
H a n d b o o k

hydra

Firmware: 3.200000
GMT: Wed Oct 12 11:15:00 2011

Contents

About this documentation.....	5
Introduction to Venus-3.....	7
Venus-3 is an interpreter language and combines the languages Venus-1 and Venus-2.....	8
History.....	8
Command syntax.....	9
Blocking and non blocking commands.....	11
Data reply message to the host.....	12
Broadcast commands.....	12
Parameter storage.....	13
Example of a typical program to control the hydra.....	13
Communication.....	14
Device structure.....	16
Position sensor routing.....	19
Motor commutation handling.....	23
Determination of rotor to motor angle offset.....	24
Auto commutation.....	25
Inaccuracies and motor optimization.....	26
Introduction to manual operation.....	29
Manual drivers and manual devices.....	30
Assignment example.....	34
Motion control arbitration.....	35
Complete examples.....	37
Introduction to trigger function.....	43

1. Output trigger function.....	45
Output trigger examples.....	48
2. Triggered position capture function.....	50
Triggered position capture example.....	51
Introduction to clock and direction operation.....	54
Clock and direction controlled motion function.....	55
Introduction to digital I/O processing.....	58
I/O organization.....	59
Action command function.....	62
Venus I/O access.....	67
Action command examples.....	69
Initial event generation.....	77
Initial event example.....	79
Introduction to command chain function.....	82
Devices.....	93
Controller [Device 0]	94
Axis [Device 1..2]	97
Sensor [Device 3]	100
Datatypes.....	104
Controller.....	107
ControllerIO.....	128
Dynamics.....	134
InputOutput.....	145

Interpreter.....	225
Manual operation.....	262
Mechanic.....	278
Mechanic setup.....	283
Motion.....	312
Motor.....	336
Safety functions.....	372
Sensoric.....	375
Servo.....	387
Status and position.....	416
Switches and reference mark.....	433
Trigger.....	443
State Reference.....	477
Command Reference.....	480

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 separated 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 transferred 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.

Ctrl+C move stop

Any move currently running at any motor axis will immediately be stopped upon application of the **Ctrl+C** short cut. The braking slope will be set according to either **Stop deceleration** (p. 137) or **Acceleration** (p. 135) - whatever setting is higher at the moment. To see if any further action goes with the short cut, see description of respective move command.

Blocking and non blocking commands

As opposed to former SMC controllers which feature command queues, Hydra has no more blocking commands. All commands will be executed immediately, regardless if the preceeding command has finished execution or not. This is especially relevant for programmed motion where, during a running move, the current target position is discarded and immediately replaced by a new one as soon as a new move request has been encountered.

Note that subsequent to such premature move abortion, there is a short period of time during which further move requests will be rejected. This condition is shown by the *device busy* flag in the respective **Axis status** (p. 417) register which will go high whenever move requests are blocked. Moreover, continued fast freewheeling move abortion (especially by **Ctrl+C** or **nabort** (p. 322)) can under certain rare circumstances lead to an operation state where the *axis moving* flag in the respective **Axis status** register will not return to inactive state when the final move has finished, and is therefore not recommended. Transmission

of the next sole move request, *Ctrl+C* or ***nabort*** will remove this condition.

Note further that the ***ast*** (p. 421) command can be utilized to block command execution until a running move has finished and produce an automatic status reply afterwards.

Producing an automatic status reply message

With the following sequence of instructions a synchronous status reply can be generated (applies here to the 1. device) :

	Command
1:	10.2 1 <i>nmove</i> <small>(p. 314)</small>
2:	0 1 <i>ast</i> <small>(p. 421)</small>

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 when requested by the host.

