:	10 1 nr	Moves device 1 by 10 mm.

Causes device 1 motor axis to turn twice.

Mechanic setup

Calibration move

Composite move for the purpose of finding the lower motion range limit and defining the position origin. Partial operations in the order of execution:

- motion towards "Cal" end switch until touching
- backward motion until switch is released
- continuation of backward motion until ***Calibration switch distance*** (p. 88) is covered
- definition of actual position as new origin of the reference coordinate system, i.e nominal position is 0 afterwards (see ***Device position*** (p. 190))
- setting of ***Hardware limits*** (p. 92) to 0 (lower limit) and default (upper limit)

Commands

ncal (ncalibrate)

Initiates a ***Calibration move***.

Syntax

{device} ***ncal***

Calibration switch distance

storable

Distance from "Cal" end switch at which ***Calibration move*** (p. 87) terminates.

Commands

Properties

Type: double
Unit: mm

getncalswdist

Returns the current setting of the ***Calibration switch distance***.

Syntax

{device} ***getncalswdist***

Reply

[dist from switch]

setncalswdist

Sets the ***Calibration switch distance***.

Syntax

[dist from switch] {device} ***setncalswdist***

Calibration velocity

storable

Velocity at which a ***Calibration move*** (p. 87) will run. Set separately for motion towards switch and backward motion, respectively.

Commands

Properties

i	Name	Type	Unit
1	[vel into switch]	double	mm/s
2	[vel out of switch]	double	mm/s

getncalvel

Returns the current setting of the ***Calibration velocity***.

Syntax

[*i*] {device} ***getncalvel***

Reply

[*Value*]

setncalvel

Sets the ***Calibration velocity***.

Syntax

[*Value*] [*i*] {device} ***setncalvel***

Hardware limits

not storable

Limits of available motion range, applying to all moves except ***Calibration move*** (p. 87) and ***Range measure move*** (p. 105). The effect is that if the move's end position exceeds the specified range, the respective range limit will be targetted instead.

Limits will be affected by each move touching either end switch.

For initialization of the limits with defined values after powerup, see ***Initial limits*** (p. 94) .

Commands

Properties

Name	**Type**	**Unit**
[upperlimit]	double	mm
[lowerlimit]	double	mm

getnlimit

Returns the current setting of the ***Hardware limits***.

Syntax

{device} ***getnlimit***

Reply

[lowerlimit] [upperlimit]

setnlimit

Sets the ***Hardware limits***.

Syntax

[lowerlimit] [upperlimit] {device} ***setnlimit***

Initial limits

storable

Defines storable values for the initialization of the ***Hardware limits*** (p. 92) . The limits are valid after powerup until changed manually or by any hit of an end switch during execution of a move command except for ***Calibration move*** (p. 87) .

⚠

Regard that the initial limits will not be matched upon change of the ***Position origin*** (p. 100) .

Commands

Properties

Name	Type	Unit
[upperlimit]	double	mm
[lowerlimit]	double	mm

---

getinilimit

Returns the current setting of the ***Initial limits***.

Syntax

{device} ***getinilimit***

Reply

[lowerlimit] [upperlimit]

setinilimit

Sets the ***Initial limits***.

Syntax

[lowerlimit] [upperlimit] {device} ***setinilimit***

Motion direction

storable

Defines the device's spatial orientation. When inverted, the motor's rotation/motion direction and the end switch assignment are being inverted simultaneously.

With the position decreasing, the device is always moving towards "Cal" switch. With the position increasing, the device is always moving towards "RM" switch. This does NOT depend on the parameter setting.

Regard that the setting has to be consistent with the count direction of the respective position sensor. Put the device into open loop mode before changing the parameter setting in order to avoid unwanted motion in either direction.

Commands

Properties

Type: int

setmotiondir

Sets the ***Motion direction***.

Syntax

[direction] {device} ***setmotiondir***

getmotiondir

Returns the current setting of the ***Motion direction***.

Syntax

{device} ***getmotiondir***

Reply

[direction]

Motion function

storable

Defines the respective activation states of ***Calibration move*** (p. 87) , ***Range measure move*** (p. 105) and ***Reference move*** (p. 117) by a bit-coded value. If the respective bit is low, each corresponding move request will be discarded.

Calibration move and ***Range measure move*** should be disabled if no corresponding end switch is connected. ***Reference move*** should be disabled if the measurement system connected does not provide a reference mark.

Bit number	description
0	***Calibration move*** activation
1	***Range measure move*** activation
2	***Reference move*** activation

Commands

Properties

Type: int

getmotionfunc

Returns the current setting of the ***Motion function***.

Syntax

{device} ***getmotionfunc***

Reply

[mask]

setmotionfunc

Sets the ***Motion function***.

Syntax

[mask] {device} ***setmotionfunc***

Examples

Example 1

Task: Disable ***Range measure move*** (p. 105), but leave ***Calibration move*** (p. 87) and ***Reference move*** (p. 117) enabled.

	Command	Description
1:	5 1 ***setmotionfunc***	Sets bit 2 and 0 and clears bit 1 in the register.

Result: Any ***Calibration move*** request will be ignored afterwards.

Position origin

storable

Defines location of position origin relative to its initial location.

Commands

Properties

Type: double
Unit: mm

getnpos

Returns current ***Position origin***.

Syntax

{device} ***getnpos***

Reply

[position]

Examples

Example 1

A sequence of commands (targeting device 1), showing the ***setnpos*** (p. 101)/***getnpos*** mode of operation, assuming no ***setnpos*** command was executed since last powerup, and that the effective direction of the offset (see ***Position origin configuration*** (p. 103)) is set to default (0).

	Command	Description
1:	1 ***getnpos***	Returns origin offset which is 0 by default.
2:	20 1 ***nm***	Induces a 20 mm move into positive direction. So upon arrival, origin is located 20 mm from current nominal position in negative direction. Nominal position is now 20 mm.
3:	10 1 ***setnpos***	Shifts origin to position which is located 10 mm from current nominal position (20 mm) in positive direction. So from its former position, origin is being shifted by (20 mm + 10 mm) = 30 mm into positive direction. Nominal position is now -10 mm.
4:	1 ***getnpos***	Returns current origin offset which is now 30 mm.
5:	-20 1 ***setnpos***	Shifts back origin to position which is located 20 mm from current nominal position in negative direction, i.e. to its initial position. Nominal position is 20 mm again.
6:	1 ***getnpos***	Returns current origin offset which is 0 again.

setnpos

Redefines ***Position origin***. Value has to be entered relative to current nominal position.

Syntax

[position] {device} ***setnpos***

Examples

Example 1

	Command	Description
1:	50 1***setnpos***	Shifts origin of device 1 to position located 50 mm from current nominal position in positive direction. Nominal position is -50 mm afterwards.

Example 2

	Command	Description
1:	-20 2***setnpos***	Shifts origin of device 2 to position located 20 mm from current nominal position in negative direction. Nominal position is 20 mm afterwards.

Note: Examples 1 and 2 assuming that the effective direction of the offset (see ***Position origin configuration***) is set to default (0).

Example 3

	Command	Description
1:	0 1 ***setnpos***	Shifts origin of device 1 to current nominal position. Nominal position is 0 afterwards.

Position origin configuration

storable

Configures the handling of the ***Position origin*** (p. 100) . The only parameter to be configured at the time is the effective direction, i.e. the sign of the origin.

Commands

Properties

i	Name	Type	Description
1	[sign]	int	0: ***Position origin*** defines offset of new user origin, measured from initial origin 1: ***Position origin*** defines offset of initial origin, measured from new user origin

getorgconfig

Returns current ***Position origin configuration***.

Syntax

[*i*] {device} ***getorgconfig***

Reply

[*Value*]

setorgconfig

Sets the ***Position origin configuration***.

Syntax

[*Value*] [*i*] {device} ***setorgconfig***

Examples

Example

Precondition: Each axis is positioned at its initial origin.

Note: Unlike in the table below, all commands have to be entered *in one line*.

	Command	Description
1:	0 1 1 ***setorgconfig***	Configure origin direction at axis 1.
2:	1 1 2 ***setorgconfig***	Configure origin direction at axis 2.
3:	10 1 ***setnpos***	Define new position origin at axis 1.
4:	10 2 ***setnpos***	Define new position origin at axis 2.
5:	1 ***np***	Returns -10.000000.
6:	2 ***np***	Returns 10.000000.
7:	1 ***getnpos*** (p. 100)	Returns 10.000000.
8:	2 ***getnpos***	Returns -10.000000.

Range measure move

Composite move for the purpose of finding the upper motion range limit. Partial operations in the order of execution:

- motion towards "RM" end switch until touching
- backward motion until switch is released
- setting of upper hardware limit (see ***Hardware limits*** (p. 92)) to actual position

Commands

nrm (nrangemeasure)

Initiates a ***Range measure move***.

Syntax

{device} ***nrm***

Range measure velocity

storable

Velocity at which a ***Range measure move*** (p. 105) will run. Set separately for motion towards switch and motion out of switch, respectively.

Commands

Properties

i	Name	Type	Unit
1	[vel into switch]	double	mm/s
2	[vel out of switch]	double	mm/s

setnrmvel

Sets the ***Range measure velocity***.

Syntax

[*Value*] [*i*] {device} ***setnrmvel***

getnrmvel

Returns the current setting of the ***Range measure velocity***.