Broadcast commands

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

Parameter storage

Most parameter settings are storable. So they are not lost after power off. Use commands **nsave** (p. 245) or **csave** (p. 243) 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. 144)	Velocity setting, device-1
2:	5 2 snv	Velocity setting, device-2
3:		
4:	2 1 setpitch (p. 282)	Pitch setting, device-1
5:	2 2 setpitch	Pitch setting, device-2
6:	100 1 sna (p. 136)	Acceleration setting, device-1
7:		
8:	1 nsave (p. 245)	Save all parameters, device-1
9:	2 nsave	Save all parameters, device-2
10:		
11:	1 ncal (p. 286)	Move to endswitches, device-1
12:	1 nrm (p. 306)	(find limits)

	Command	Description
13:		
14:	2 <i>ncal</i>	Move to endswitches, device-2
15:		
16:		
17:	15 1 <i>nm</i> (p. 314)	Positioning absolute, device-1
18:		
19:	1 <i>nst</i> (p. 420)	Ask for status, device-1
20:	1 <i>gne</i> (p. 252)	Ask for command decoding error (venus error)
21:	1 <i>np</i> (p. 429)	Ask for the actual position of device 1
22:		
23:	2.003 2 <i>nm</i>	Positioning absolute, device 2
24:		
25:	2 <i>nst</i>	Ask for status, device 2
26:	2 <i>np</i>	Ask for actual position of device 2

Communication

Venus communication is available via

- the Ethernet interface
- the user RS232 port

Ethernet

With the Ethernet interface, ports 400 and 402 are simultaneously available. Default IP address is 192.168.129.200.



Shortcuts (like *Ctrl+C*) must be followed by a CR/LF line termination in order to work properly.



Host driver software must not apply the TCP_NODELAY option.

RS232

With the RS232 interface, connection settings are:

- variable baud rate; default is 38.4 kBaud
- 8 data bits
- no parity bit
- 1 stop bit
- flow control off

Shortcuts (like *Ctrl+C*) work without line termination.

Device structure

The firmware organizes the Hydra controller into 4 *Devices*. Each device is a unit with a unique index and a parameter and command set of its own. However, the individual parameter and command sets can overlap with each other. Generally, with each command, the Venus interpreter selects a specific device by its index.

The devices are in particular:

- the **Controller** device (index 0)
- the two **Axis** devices (index 1 and 2)
- the **Sensor** device (index 3)

The **Controller** device provides functionality that is not especially tied to any specific motor axis or position sensor. The device index (0) is not entered with the command line string, so the minimum command line contains only the command string itself. For instance, a command line querying the current controller status is written

st

The **Axis** devices provide functionality that is especially tied to either of the motor axes 1 and 2 *or* position sensor interface ports 1 and 2. The corresponding device index (1 or 2) has to be entered with each command line, so the minimum command line contains the command string and a device index. For instance, a command line querying the current status at motor axis 1 is written

1 nst

A command line querying the raw track amplitudes at the position sensor(s) connected to the sensor interface port 2 is written

2 S

It is evident that an **Axis** device command can target either a motor axis or a position sensor interface. This varies with each particular command. Since only a few commands

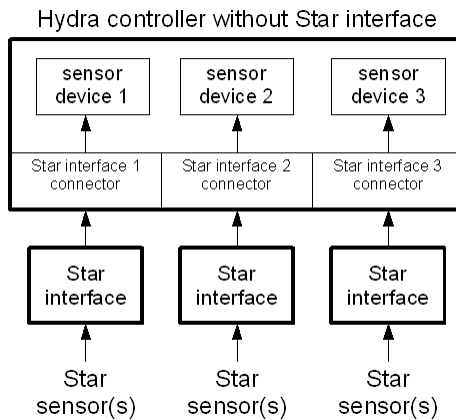
target sensor interfaces, the general rule is that the device index specifies the motor axis if not otherwise noted.

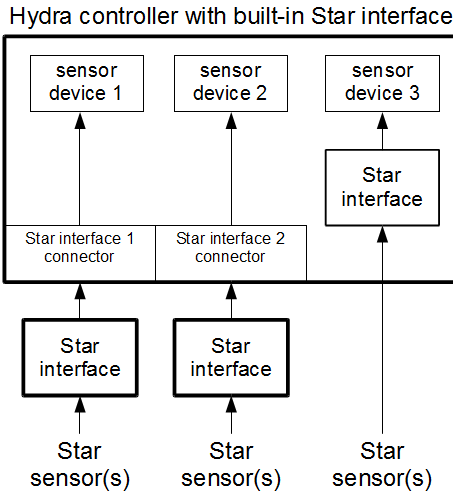
The **Sensor** device provides functionality that is especially tied to the position sensor interface port 3. The corresponding device index (3) has to be entered with each command line, so the minimum command line contains the command string and a device index. For instance, a command line querying the raw track amplitudes at the position sensor(s) connected to the sensor interface port 3 is written

3 S

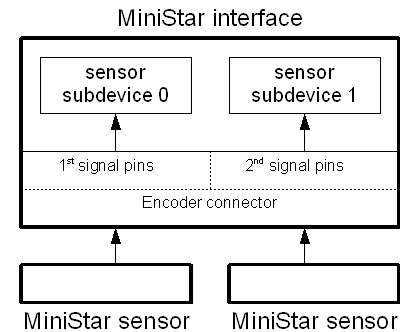
Position sensor routing

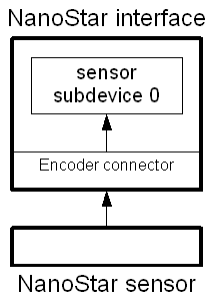
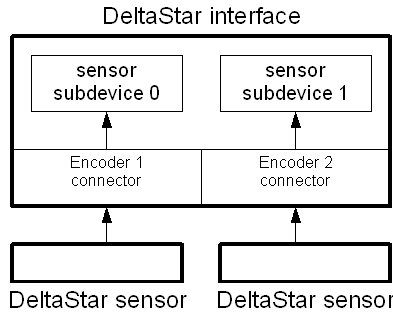
The Hydra controller features 3 Star interface ports which the Venus interpreter distinguishes by means of the Star sensor **device** index. Ports 1 and 2 provide connection of one external Star interface (i.e. any member of the Star interface family) each, using the Star interface 1 and 2 sockets. Access 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. Additionally, Port 2 supports connection of a QuickStep clock/direction interface.





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.





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.

Motor commutation handling

Determination of rotor to motor angle offset

When Hydra is used as a linear or AC motor drive, it is important that the initial motor and rotor angles and the interjacent angle offset (subsequently simply referred to as the *angle offset*) be determined prior to permanent application of motor power. Otherwise, the position control will not operate properly, and the motor action will be indeterminate.

The rotor angle is a calculated equivalent of the position determined by the translational or rotational position encoder (-> Star sensor) at any given slide or rotor location. The motor angle is a calculated equivalent of the location the slide or rotor is forced to when the motor windings are charged with a voltage pattern derived from any position calculated by the controller, as done during normal operation. Provided the instrumental parameters concerning the motor, machine and scale geometry (***Motor pole pairs*** (p. 366), ***Pitch*** (p. 281), ***Motion direction*** (p. 295), and ***Scale period*** (p. 377)) have been set correctly, the differential action of motor and rotor angle will be approximately equal, but there will usually be an offset which must be compensated for.

For instance, with the respective axis being fed with position 0 by the controller, a linear motor slide may be forced to a location resulting in encoder position 10 mm. Then, if the same axis is fed with position 5 mm, the slide will be forced to encoder position 15 mm. Obviously, the position offset is 10 mm.

For a uniform handling, Hydra relates the position offset to the motor phasewidth, thereby transforming it into an angle offset, and normalizes it to 1.0 (i.e. it will range between 0.0 and 1.0). The motor phasewidth, again, is the quotient of ***Pitch*** (p. 281) and ***Motor pole pairs*** (p. 366) .

For instance, if the rotor to motor position offset is 0.1 mm in a system with a 2.0 mm spindle and a 4 pole pairs motor,

phasewidth is $2.0 \text{ mm} / 4 = 0.5 \text{ mm}$, and the angle offset following is $0.1 \text{ mm} / 0.5 \text{ mm} = 0.2$.

Auto commutation

The angle offset can be detected utilizing the **Auto commutation** (p. 339) feature, which briefly applies a defined temporal voltage pattern to the motor windings and simultaneously evaluates motor action via the position encoder. The auto commutation procedure may result in a short humming noise.

When enabled, the auto commutation will automatically be done once during every controller powerup, and once after every **Motor restart** (p. 374). Subsequently, the result will automatically be included in the position control. With the auto commutation disabled, the result of the last auto commutation procedure will become effective instead. The angle offset is a storable parameter, so after **Configuration storage (Controller)** and **Reset** (p. 121), the controller can still reuse it.