Syntax

[*i*] {device} ***getnrmvel***

Reply

[*Value*]

Reference velocity

storable

Defines motion velocity for reference position mark detection. The setting takes effect upon execution of ***refmove*** (p. 112) and ***nrefmove*** (p. 117).

Commands

Properties

i	Name	Type
1	[forward velocity]	double
2	[backward velocity]	double

getnrefvel

Returns the current setting of the ***Reference velocity***.

Syntax

[*i*] {device} ***getnrefvel***

Reply

[*Value*]

setnrefvel

Sets the ***Reference velocity***.

Syntax

[*Value*] [*i*] {device} ***setnrefvel***

Motion

Absolute move

Moving the axis is controlled by this state. The parameter is the target position starting at the actual position.
If previous move action has not finished and a new move command is received, then the current motion will be interrupted and the new motion is initiated to the new target.
If the new move is in the same direction as the previous one, then no motion stop will be initiated. Only the acceleration and the velocity will be adapted to the new move command.
Actual a move must be in range of -+200 m.
The resolution is 1 nm if the default unit of mm is used.

Commands

Properties

Type: double

nm (nmove)

Syntax

[targetposition] {device} ***nm***

Controller reference move

Extended version of ***Reference move*** (p. 117) ; same function, but run at all available axes simultaneously. All axes will cover the same given distance, but each one will conform its individual ***Reference velocity*** (p. 108) and ***Reference configuration*** (p. 194) .

Commands

Properties

Type: double
Unit: mm

refmove

Initiates a ***Controller reference move***.

Syntax

[distance] ***refmove***

Move abortion

Immediate stop of currently running move. ***Stop deceleration*** (p. 47) is used as stopping slope.

Commands

nabort

Executes ***Move abortion***.

Syntax

{device} ***nabort***

Parameterised absolute move

not storable

Absolute move which takes individual dynamic parameter settings (velocity, acceleration) instead of using the global ones.

The *acceleration* parameter also defines the deceleration at move termination. The *deceleration* parameter is a noneffective dummy at the time. It has to be entered with the command line nonetheless.

Commands

Properties

Name	Type	Unit	Description
[targetposition]	double	mm	target position of move
[velocity]	double	mm/s	target velocity during move
[acceleration]	double	mm/s²	acceleration / deceleration
[deceleration]	double	-	noneffective

pm

Initiates a ***Parameterised absolute move***.

Syntax

{device} ***pm***

Parameterised relative move

not storable

Relative move which takes individual dynamic parameter settings (velocity, acceleration) instead of using the global ones.

The *acceleration* parameter also defines the deceleration at move termination. The *deceleration* parameter is a noneffective dummy at the time. It has to be entered with the command line nonetheless.

Commands

Properties

Name	Type	Unit	Description
[distance]	double	mm	move distance to be covered
[velocity]	double	mm/s	target velocity during move
[acceleration]	double	mm/s²	acceleration / deceleration
[deceleration]	double	-	noneffective

pr

Initiates a ***Parameterised relative move***.

Syntax

{device} ***pr***

Random move

A sequence of moves targeting virtually random positions at virtually random velocities. The distance and velocity of each invidual move are calculated in a way it will not take more than about 8 seconds. Move velocity will be ***Velocity*** (p. 49) at most, and move acceleration/deceleration will be ***Acceleration*** (p. 45) . Upon execution of Ctrl+C or ***nabort*** (p. 113), the device will halt immediately.

⚠ For move parameter calculation, valid move range limits are needed. Therefore execution of the move will fail unless the limits have been set before - either manually or by execution of ***ncal*** (p. 87) and ***nrm*** (p. 105) (in this order).

Commands

nrandmove

Executes ***Random move***.

Syntax

{device} ***nrandmove***

Reference move

Relative move covering the given distance. While moving, reference position mark detection is performed according to ***Reference configuration*** (p. 194). If the mark is detected before reaching the end position, the device will brake and return to the mark's location. See also ***Reference velocity*** (p. 108) and ***Reference status*** (p. 196) .

Decreasing ***Reference velocity*** setting helps increase the precision of reference detection. With the device moving too fast, one of the following may happen:

- reference detection fails though the mark is covered by move range
- with reference mark found, motion stops at its edge, so afterwards the reference signal may be alternating or inactive

Commands

nrefmove.. 117

Properties

Type: double
Unit: mm

nrefmove

Initiates a ***Reference move***.

Syntax

[distance] {device} ***nrefmove***

Relative move

Move covering the given distance. Uses ***Velocity*** (p. 49) and ***Acceleration*** (p. 45) .

⚠ With firmware revisions older than 2.200, the distance always refers to the lastly calculated end position as the move starting point. That means if a running Relative move is cancelled by execution of another Relative move, the altogether covered distance will be the sum of the individual move distances. It also means that if a running manual or calibration move is cancelled by execution of a relative move, the axis will run into the respective hardware limit. With newer revisions, the distance always refers to the current nominal position in order to avoid this.

Commands

Properties

Type: double
Unit: mm

---

nr (nrmove)

Executes ***Relative move***.

ℹ With firmware revisions older than 2.200, if the distance entered is zero, a currently running move will be stopped. With newer revisions, the same move request will be ignored.

Syntax

[distance] {device} ***nr***

Vector absolute move

Commands

Properties

Name	Type
[y-value]	double
[x-value]	double
[status]	int

m

Syntax

[x-value] [y-value] ***m***

v2r

Syntax

[x-value] [y-value] ***v2r***

Reply

[status]

---

v2m

Syntax

[x-value] [y-value] ***v2m***

Reply

[status]

---

r

Syntax

[x-value] [y-value] ***r***

Vector acceleration

storable

Commands

Properties

Type: double

ga (getaccel)

Returns the current setting of the ***Vector acceleration***.

Syntax

ga

Reply

[acceleration]

sa (setaccel)

Sets the ***Vector acceleration***.

Syntax

[acceleration] ***sa***

Vector velocity

storable

Commands

Properties

Type: double

sv (setvel)

Sets the ***Vector velocity***.

Syntax

[velocity] ***sv***

gv (getvel)

Returns the current setting of the ***Vector velocity***.

Syntax

gv

Reply

[velocity]

Motor

Absolute motor current

🔒 read-only

Absolute value of the overall motor current vector which is formed out of the phase current values (***Motor phase current*** (p. 135)).

Commands

Properties

Type: double
Unit: A

---

gi (getcurrent)

Returns the actual ***Absolute motor current***.

Syntax

{device} ***gi***

Reply

[absolutcurrent]

Auto commutation

storable

Automatic commutation feature, on the purpose of detecting the initial axis or slide position before applying motor power. If enabled, it is executed once during powerup. Upon success (i.e. if the procedure leads to a definite result), the axis or slide movement caused by motor power application will be reduced to a minimum. Parameter settings vary according to motor type and load.

2500 µs has emerged to be a ***time*** parameter value that works in most cases, whereas the voltage parameter value is motor type specific.

Commands

Properties

i	**Name**	**Type**	**Unit**	**Description**
1	[voltage]	double	V	Absolute motor vector voltage during procedure.
2	[time]	int	µs	Application time of single angle step. To be set in steps of 250 µs; otherwise rounded off or up, respectively. Disables procedure if 0 (or less than 125 µs).

getamc

Returns the current setting of the ***Auto commutation***.

Syntax

[*i*] {device} ***getamc***

Reply

[*Value*]

setamc

Configures the ***Auto commutation***.

Syntax

[*Value*] [*i*] {device} ***setamc***

Motor brake

storable

Mechanical motor brake. It is possible to dedicate one of the digital controller outputs to a special function which is designed to control a mechanical motor brake, especially for the use with vertical axes to be held in position while the motor is without power. During powerup, the brake output will be kept at low level until the motor is powered, then go high to open the brake. In case of emergency power off, the output will switch to low level again and kept at this state until the motor is repowered by ***Motor restart*** (p. 145) .

output index	destination
0	controller TTL I/O 1 *
1	controller TTL I/O 2 *
2	controller open drain output

* hardware I/O, invariably configured as output

⚠ Neither does the brake control output function generally overrule the ***Digital output level*** (p. 55) function, nor vice versa. Any level settings will be executed. It is up to the user to avoid potential conflicts.

Commands

Properties

i	Name	Type	Description
0	[brake enable]	int	brake function (0: disabled 1: enabled)
1	[output]	int	brake control output index

getbrakefunc

Returns the current configuration of the ***Motor brake*** control function.

Syntax

[*i*] {device} ***getbrakefunc***

Reply

[*Value*]

setbrakefunc

Configures the ***Motor brake*** control function.

Syntax

[*Value*] [*i*] {device} ***setbrakefunc***

Examples

Example 1

	Command	Description
1:	2 1 ***setbrakefunc***	Configure the open drain output to control the motor brake.
2:	1 0 ***setbrakefunc***	Enable motor brake control function.

Motor current limit

storable

Defines the motor current limit used for safety purposes. As soon as the actual ***Absolute motor current*** (p. 127) exceeds this limit during operation, the device will be put into emergency state; i.e. the motor will be powered down and kept shut down until ***Motor restart*** (p. 145) .

Commands

Properties

Type: double
Unit: A

---

getmaxcurrent

Returns the current setting of the ***Motor current limit***.

Syntax

{device} ***getmaxcurrent***

Reply

[maximum_current]

---

setmaxcurrent

Sets the ***Motor current limit***.

Syntax

[maximum_current] {device} ***setmaxcurrent***

Motor phase current

read-only

Measured current at 2 motor phases. See also ***Absolute motor current*** (p. 127) .