The auto commutation can be configured via a *voltage* and a *time* parameter, which have to be determined in the system the Hydra controller is going to be used in. It will only yield correct results when parameterized properly.

Auto commutation with absolute position encoders

With absolute position encoders, the angle offset is a system constant, not dependent on the initial slide or rotor location. Therefore, it would theoretically be sufficient to auto commute once at an arbitrary position, then disable the auto commutation and store configuration.

Auto commutation with incremental position encoders

With incremental position encoders, the angle offset will obviously vary depending on the initial slide or rotor location. Therefore, it cannot simply be handled as a system constant, but must be determined anew with each powerup sequence. So angle offset storage for reuse will not do unless fixed initial conditions are provided by the user whenever a controller reset or motor restart is initiated. Therefore, auto commutation should never be disabled.

Inaccuracies and motor optimization

Due to motor build tolerances, the progress of the field generated by the permanent magnets is usually not strictly periodical over position, as presumed by the position control. Therefore, the angle offset will not be strictly be constant over the whole travel range, but vary within certain limits. Moreover, the resolution of the auto commutation result is limited. It seems recommendable to average out varying auto commutation results over position and time, thus finding an optimum value, as this will increase motor performance and direction symmetry of motor force/torque.

Optimization with absolute position encoders

With absolute position encoders, it is possible to enter the optimum angle offset manually as a Venus parameter (**Motor current shift** (p. 348)), thereby overwriting the last auto commutation result, store it afterwards for general optimization, and disable auto commutation. Regard that, in turn, the next auto commutation procedure will overwrite the parameter on its part if not disabled.

Optimization with incremental position encoders

With incremental position encoders, it is inevitable to let the controller start off motor operation using the somewhat "raw" result of the initial auto commutation. However, if there are any fixed points within the travel range such as a limit switch or reference mark, it is possible to relate the optimum angle offset to this point and regard the result as a system constant. This fixed point will be referred to as the *home location* or *home position* for motor optimization throughout this document. With Hydra, either the calibration limit switch or the reference mark can be configured to be the home location.

Once the home location has been moved towards and found by the controller, the auto commutation result can be replaced by the optimum angle offset. Hydra holds this as a Venus parameter (***Motor parameters*** (p. 357), *optimized angle offset* entry). The procedure of replacement will be referred to as *motor optimization* throughout this document. With the optimization facility being armed, the motor will automatically be optimized subsequent to every ***Calibration move*** (p. 284) or ***Reference move*** (p. 327), depending on the home location chosen. The "cal" configuration is to be used if no reference mark is available or in a system with a translational position encoder that features several reference marks, so non-ambiguous detection is impossible. Otherwise, the reference mark will usually yield the best results on this purpose.

Steps to take for initial use (in the given order):

We presume that auto commutation is parameterized correctly.

1. Switch on controller. Initial slide/rotor position has no importance.
2. Wait for powerup sequence to finish.
3. Find optimum angle offset, averaging out several auto commutation results.

4. Select motor optimization home location (must be done prior to step 5). Selection must not be changed afterwards. Keep optimization disabled.
5. Find home location by execution of corresponding move.
6. Enter optimum angle offset (will be denied if step 5 was omitted). The value entered will automatically be related to the home position, thus making a system constant.
7. Arm motor optimization.
8. Again, find home location by execution of corresponding move. Optimization is done automatically.
9. If the result is satisfactory, store parameters.
10. Initiate **Reset** (p. 121) .

Now motor optimization will be done upon completion of every calibration move or every reference move, depending on the configuration.

Introduction to manual operation

Manual drivers and manual devices

As an option, Hydra can be driven manually by a number of Controller Area Network devices (*manual drivers*) at the CAN connector. *Manual drivers* supported:

- 2-channel joystick
- 3-channel joystick
- 2-channel handwheel

The manual network is designed to take up to 2 joysticks and up to 2 handwheels. At the time, it supports up to 1 joystick and up to 1 handwheel.