Commands

Properties

Name	Type	Unit
[current phase1]	double	A
[current phase2]	double	A

gc

Returns the actual ***Motor phase current***.

Syntax

{device} ***gc***

Reply

[current phase1] [current phase2]

Motor phase number

storable

Number of motor phases. To be set to 2 or 3, depending on the used motor.

⚠

A mismatched setting will prevent proper motor operation.

Commands

Properties

Type: int

setphases

Sets the ***Motor phase number***.

Syntax

[phase] {device} ***setphases***

getphases

Returns the current setting of the ***Motor phase number***.

Syntax

{device} ***getphases***

Reply

[phase]

Motor pole pairs

storable

Number of motor polepairs. Normally set to 50 or 100 when the controller is used with a stepper motor. To be set to 1 when the controller is used as a linear motor drive.

⚠ A mismatched setting will lead to a mismatch between delivered and actually covered move distances.

Commands

Properties

Type: int

setpolepairs

Sets the ***Motor pole pairs***.

Syntax

[polepairs] {device} ***setpolepairs***

getpolepairs

Returns the current setting of the ***Motor pole pairs***.

Syntax

{device} ***getpolepairs***

Reply

[polepairs]

Motor voltage gradient

storable

Velocity gradient of motor phase voltage amplitude.

⚠

Caution - to be handled with care! In order to prevent motor and controller hardware from sustaining damage, the respective maximum current ratings have to be met. Use ***gc*** (p. 135) and ***gi*** (p. 127) for current observation.

Commands

Properties

Type: double

setumotgrad

Sets the ***Motor voltage gradient***.

Syntax

[voltage_gradient] {device} ***setumotgrad***

getumotgrad

Returns the current setting of the ***Motor voltage gradient***.

Syntax

{device} ***getumotgrad***

Reply

[voltage_gradient]

Motor voltage minimum

storable

Base rate of motor phase voltage amplitude.

⚠

Caution - to be handled with care! In order to prevent motor and controller hardware from sustaining damage, the respective maximum current ratings have to be met. Use ***gc*** (p. 135) and ***gi*** (p. 127) for current observation.

Commands

getumotmin..142

setumotmin..143

Properties

Type: double
Unit: V

---

getumotmin

Returns the current setting of the ***Motor voltage minimum***.

Syntax

{device} ***getumotmin***

Reply

[minimum_voltage]

---

setumotmin

Sets the ***Motor voltage minimum***.

Syntax

[minimum_voltage] {device} ***setumotmin***

Safety functions

Motor restart

Motor restart feature. Allows for resuming full operation from emergency state without needing a controller reset or losing the position or the reference coordinate system.

Commands

init

Upon execution, the following will happen:

- Motor powerup according to actual motor parameters (***Motor voltage minimum*** (p. 142) , ***Motor voltage gradient*** (p. 140)). Unwanted motion is reduced to a minimum.
- Reactivation of positioning controller according to actual ***Positioning control mode*** setting.

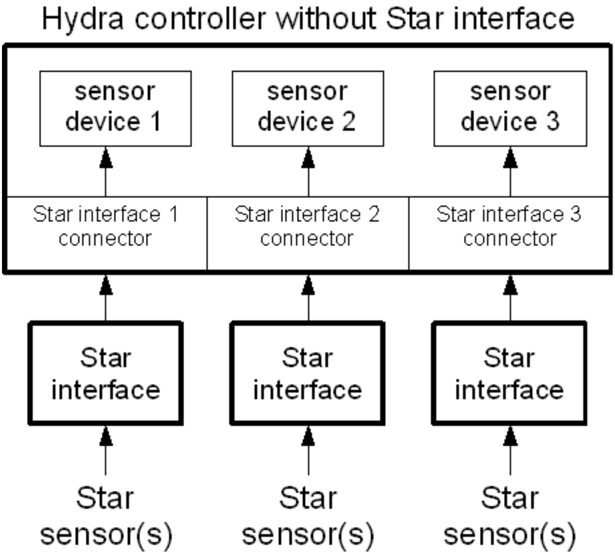
Syntax

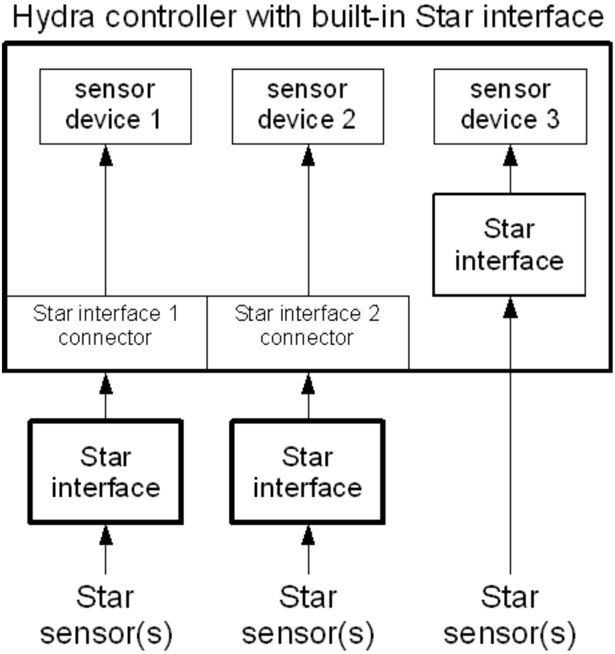
{device} ***init***

Sensoric

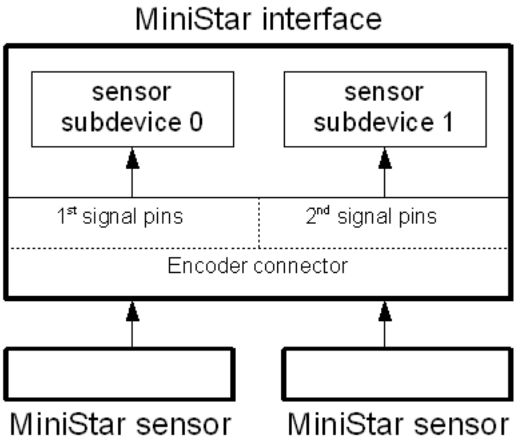
Position sensor routing

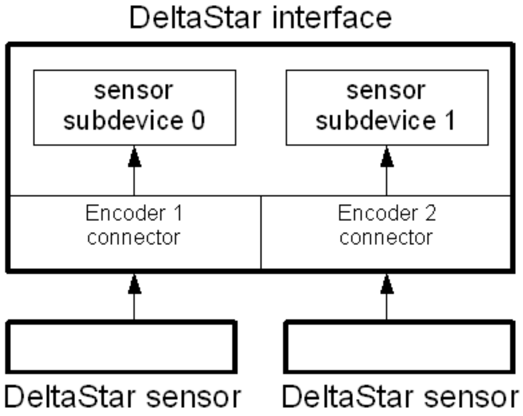
The Hydra controller features 3 Star interface ports which the Venus interpreter distinguishes by means of the Star sensor **device** index. Ports 1 and 2 provide connection of one external Star interface (i.e. any member of the Star interface family) each, using the Star interface 1 and 2 sockets. Access to Port 3, however, depends on the respective Hydra model. If your controller is equipped with a built-in Star interface, the latter is internally connected to Port 3, providing for direct external connection of matching position sensors at the encoder 1/2 sockets. Otherwise, Port 3 is accessible via the Star interface 3 socket.

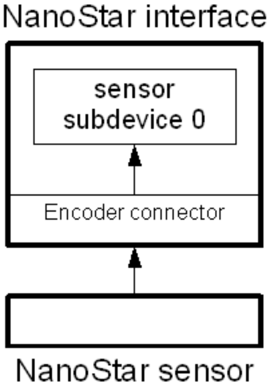




The number of position sensors connectable to a single Star interface is up to 2, varying with the model. The venus interpreter distinguishes the different sensors connected to one Star interface by means of the Star sensor **subdevice** index. The following diagrams depict the sensor connector to subdevice index association for each member of the Star family.


MiniStar interface
sensor subdevice 0
sensor subdevice 1
1st signal pins
2nd signal pins
Encoder connector
MiniStar sensor
MiniStar sensor


DeltaStar interface
sensor subdevice 0
sensor subdevice 1
Encoder 1 connector
Encoder 2 connector
DeltaStar sensor
DeltaStar sensor



Normally, a position measurement system is linked to a motor axis for use with a position controller. Therefore each connected sensor has to be routed to the destination axis device, specifying Star sensor device and subdevice and axis device indexes. See ***Sensor Assignment*** on how sensor routing is done. After routing, sensor position and (if applicable) trigger functions can be accessed using the axis device index.

Scale period

not storable

Commands

Properties

Type: double

getclperiod

Returns the current setting of the ***Scale period***.

Syntax

{device} ***getclperiod***

Reply

[Period]

setclperiod

Sets the ***Scale period***.

Syntax

[Period] {device} ***setclperiod***

Sensor amplitudes

🔒

read-only

Commands

Properties

Name	Type
[sin amplitude track 0]	int
[cos amplitude track 0]	int
[sin amplitude track 1]	int
[cos amplitude track 1]	int
[sin amplitude track 2]	int
[cos amplitude track 2]	int

S

This command returns the amplitudes of the sensor. i.e. sin and cos amplitudes of the 3 possible sensors

Syntax

{device} ***S***

Reply

[sin amplitude track 0] [cos amplitude track 0] [sin amplitude track 1] [cos amplitude track 1] [sin amplitude track 2] [cos amplitude track 2]

Examples

GetAmplitude

1 ***S***

SC

This command returns the corrected amplitudes of the sensor. i.e. sin and cos amplitudes of the 3 possible sensors

Syntax

{device} ***SC***

Reply

[sin amplitude track 0] [cos amplitude track 0] [sin amplitude track 1] [cos amplitude track 1] [sin amplitude track 2] [cos amplitude track 2]

Sensor dependency

not storable

Commands

Properties

Type: int

sddump

Syntax

[eeprom update] {device} ***sddump***

Reply

[eeprom update]

sdupdate

Syntax

[eeprom update] {device} ***sdupdate***

Reply

[eeprom update]

Sensor temperature

🔒

read-only

Commands

Properties

Type: double

gettemp

Returns the actual ***Sensor temperature***.

Syntax

{device} ***gettemp***

Reply

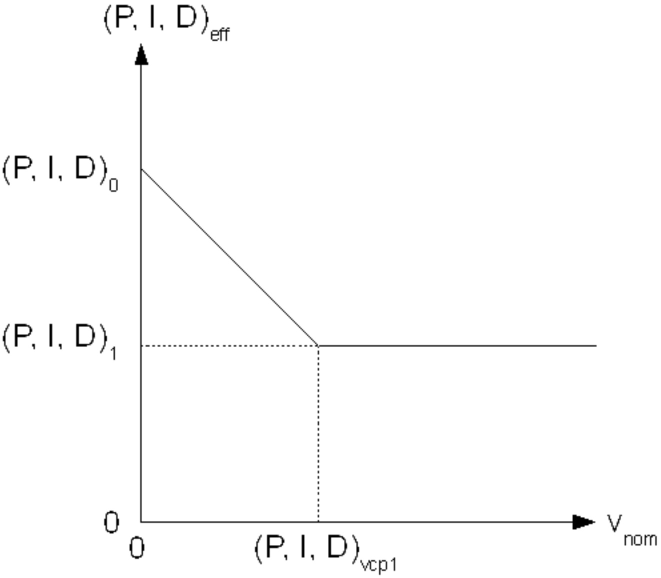
[temperature]

Servo

Adaptive positioning control

storable

Configuration parameters used for positioning control when dynamic regulator mode is selected. In dynamic mode, the user can have the effective factors of P, I, and D portions vary individually, depending on the current nominal velocity. There is a halt setting (P_0, I_0, D_0), a control point velocity (P_{vcp1}, I_{vcp1}, D_{vcp1}), and a terminal setting (P_1, I_1, D_1) for each of the three parameters. When nominal velocity is zero, the respective halt setting is taken as the effective value of each of the parameters. With increasing nominal velocity, the effective value varies linearly in a way that it equals the terminal setting when the nominal velocity reaches the respective control point velocity. With the nominal velocity exceeding the control point velocity, the respective terminal setting is effective.



Note that in dynamic mode, ***Servo control*** (p. 173) settings are still effective, except for the P, I and D factor settings being replaced by the values calculated here.

See ***Positioning control mode*** (p. 163) for positioning controller mode selection.

Commands

Properties

i	**Name**	**Type**	**Unit**
1	[P0]	double	-
2	[I0]	double	-
3	[D0]	double	-
4	[Pvcp1]	double	mm/s
5	[Ivcp1]	double	mm/s
6	[Dvcp1]	double	mm/s
7	[P1]	double	-
8	[I1]	double	-
9	[D1]	double	-

getadaptive

Returns the current setting of the ***Adaptive positioning control***.

Syntax

[*i*] {device} ***getadaptive***

Reply

[*Value*]

setadaptive

Sets the ***Adaptive positioning control***.

Syntax

[*Value*] [*i*] {device} ***setadaptive***

Positioning control mode

storable

Determines if the positioning regulator loop is closed and, if so, in which way the regulator works.

There are two modes of operation:

- In standard mode, ***Servo control*** (p. 173) settings are used for position control. The effective P, I, and D factors are taken directly from the respective parameter settings, thus constant throughout operation.
- In dynamic mode, ***Servo control*** settings are still effective, except for the P, I and D factor settings. The effective P, I, and D factors vary over nominal velocity. Their calculation is based on the parameter settings of ***Adaptive positioning control*** (p. 159) .

value	description
0	regulator loop is open
1	standard regulator mode
2	dynamic regulator mode

Commands

Properties

Type: int

setcloop

Sets the ***Positioning control mode***.

Syntax

[state] {device} ***setcloop***

Examples

	Command	Description
1:	2 1 ***setcloop***	Sets the device 1 closed loop mode at 2.

The positioning regulator will operate in dynamic mode.

---

getcloop

Returns the current setting of the ***Positioning control mode***.

Syntax

{device} ***getcloop***

Reply

[state]

Sensor assignment

storable

Selection of relevant position sensor. There is exactly one position sensor assigned to each Axis device, specified by the sensor device (port to which the sensor interface is connected) and subdevice (sensor selection) indexes. We also refer to this assignment as "sensor mapping" or "sensor routing".

For proper operation of positioning control, make sure that

- sensor selections and connections made are consistent
- sensor properties and respective parameter settings are consistent

Note that if the sensor interface connected to the selected port supports trigger functionality, the setting also affects the mapping of the trigger hardware. This is important because the trigger function support may be tied to one particular subdevice by hardware specification.

There is no particular parameter setting for actually "unmapping" or disconnecting an Axis device from any position sensor physically available. To achieve this, any open/unused position sensor channel can be assigned.

Commands

Properties

Name	Type	Description
[sensor device]	int	position measurement device
[sensor subdevice]	int	position measurement subdevice

setassignment

Sets the ***Sensor assignment***.

For changing the assignment,

- first, open the positioning control loop using ***setcloop*** (p. 163)
- make sure all trigger functions are deactivated
- then change the setting applying ***setassignment***
- store the new setting with the position control loop still open using ***save***
- apply ***reset*** (p. 34) and wait until axis is powered up again
- check for proper and consistent connections and parameterization, correct if need be
- close the positioning control loop using ***setcloop***
- check if loop works properly, fix problems if necessary
- last, store the new setting with the position control loop now closed using ***save***

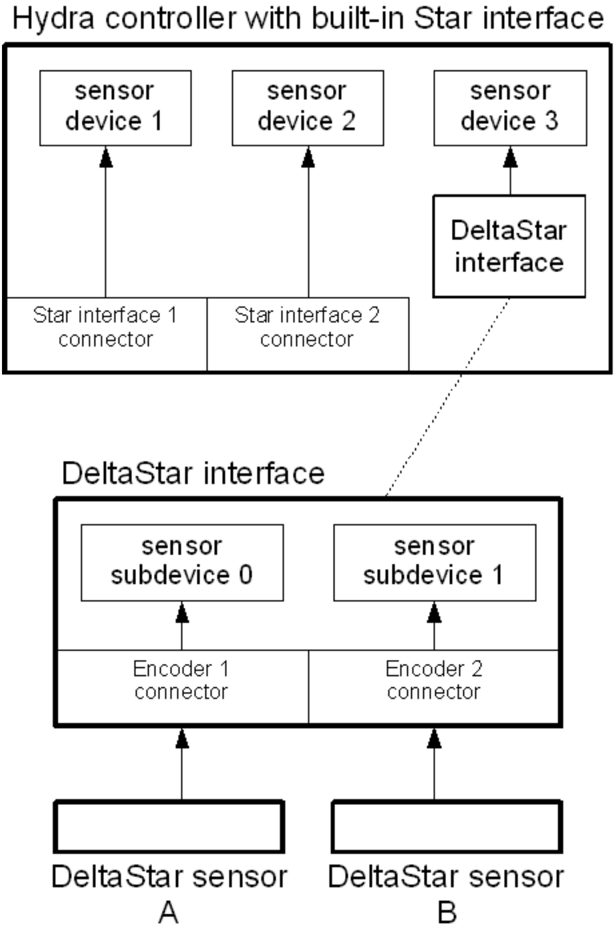
Syntax

[sensor subdevice] [sensor device] {device} ***setassignment***

Examples

Example 1

Preconditions: The Hydra controller is equipped with a built-in DeltaStar sensor interface to which two DeltaStar sensors A and B are connected as shown below.

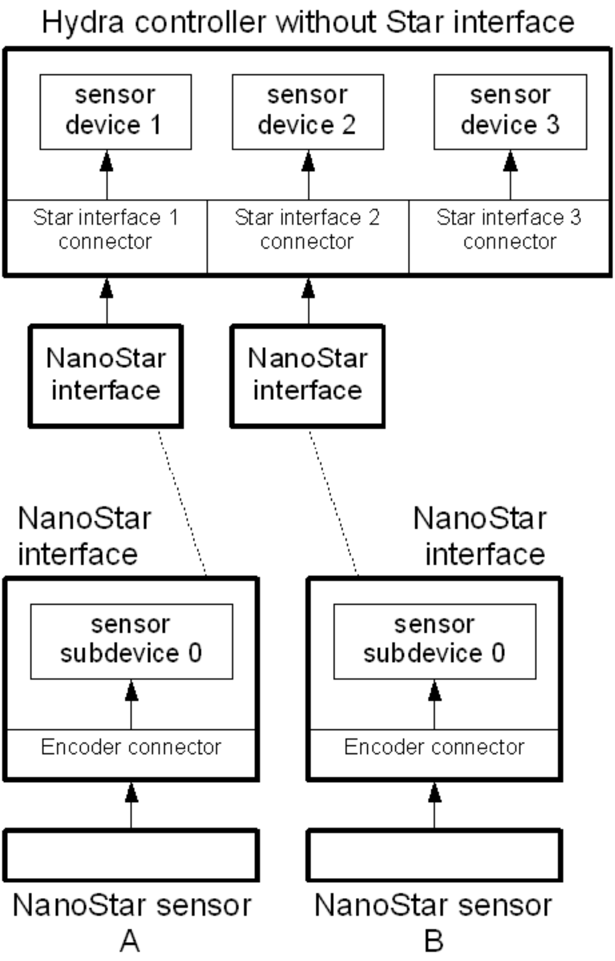


Task: The sensor A is to be assigned to axis device 1, sensor B to axis device 2.