This chapter deals with manual position generation exclusively. For CAN device pushbutton operation see introductory chapter "Introduction to digital I/O processing".

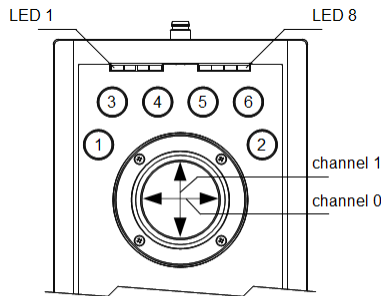
Each Hydra axis device holds 4 *manual devices* with a ***Manual device parameters*** set each. A *manual device* is a virtual slot any channel of any connected *manual driver* can virtually be plugged into in order to drive the respective axis. In doing so, multiple use is possible, so each driver channel can be used to drive one axis only or both axes simultaneously.

A manual driver channel is assigned to a manual device via the *input driver* and *input channel* entries of the respective ***Manual device parameters*** set. The following driver and channel indexes are available:

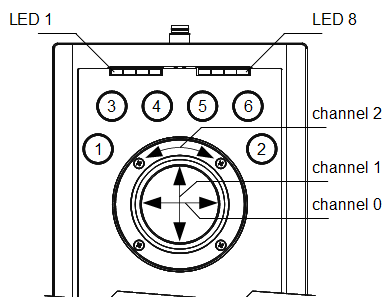
manual driver	driver index*	number of channels (m)	channel index
1 st CAN joystick	0	2	0...1
		3	0...2
1 st CAN handwheel	1	2	0...1
2 nd CAN joystick	2	2	0...1
		3	0...2
2 nd CAN handwheel	3	2	0...1

Manual driver properties

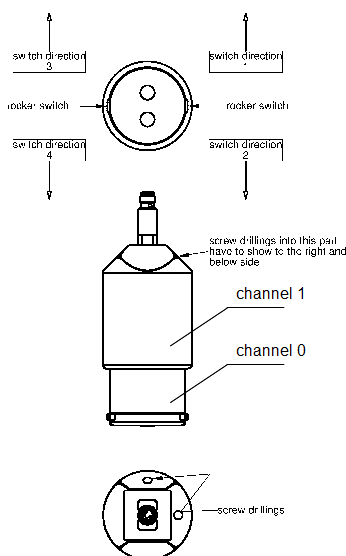
*2-channel and 3-channel CAN joysticks share the same driver indexes. The number of available channels is auto-detected by firmware.



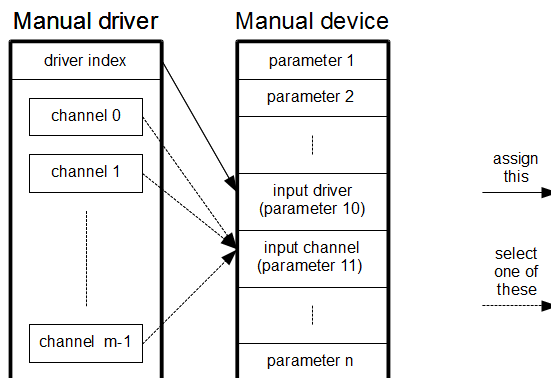
2-channel joystick



3-channel joystick



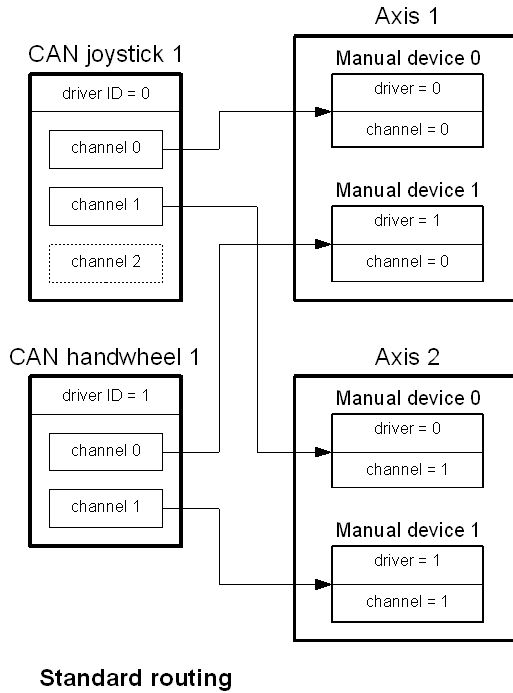
2-channel handwheel



Manual driver channel assignment

By default,

- index 0 channels of joystick 1 and handwheel 1 drive axis 1 via *manual devices* 0 and 1
- index 1 channels of joystick 1 and handwheel 1 drive axis 2 via *manual devices* 0 and 1