	Command	Description
1:	0 3 1 ***setassignment***	Assign sensor A (subdevice 0, device 3) to axis 1.
2:	1 3 2 ***setassignment***	Assign sensor B (subdevice 1, device 3) to axis 2.

Example 2

Preconditions: Two external NanoStar interfaces are connected to the Hydra controller, to which again two NanoStar sensors A and B are connected as shown below.



Task: The sensor A is to be assigned to axis device 1, sensor B to axis device 2.

	Command	Description
1:	0 1 1 ***setassignment***	Assign sensor A (subdevice 0, device 1) to axis 1.
2:	0 2 2 ***setassignment***	Assign sensor B (subdevice 0, device 2) to axis 2.

getassignment

Returns the current setting of the ***Sensor assignment***.

Syntax

{device} ***getassignment***

Reply

[sensor subdevice] [sensor device]

Sensor status

read-only

Commands

Properties

Type: int

getsensorstatus

Returns the actual ***Sensor status***.

Syntax

{device} ***getsensorstatus***

Reply

[sensor status]

sensorreconnect

Syntax

{device} ***sensorreconnect***

Reply

[sensor status]

Servo control

storable

Standard PID positioning regulator ***(P factor, I factor, D factor)*** with some extra function for linear motor drive operation ***(boost value, load distance)*** . Additionally, the regulator can be limited regarding the integral portion ***(max I vel, I limit)*** . If nominal position and executive position deviate by more than what ***max pos error*** specifies, the device will run into emergency off state (see also ***Motor restart*** (p. 145)).

With either of the parameters ***boost value, load distance, max I vel, I limit,*** and ***max pos error*** set to 0, the respective function is disabled.

Adaptive positioning control (p. 159) is an option to vary P, I and D factors dynamically depending on the nominal velocity.

The ***boost value*** and ***load distance*** parameters have to be handled with care. It is advised to use the ***Motor current limit*** (p. 133) option before changing these.

The change of ***P factor, I factor, max I vel*** and ***I limit*** parameters while active may lead to a temporary discontiniuity in position control which results in a minor motor axis leap.

When operating as ***Adaptive positioning control*** with ***max I vel*** set, the effective velocity limitation will vary with the effective I factor as follows:

$vIMax_{eff}$ = ***max I vel*** * I_{eff} / ***I factor***

When driving a linear motor, having velocity limitation enabled (***max I vel*** other than 0) may lead to improper driving.

For activation and deactivation of position regulation in either mode (standard or adaptive), see ***Positioning control mode*** (p. 163) .

Commands

Properties

i	Name	Type	Unit	Description
1	[P factor]	double	-	coefficient of proportional portion
2	[I factor]	double	-	coefficient of integral portion
3	[D factor]	double	-	coefficient of differential portion
4	[I limit]	double	mm*s	absolute integral portion limit
5	[boost value]	double	-	factor for load dependent voltage boost (linear motor drive only)
6	[load distance]	double	mm	motor load distance limit (linear motor drive only)
7	[max pos error]	double	mm	position deviation threshold for emergency function
8	[max I vel]	double	mm/s	absolute variation velocity limit of integral portion (stepper motor drive only)

setsp

Configures the ***Servo control***.

Syntax

[*Value*] [*i*] {device} ***setsp***

getsp

Returns the current setting of the ***Servo control***.

Syntax

[*i*] {device} ***getsp***

Reply

[*Value*]

Target window

storable

Width of entrance and exit windows for the position settlement indication or in-window bit (see ***Axis status*** (p. 182) Bit 5). As soon as the absolute position aberration constantly remains below the specified entrance window width for a period of time specified by ***Time on target*** (p. 179) , the in-window bit changes from inactive to active state to indicate that the position has settled. As soon as the absolute position aberration once exceeds the exit window width, the indication bit changes from active to inactive state. Useful for closed loop operation only. Inactive if any parameter is set to 0.

Commands

Properties

Name	Type	Unit
[exit width]	double	mm
[entrance width]	double	mm

getclwindow

Returns the current setting of the ***Target window***.

Syntax

{device} ***getclwindow***

Reply

[entrance width] [exit width]

setclwindow

Sets the ***Target window***.

Syntax

[entrance width] [exit width] {device} ***setclwindow***

Time on target

storable

Position settling time for the in-window indication feature (see ***Target window*** (p. 177)).

Commands

Properties

Type: double
Unit: s

setclwintime

Sets the ***Time on target***.

Syntax

[value] {device} ***setclwintime***

getclwintime

Returns the current setting of the ***Time on target***.

Syntax

{device} ***getclwintime***

Reply

[value]

Status and position

Axis status

read-only

Actual status of the axis device. The bits of the returned status word have the following meaning:

Bit number	description
0	axis moving
1	manual move running
2	one or more machine errors occurred *
3	reserved
4	reserved
5	actual position in defined window
6	reserved for optional motionlimits
7	emergency stopped
8	motor power disabled
9	emergency-off switch active *
10	device busy - move commands discarded
11...30	reserved
31	invalid status - reset required**

* status depends on controller, not on singular axis device
** usually occurs when the emergency off switch is engaged during powerup

If all machine errors are read off the machine error stack then bit 4 is 0, but the condition for the error may not be removed (emergency stop is active). In all those cases the bit 8 is active and shows this situation. As soon as the emergency condition is removed, the ***Motor restart*** (p. 145) feature can be utilized to reenable the power stage of the device without a complete hardware or software reset (as necessary with some other SMC controllers).

For watching the complete system without decoding each singular axis, see ***Controller status*** (p. 186) .

Commands

Properties

Type: int

est

Returns the actual ***Axis status***.

Syntax

{device} ***est***

Reply

[status]

nst (nstatus)

Returns the actual ***Axis status***.

Syntax

{device} ***nst***

Reply

[status]

ast

This command is an extended form of the ***nst*** . The lower 16 bits represents the status of the device. The next 10 bits represents the last machine error code and the next bits are for the devicenr with caused the error. The transmition of the return value is delay until the last move stops in the opposite to the command ***est*** (p. 183) which delivers the status immediately. ^C CRLF (control C only on one line finisch with CR and LF characters) aborts the move execution on one or both axes and this command ***ast*** delivers the status. (since Version 1.71)

Syntax

{device} ***ast***

Reply

[status]

Controller position

read-only

Integrated ***Device position*** (p. 190) of both axis devices. The single positions displayed conform the respective ***Position display selection*** (p. 191) settings.

Commands

Properties

Name	Type
[position X]	double
[position Y]	double

p

Returns the actual ***Controller position***.

Syntax

p

Reply

[position X] [position Y]

Controller status

🔒

read-only

Partially a controller device status indication, partially a bitwise linkage of all ***Axis status*** (p. 182) registers which varies from one flag to another.

Bit number	description	linkage
0	axis moving	axis disjunction
1	manual move running	axis disjunction
2	one or more machine errors occurred	controller
3	reserved	-
4	reserved	-
5	actual position in defined window	axis conjunction
6	reserved for optional motionlimits	-
7	emergency stopped	axis disjunction
8	motor power disabled	axis disjunction
9	emergency-off switch active	controller
10	device busy - move commands discarded	axis disjunction
11...30	reserved	-
31	invalid status - reset required*	axis disjunction

* usually occurs when the emergency off switch is engaged during powerup

- An **axis disjunction** means the controller status flag will be set if **at least one** of the corresponding ***Axis status*** flags is set.
- An **axis conjunction** means the controller status flag will be set if **all** corresponding ***Axis status*** flags are set.
- A **controller linkage** means the corresponding flag is global, i.e. the bit setting is always equal in both the ***Axis status*** and ***Controller status*** (p. 186) registers.

Commands

Properties

Type: int

st (status)

Returns the actual ***Controller status***.

Syntax

st

Reply

[status]

Examples

If ***st*** returns 1 it means that at least one of the axis devices is executing a programmed move, and that none of the other status bits is active at either device.

Device position

read-only

Actual device position, displayed selectively as the nominal or measured position (see ***Position display selection*** (p. 191)).

Commands

Properties

Type: double
Unit: mm

---

np

Returns the actual ***Device position***.

Syntax

{device} ***np***

Reply

[position]

Position display selection

storable

Selection of position displayed as ***Device position*** (p. 190) .

Commands

Properties

Type: int

value	selection
0	nominal position
1	measured position

setselpos

Sets the ***Position display selection***.

Syntax

[position source] {device} ***setselpos***

getselpos

Returns the current setting of the ***Position display selection***.

Syntax

{device} ***getselpos***

Reply

[position source]

Switches and reference mark

Reference configuration

storable

Defines the mode of operation for reference position mark detection. The setting takes effect upon execution of ***refmove*** (p. 112) and ***nrefmove*** (p. 117).

Commands

Properties

Type: int

value	mode
0	detection on rising edge
1	detection on falling edge
2	detection disabled

getref

Returns current setting of ***Reference configuration***.

Syntax

{device} ***getref***

Reply

[config]

setref

Sets ***Reference configuration***.

Syntax

[config] {device} ***setref***

Examples

	Command	Description
1:	1 1 ***setref***	Enables device 1 reference position mark detection on falling edges.

Upon execution of ***refmove*** (p. 112) or 1 ***nrefmove*** (p. 117), device 1 will brake as soon as a reference signal high-to-low transition occurs, then return to respective position.

Reference status

🔒 read-only

Status of reference mark detection. Bits 0 and 1 are affected by ***Reference move*** (p. 117) only. Once set, they stay set until next execution of ***Reference move***.

Commands

Properties

Type: int

bit	value	description
0	1	reference detected if 1
1	2	end switch detected if 1
2	4	reference active if 1

getrefst

Returns ***Reference status***.

Syntax

{device} ***getrefst***

Reply

[status]

Switch configuration

storable

Configuration of end switches. Options are normally open (NO), normally closed (NC) or disabled.

⚠ The reference potential for each of the switches is selected by hardware (configuration plug).

Commands

Properties

i	Name	Type	Description
0	[calibration sw.]	int	0 = NO, 1 = NC, 2 = disabled
1	[range measure sw.]	int	0 = NO, 1 = NC, 2 = disabled

setsw

Sets the ***Switch configuration***.

Syntax

[*Value*] [*i*] {device} ***setsw***

getsw

Returns the current setting of the ***Switch configuration***.

Syntax

[*i*] {device} ***getsw***

Reply

[*Value*]

Switch status

not storable

States of end switches.

Commands

Properties

Name	Type	Description
[range measure sw.]	int	0 = inactive, 1 = active
[calibration sw.]	int	0 = inactive, 1 = active

getswst

Returns the actual ***Switch status***.

Syntax

{device} ***getswst***

Reply

[calibration sw.] [range measure sw.]

Trigger

If your Hydra controller is equipped with a DeltaStar position sensor interface (NOT the DeltaStar Eco model), trigger signal output and position capture function is available. Each single trigger unit is part of a particular DeltaStar subdevice. Just as with position query, the trigger unit is not directly accessed by index of the respective subdevice, but by index of the Hydra axis device it is assigned to. The overview on the following page depicts the standard and optional routings. If needed, the routing can be changed by ***Sensor assignment*** (p. 165) .

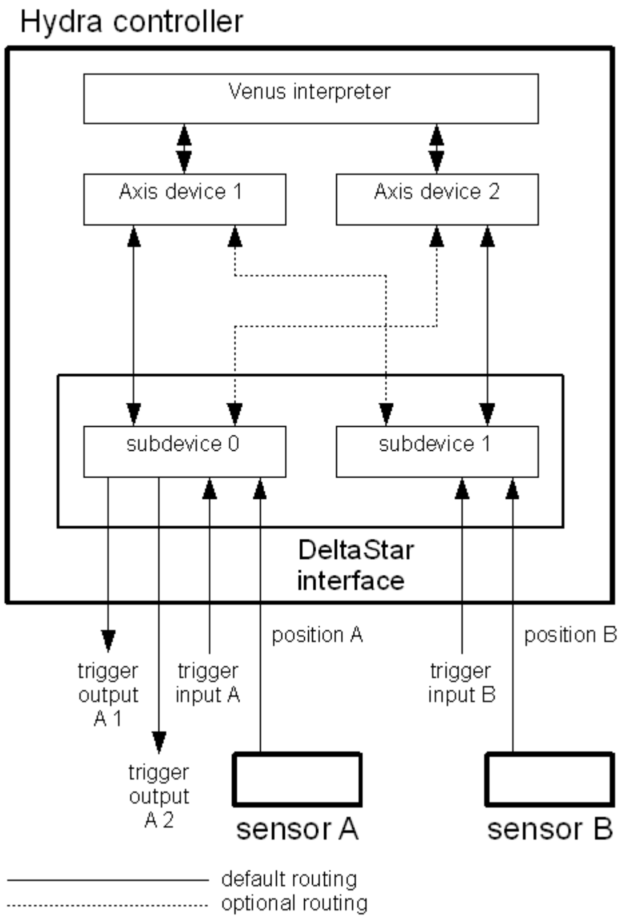
The DeltaStar subdevice 0 offers one trigger signal input (**trigger input A**) for position capture and two trigger signal outputs (**trigger output A 1**, **trigger output A 2**). Subdevice 1 offers one trigger signal input (**trigger input B**) only; subdevice 1 trigger outputs are physically available, but not driven.

1. Output trigger function

The Hydra output trigger concept distinguishes generation of trigger events from the generation of the actual signal output as shown below.

The **trigger event generator** puts out a single trigger event whenever certain conditions (mostly concerning the currently measured slide or rotor position) are met. ***Trigger event setup*** (p. 221) provides for definition of these conditions.

The trigger event generator simultaneously drives a suitable number of **trigger outputs** (up to 2 with the DeltaStar interface) which form trigger pulses out of each event, individually configurable regarding such parameters as ***Trigger output polarity*** (p. 232), ***Trigger output pulse width*** (p. 234), and ***Trigger output delay*** (p. 230). The overall result will be the output of several synchronized trigger output signals.



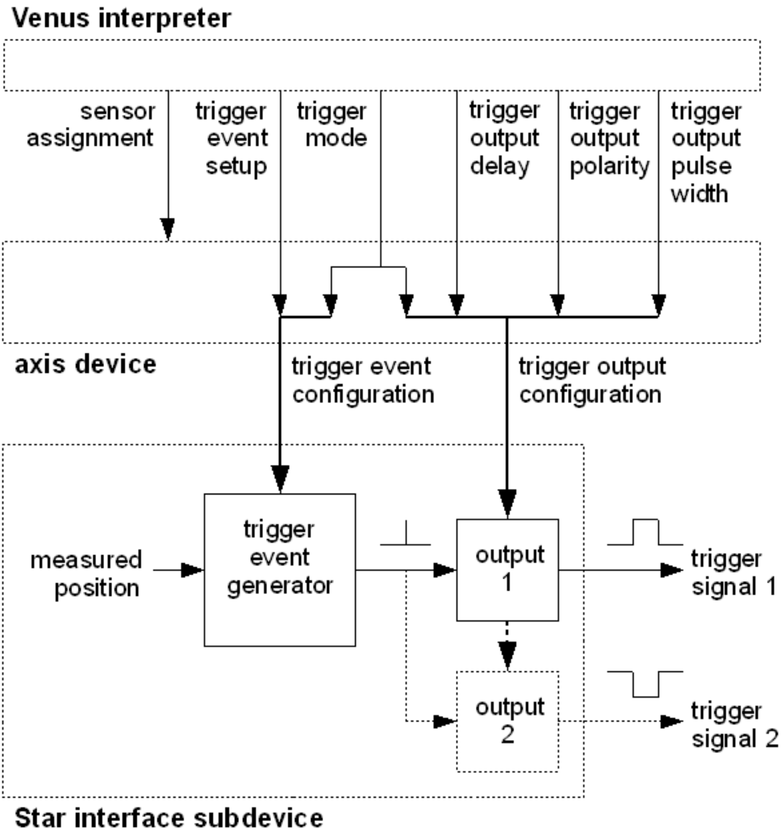
signal name	hardware manual pin description
trigger output A 1	+TRIG_Out1_A
trigger output A 2	+TRIG_Out2_A
trigger input A	+TRIG_IN_A
trigger input B	+TRIG_IN_B

trigger overview

Note that

- whatever configuration options the Hydra firmware may offer, support is depending on the respective DeltaStar logic/firmware revision
- the DeltaStar interface features merely one trigger unit at subdevice 0, driving up to 2 trigger signal outputs

Trigger mode (p. 223), at last, allows for trigger activation and selection of different operation modes. It affects both the **trigger outputs** and the **trigger event generator**.



Example 1

Preconditions: The Hydra controller is equipped with a DeltaStar sensor interface. The subdevice 0 is assigned to axis device 1 (see ***Sensor assignment***), so axis 1 is the position source. Current position at axis 1 is 0.

Task: Move axis 1 to 100 mm, trigger output 1 thereby generating 9 active high 5 µs wide trigger pulses, starting at 10 mm, stopping at 90 mm, delayed by 8 µs; direction signal at trigger output 2, start polarity high.

Note: Unlike in the table below, all commands have to be entered *in one line*.

	Command	Description
1:	5 1 1 ***settroutpw*** (p. 234)	Set trigger pulse width at output 1 at axis 1 to 5 µs.
2:	8 1 1 ***settroutdelay*** (p. 230)	Set trigger delay at output 1 at axis 1 to 8 µs.
3:	0 1 1 ***settroutpol*** (p. 233)	Specify trigger polarity at output 1 at axis 1 to be active high.
4:	1 2 1 ***settroutpol***	Set start polarity at output 2 at axis 1 high.
5:	3 1 ***settr*** (p. 228)	Arm axis 1 trigger for generation of equidistant trigger pulses at output 1 and the direction signal at output 2.
6:	10 90 9 1 ***settrpara*** (p. 221)	Specify first and last trigger events to occur at axis 1 positions 10 mm and 90 mm, overall number of equidistant pulses being 9 (=> trigger interval = 10 mm).
7:	100 1 ***nm***	Move axis 1 to 100 mm.