Assignment example

Preconditions: The Hydra controller is equipped with one 2-channel CAN joystick.

Task: Assign joystick channel 0 to Axis 2 and joystick channel 1 to Axis 1.

Preparation: We use the first free slot of each axis which is **Manual device 0**. The *input driver* and *input channel* indexes in **Manual device parameters 0** (p. 267) are 10 and 11, respectively.

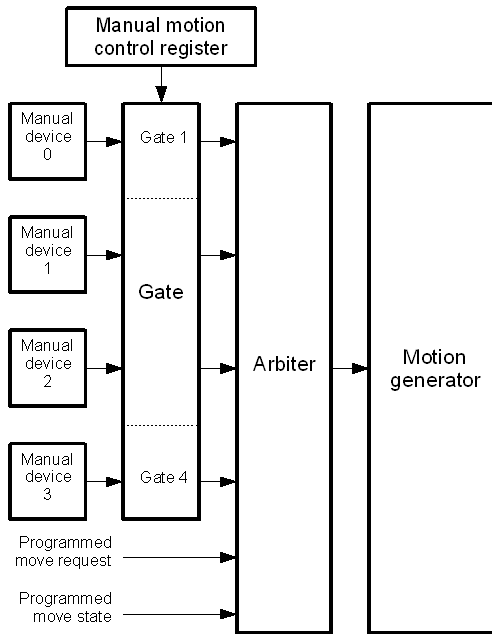
Note: Unlike in the table below, all commands have to be entered *in one line*.

	Command	Description
1:	0 10 1 1 setmanpara <small>(p. 265)</small>	Assign manual driver <i>joystick 1</i> to Manual device 0 of Axis 1.
2:	0 10 1 2 setmanpara	Assign manual driver <i>joystick 1</i> to Manual device 0 of Axis 2.
3:	1 11 1 1 setmanpara	Select channel 1 of manual driver assigned above (<i>joystick 1</i>) to feed Manual device 0 of Axis 1.
4:	0 11 1 2 setmanpara	Select channel 0 of manual driver assigned above (<i>joystick 1</i>) to feed Manual device 0 of Axis 2.

Motion control arbitration

The *manual devices* of each axis can individually be enabled or disabled via a bit matrix (s. **Manual motion control** (p. 275) register); several *manual devices* may simultaneously be enabled at one axis. An arbiter inhibits conflicting accesses to axis motion.

Note that pushbutton function will be disabled along with the joystick or handwheel function if disabled via the bit matrix. Routing and enable state have no impact on LED function, though. Another facility of disabling motion data generation is the *mode* entry of the corresponding **Manual device parameters** set which does not affect pushbutton functionality.



Axis motion control arbitration

Motion control arbitration rules:

- Any initiation of a programmed move will stop any manual move at any time.
- As long as a programmed move is running, all manual input will be discarded.
- Manual motion control can selectively be configured to be enabled or disabled after any programmed move termination.
- As long as no programmed move is running,
 - The first manual device producing valid motion data (data that cause the respective axis to be displaced from its current location) gains motion

control. It remains motion master as long as it continues to produce valid motion data.

- If a manual device requests motion control by producing valid motion data while another device is still motion master, action will be discarded until the current motion master on its part stops producing valid motion data. Subsequently, motion control is handed over to the requesting device.

Complete examples

Complete parameterization example for joystick

Preconditions: The Hydra controller is equipped with one 2-channel CAN joystick.

Task: Configure channel 1 of joystick to drive axis 1 at 20 mm/s max. and 200 mm/s², using cubic progression and standard motion direction. Configure channel 0 of joystick to drive axis 2 at 10 mm/s max. and 500 mm/s², using square progression and reverse motion direction. Elongation threshold is to be set to 0.1 at both axes. Joystick position generation is to be always active at both axes.

Preparation: We use the first free slot of each axis which is **Manual device 0**.

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

	Command	Description
1:	20 1 1 1 setmanpara (p. 265)	Set full elongation velocity at Manual device 0 of Axis 1 to 20 mm/s (<i>velocity entry</i>).
2:	200 2 1 1 setmanpara	Set acceleration at Manual device 0 of Axis 1 to 200 mm/s ² (<i>acceleration entry</i>).
3:	2 6 1 1 setmanpara	Set elongation-to-velocity transfer function at Manual device 0 of Axis 1 to cubic progression (<i>input function entry</i>).
4:	2 7 1 1 setmanpara	Set position data generation at Manual device 0 of Axis 1 to be enabled throughout operation (<i>mode entry</i>).
5:	0.1 8 1 1 setmanpara	Set position data generation threshold at Manual device 0 of Axis 1 to 0.1 (<i>threshold entry</i>).
6:	0 9 1 1 setmanpara	Set effective direction at Manual device 0 of Axis 1 to standard direction (<i>direction entry</i>).
7:	0 10 1 1 setmanpara	Assign manual driver <i>joystick 1</i> to Manual device 0 of Axis 1 (<i>input driver entry</i>).
8:	1 11 1 1 setmanpara	Select channel 1 of manual driver assigned above (<i>joystick 1</i>) to feed Manual device 0 of Axis 1 (<i>input channel entry</i>).
9:	10 1 1 2 setmanpara	Set full elongation velocity at Manual device 0 of Axis 2 to 10 mm/s (<i>velocity entry</i>).
10:	500 2 1 2 setmanpara	Set acceleration at Manual device 0 of Axis 2 to 500 mm/s ² (<i>acceleration entry</i>).
11:	1 6 1 2 setmanpara	Set elongation-to-velocity transfer function at Manual device 0 of Axis 2 to square progression (<i>input function entry</i>).

	Command	Description
12:	2 7 1 2 setmanpara	Set position data generation at Manual device 0 of Axis 2 to be enabled throughout operation (<i>mode</i> entry).
13:	0.1 8 1 2 setmanpara	Set position data generation threshold at Manual device 0 of Axis 2 to 0.1 (<i>threshold</i> entry).
14:	1 9 1 2 setmanpara	Set effective direction at Manual device 0 of Axis 2 to reverse direction (<i>direction</i> entry).
15:	0 10 1 2 setmanpara	Assign manual driver <i>joystick 1</i> to Manual device 0 of Axis 2 (<i>input driver</i> entry).
16:	0 11 1 2 setmanpara	Select channel 0 of manual driver assigned above(<i>joystick 1</i>) to feed Manual device 0 of Axis 2 (<i>input channel</i> entry).
17:	1 1 setmanctrl (p. 276)	Enable Manual device 0 of Axis 1.
18:	1 2 setmanctrl	Enable Manual device 0 of Axis 2.

Task: Keep above routing. Disable position data generation at both axes, but keep pushbutton function.

	Command	Description
1:	0 7 1 1 setmanpara	Set position data generation at Manual device 0 of Axis 1 to be disabled throughout operation.
2:	0 7 1 2 setmanpara	Set position data generation at Manual device 0 of Axis 2 to be disabled throughout operation.

Task: Keep above routing. Disable position data generation at both axes and pushbutton function.

	Command	Description
1:	0 1 <i>setmanctrl</i>	Disable Manual device 0 of Axis 1.
2:	0 2 <i>setmanctrl</i>	Disable Manual device 0 of Axis 2.

Complete parameterization example for handwheel

Preconditions: The Hydra controller is equipped with one 2-channel joystick and one 2-channel CAN handwheel.

Task: Expand preceeding example as follows: configure both wheels of additional handwheel unit to drive axis 1, with the front wheel operating as a fine drive at 1 mm/rev and the rear wheel operating as a coarse drive at 10 mm/rev. Both axes are to run at 50 mm/s max. and 500 mm/s². Handwheel position generation is to be always active.