Result: Trigger signals will be put out as specified while axis is moving.

Example 2

Preconditions: The Hydra controller is equipped with a DeltaStar sensor interface; the subdevice 0 is assigned to axis device 1 (see ***Sensor assignment***).

Task: Put out a continuous 500 Hz signal at both trigger outputs, 3 µs pulsewidth at both outputs, active high at output 1, inverted and delayed by 7 µs at output 2.

	Command	Description
1:	3 1 1 ***settroutpw***	Set trigger pulse width at output 1 at axis 1 to 3 µs.
2:	3 2 1 ***settroutpw***	Set trigger pulse width at output 2 at axis 1 to 3 µs.
3:	7 2 1 ***settroutdelay***	Set trigger delay at output 2 at axis 1 to 7 µs.
4:	0 1 1 ***settroutpol***	Specify trigger polarity at output 1 at axis 1 to be active high.
5:	1 2 1 ***settroutpol***	Specify trigger polarity at output 2 at axis 1 to be active low.
6:	5 1 ***settr***	Arm axis 1 trigger for continuous mode.

Result: Trigger signals will be put out immediately as specified.

2. Triggered position capture function

Each axis device of the Hydra controller holds a buffer of its own to take a number of captured position values from the DeltaStar subdevice assigned by ***Sensor assignment***. The capture function can be enabled by ***Trigger capture mode*** (p. 212) for *one subdevice at a time*. An external trigger signal has to be provided at the selected subdevice's trigger input.

With the position capture function enabled at either subdevice, the DeltaStar interface watches the respective signal input. Whenever a trigger event (for definition of input trigger events see ***Trigger capture polarity*** (p. 214)) occurs at a specified DeltaStar signal input, the current position value at the associated DeltaStar subdevice is latched at the first free location of the associated Hydra axis device's position buffer. As soon as a capture sequence is complete, the position values can be read out via ***Trigger capture position*** (p. 216) .

⚠ Note that whenever the capture function is enabled at either subdevice, the measured position at subdevice 1 forwarded to the associated axis device's position controller will freeze for 500 µs duration with the occurrance of each single trigger input event. This may cause a discontinuity in positioning if that axis is moved simultaneously in closed loop mode. Therefore it is strongly recommended to either drive the axis in open loop mode or - if possible - keep the axis in standstill while a capture sequence is running.

Example 3

Preconditions: The Hydra controller is equipped with a DeltaStar sensor interface. The subdevice 0 is assigned to axis device 1 (see ***Sensor assignment***). Current position at axis 1 is 0. Velocity and acceleration settings are 250 mm/s and 1000 mm/s^2, respectively. External active low trigger signal is provided at trigger input A. The trigger source is directly synchronized to any move start. Controller IP address is 192.168.129.200. A TFTP command line tool is installed on the local host PC.

Task: Move axis 1 to 100 mm. Record the course of the slide/rotor by capturing the position values at time intervals of 0.1 s. A file of the record is to be stored in the local host file *record01.txt*.

Preparation: With the above settings, the move will take 0.6 s to execute. So - proper action provided - 7 position values will be captured. For security, we set the buffer size to 10. Trigger source is to be set to 10 pulses/s.

Note: Unlike in the table below, all commands have to be entered *in one line*.

	Command	**Description**
1:	0 1 ***settrinpol*** (p. 215)	Match trigger input to the active low trigger source by making it sensitive to falling edges.
2:	10 1 ***settrinsize*** (p. 209)	Prepare capture buffer to record up to 10 position values.
3:	1 1 ***settrin*** (p. 212)	Arm trigger capture function.
4:	100 1 ***nm***	Move axis 1 to 100 mm.

Result: Axis is moving, trigger pulses roll in at 0, 0.1 ... 0.6 s. Wait for finish. Then continue.

	Command	**Description**
1:	0 1 ***settrin***	Disarm position capturing.
2:	1 ***gettrindex***	Query number of captured position values. Should be 7 at least.

Query captured positions. We assume perfect action.

	Command	Description
1:	0 1 ***gettrinpos*** (p. 216)	Reply: 0.000000
2:	1 1 ***gettrinpos***	Reply: 6.250000
3:	2 1 ***gettrinpos***	Reply: 25.000000
4:	3 1 ***gettrinpos***	Reply: 50.000000
5:	4 1 ***gettrinpos***	Reply: 75.000000
6:	5 1 ***gettrinpos***	Reply: 93.750000
7:	6 1 ***gettrinpos***	Reply: 100.000000

To store the record as a text file on the host PC, firstly apply:

	Command	Description
1:	1 ***filetrinpos*** (p. 217)	Store text file of the above record.

Result: File captureposition.txt has been stored in Hydra RAM file system.

To store the record in a text file on the host PC, first open command line box and change to the target directory. Then type in (*one line*):

tftp -i 192.168.129.200 GET /ram/captureposition.txt record01.txt

Result: File record01.txt has been stored in the local target directory. It contains a list of all positions captured.

Trigger capture buffer size

not storable

Maximum number of position values to be latched by trigger capture function.

Commands

Properties

Type: int

settrinsize

Sets the ***Trigger capture buffer size***.

Syntax

[size] {device} ***settrinsize***

gettrinsize

Returns the current setting of the ***Trigger capture buffer size***.

Syntax

{device} ***gettrinsize***

Reply

[size]

Trigger capture index

not storable

Number of position values captured during last trigger capture sequence. If a trigger sequence is running at the time of query, number of positions captured so far.

Commands

Properties

Type: int

gettrinindex

Returns the current ***Trigger capture index***.

Syntax

{device} ***gettrinindex***

Reply

[index]

Trigger capture mode

not storable

Enable state of the trigger capture function.

value	capture function
0	off
1	on

Commands

Properties

Type: int

settrin

Sets the ***Trigger capture mode***.

Syntax

[enabled] {device} ***settrin***

gettrin

Returns the current setting of the ***Trigger capture mode***.

Syntax

{device} ***gettrin***

Reply

[enabled]

Trigger capture polarity

not storable

Defines the polarity of level transitions at the respective trigger input upon which position values will be captured.

value	capture event
0	rising edge (positive transition)
1	falling edge (negative transition)

Commands

Properties

Type: int

gettrinpol

Returns the current setting of the ***Trigger capture polarity***.

Syntax

{device} ***gettrinpol***

Reply

[input polarity]

settrinpol

Sets the ***Trigger capture polarity***.

Syntax

[input polarity] {device} ***settrinpol***

Trigger capture position

not storable

Position value latched by trigger capture function at a given index.

Commands

Properties

Name	Type
[index]	int
[position]	double

gettrinpos

Returns the ***Trigger capture position*** stored at the specified index.

Syntax

[index] {device} ***gettrinpos***

Reply

[position]

Trigger capture position file

not storable

Text file on the Hydra RAM file system containing a list of all position values recorded during the last position capture sequence, written upon ***filetrinpos*** (p. 217) command. With a TFTP command line tool installed on the host PC, the contents of the file can be downloaded and stored in a specified local text file as follows:

tftp -i *ip* GET /ram/captureposition.txt *target*

where

- ***ip*** is the IP address of the Hydra controller
- ***target*** is the name of the file on the local host

As opposed to using ***gettrinpos*** (p. 216), this saves communication time, and is particularly useful when a record contains a large number of captured values.

Commands

filetrinpos..217

Properties

Type: int

filetrinpos

Writes ***Trigger capture position file***.

The error code returned can be decoded as follows:

value	error
0	no error - execution successful
-1	write error
-2	capture buffer empty
-3	trigger function not available
-4	no Star interface assigned

Syntax

{device} ***filetrinpos***

Reply

[error code]

Trigger delay

not storable

Delay of pulses at the second trigger signal output.

For compatibility purpose only - not for further use. Use ***Trigger output delay*** (p. 230) instead.

Commands

Properties

Type: int
Unit: 0.5 µs ticks

gettrdelay

Returns the current setting of the ***Trigger delay***.

Syntax

{device} ***gettrdelay***

Reply

[delay]

settrdelay

Sets the ***Trigger delay***.

Syntax

[delay] {device} ***settrdelay***

Trigger event setup

not storable

Trigger event generator setup, valid with **equidistant mode** (see ***Trigger mode*** (p. 223)).

Commands

Properties

Name	Type	Unit	Description
[start position]	double	mm	first trigger position
[stop position]	double	mm	last trigger position
[number]	int	-	overall number of trigger pulses

settrpara

Sets the ***Trigger event setup***. Additionally arms the trigger if it has not already been armed by the ***settr*** command.

Syntax

{device} ***settrpara***

gettrpara

Returns the current setting of the ***Trigger event setup***.

Syntax

{device} ***gettrpara***

Reply

[start position] [stop position] [number]

Trigger mode

not storable

General output trigger operation mode and activation state.

As for trigger event generation, there are 2 modes:

- the **equidistant mode**
- the **continuous mode**

As for the configuration of the second trigger output, there are 2 modes available as well:

- the **standard mode**
- the **direction mode**

Available mode combinations and parameterization:

value	trigger events	output 2 signal
0	none (off)	no output (off)
1	equidistant	standard
2	continuous	standard
3	equidistant	direction

With the **equidistant mode**, the occurrence of trigger events depends on the course of the slide or rotor position currently measured at the respective sensor subdevice. Each position meeting the trigger conditions at a time results in one single trigger event. With the trigger function enabled and the slide/rotor moving in a given direction, the trigger event generator

- gets active generating a trigger event as soon as the current nominal position reaches a given start position (trigger sequence start)
- generates further single trigger events at steps of a constant position interval **d**
- gets inactive generating a trigger event as soon as the current nominal position reaches a given stop position (trigger sequence termination)

The trigger event parameter set includes

- the overall number of trigger pulses **n**
- the start position $\mathbf{s_1}$
- the stop position $\mathbf{s_n}$

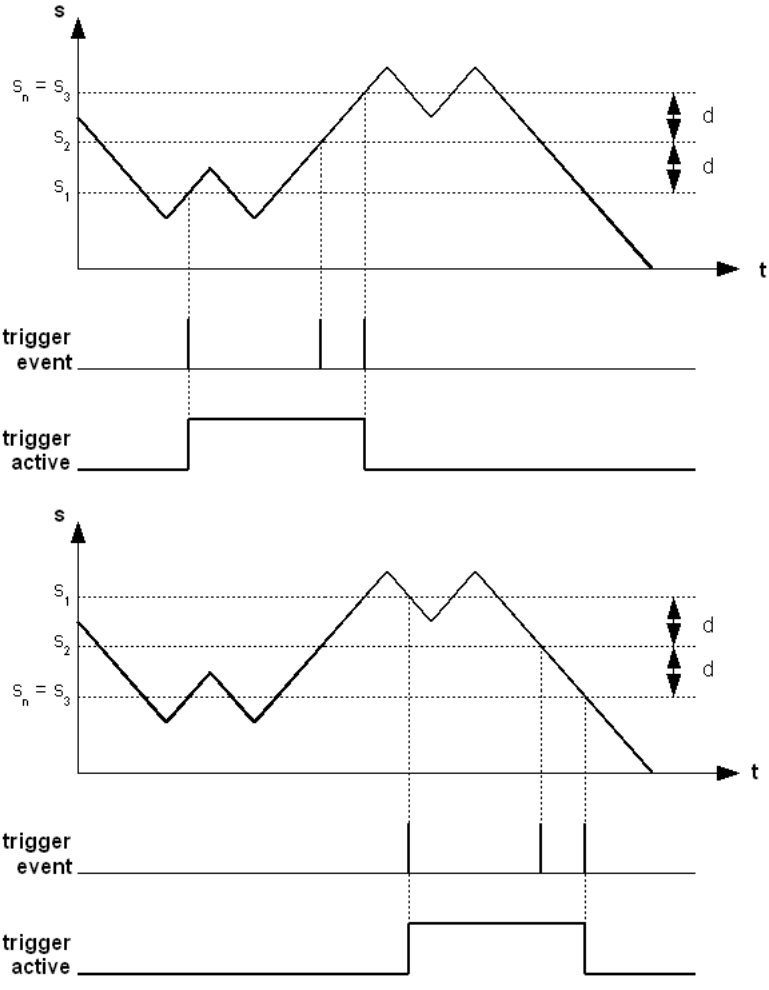
The absolute trigger position interval **d** and the appropriate moving direction are determined by **n** and the distance between $\mathbf{s_n}$ and $\mathbf{s_1}$. The trigger events are always processed in one direction from $\mathbf{s_1}$ through $\mathbf{s_n}$, as shown in the example s/t profile (**n** = 3) on the following page.

See ***Trigger event setup*** (p. 221) for equidistant trigger event configuration.

For query whether a trigger sequence is currently active (running), see ***Trigger status*** (p. 238) .

Note that if the trigger function is being switched off while a trigger sequence is running, the latter will not finish, but the trigger pulse output will stop immediately.

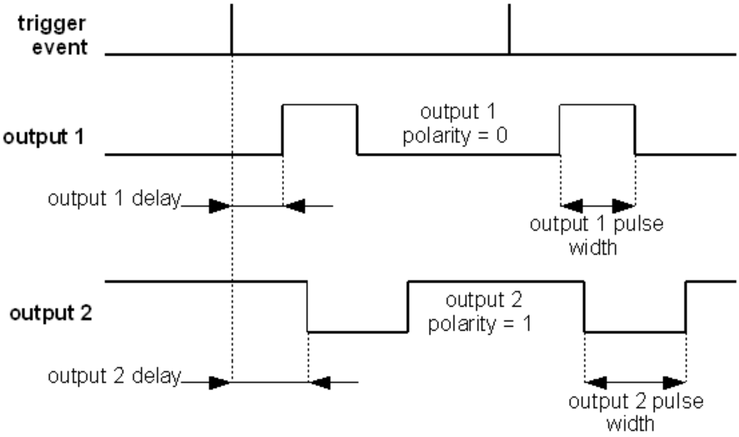
Note that as soon as a trigger sequence has finished or been stopped before reaching the final trigger event, the trigger function has to be reenabled to start the next one.



Equidistant trigger mode principle

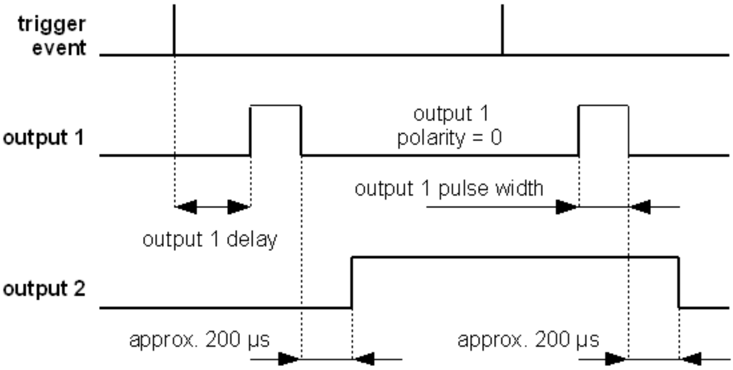
With the **continuous mode**, the trigger event generator puts out continuous asynchronous events at a constant frequency of approx. 500 Hz. It does not need further parameterization.

With the **standard mode**, both outputs operate equally, individually configurable regarding ***Trigger output delay*** (p. 230) , ***Trigger output pulse width*** (p. 234) and ***Trigger output polarity*** (p. 232) .



Standard mode principle

With the **direction mode**, output 1 operates as it does in standard mode, whereas output 2 level alternates with each single trigger event according to the scheme below. ***Trigger output polarity*** specifies the start polarity, while output 2 ***Trigger output delay*** and ***Trigger output pulse width*** are void.



Direction mode principle

Generation of trigger output signals requires connection to matching hardware with trigger function support.

Commands

Properties

Type: int

settr

Sets the ***Trigger mode***.

Syntax

[mode] {device} ***settr***

gettr

Returns the current setting of the ***Trigger mode***.

Syntax

{device} ***gettr***

Reply

[mode]

Trigger output delay

not storable

Delay of trigger pulses - individual setting at the specified signal output.

Commands

Properties

i	Name	Type	Unit
1	[output 1 delay]	double	µs
2	[output 2 delay]	double	µs

settroutdelay

Sets the ***Trigger output delay***.

Syntax

[*Value*] [*i*] {device} ***settroutdelay***

gettroutdelay

Returns the current setting of the ***Trigger output delay***.

Syntax

[*i*] {device} ***gettroutdelay***

Reply

[*Value*]

Trigger output polarity

not storable

Polarity of trigger pulses - individual setting at the specified signal output.

value	standard mode	output 2 direction mode
0	active high	start low
1	active low	start high

Commands

Properties

i	Name	Type
1	[output 1 polarity]	int
2	[output 2 polarity]	int

gettroutpol

Returns the current setting of the ***Trigger output polarity***.

Syntax

[*i*] {device} ***gettroutpol***

Reply

[*Value*]

settroutpol

Sets the ***Trigger output polarity***.

Syntax

[*Value*] [*i*] {device} ***settroutpol***

Trigger output pulse width

not storable

Width of trigger pulses - individual setting at the specified signal output.

Commands

Properties

i	Name	Type	Unit
1	[output 1 width]	double	µs
2	[output 2 width]	double	µs

settroutpw

Sets the ***Trigger output pulse width***.

Syntax

[*Value*] [*i*] {device} ***settroutpw***

gettroutpw

Returns the current setting of the ***Trigger output pulse width***.

Syntax

[*i*] {device} ***gettroutpw***

Reply

[*Value*]

Trigger pulse width

Unitary width of pulses at all trigger signal outputs.

not storable

For compatibility purpose only - not for further use. Use ***Trigger output pulse width*** (p. 234) instead.

Commands

Properties

Type: int
Unit: 0.5 µs ticks

settrwidth

Sets the ***Trigger pulse width***.

Syntax

[width] {device} ***settrwidth***

gettrwidth

Returns the current setting of the ***Trigger pulse width***.

Syntax

{device} ***gettrwidth***

Reply

[width]

Trigger status

not storable

Current trigger output status. When the output trigger function is active, goes 1 after first event of a trigger output sequence, and 0 after the last. For output trigger activation, see ***Trigger mode*** (p. 223) . For trigger event configuration, see ***Trigger event setup*** (p. 221) .

Commands

Properties

Type: int

gettrst

Returns the actual ***Trigger status***.

Syntax

{device} ***gettrst***

Reply

[status]

State Reference

Command Reference