Preparation: We use the first 2 free slots at axis 1. As **Manual device 0** is already occupied by a channel of joystick 1, we use **Manual device 1** and **Manual device 2**.

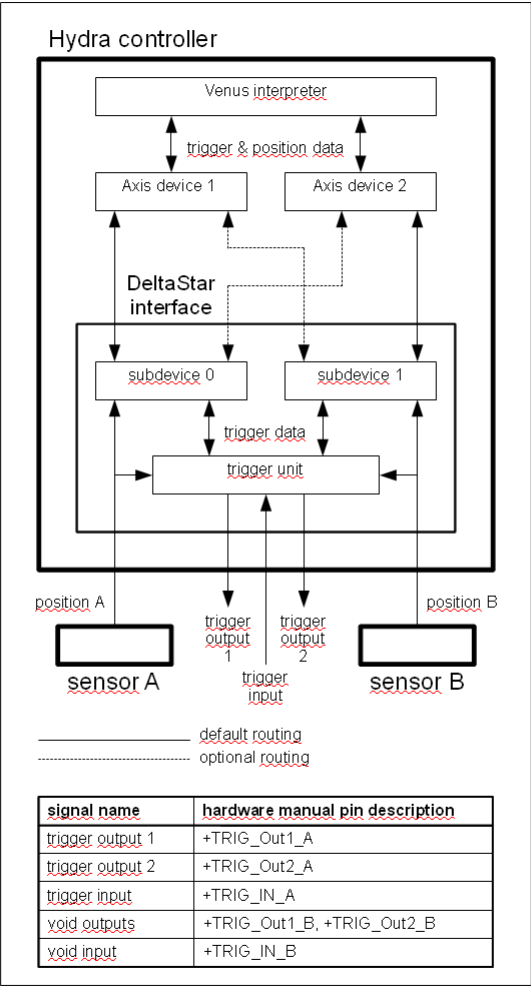
Note: Unlike in the table below, all commands have to be entered *in one line*.

	Command	Description
1:	50 1 2 1 <i>setmanpara</i> (p. 265)	Set full elongation velocity at Manual device 1 of Axis 1 to 50 mm/s (<i>velocity</i> entry).
2:	500 2 2 1 <i>setmanpara</i>	Set acceleration at Manual device 1 of Axis 1 to 500 mm/s ² (<i>acceleration</i> entry).
3:	323584 4 2 1 <i>setmanpara</i>	Set wheel encoder resolution at Manual device 1 of Axis 2 to 323584 inc./

	Command	Description
		rev., which is appropriate for SMC CAN handwheel (<i>resolution</i> entry).
4:	1 5 2 1 setmanpara	Set wheel transformation at Manual device 1 of Axis 1 to 1 mm/rev. (<i>ratio</i> entry).
5:	2 7 2 1 setmanpara	Set position data generation at Manual device 1 of Axis 1 to be enabled throughout operation (<i>mode</i> entry).
6:	1 10 2 1 setmanpara	Assign manual driver <i>handwheel 1</i> to Manual device 1 of Axis 1.
7:	0 11 2 1 setmanpara	Select channel 0 of manual driver assigned above (<i>handwheel 1</i>) to feed Manual device 1 of Axis 1.
8:	50 1 3 1 setmanpara	Set full elongation velocity at Manual device 2 of Axis 1 to 50 mm/s (<i>velocity</i> entry).
9:	500 2 3 1 setmanpara	Set acceleration at Manual device 2 of Axis 1 to 500 mm/s ² (<i>acceleration</i> entry).
10:	323584 4 3 1 setmanpara	Set wheel encoder resolution at Manual device 2 of Axis 2 to 323584 inc./rev., which is appropriate for SMC CAN handwheel (<i>resolution</i> entry).
11:	10 5 3 1 setmanpara	Set wheel transformation at Manual device 2 of Axis 1 to 10 mm/rev. (<i>ratio</i> entry).
12:	2 7 3 1 setmanpara	Set position data generation at Manual device 2 of Axis 1 to be enabled throughout operation (<i>mode</i> entry).
13:	1 10 3 1 setmanpara	Assign manual driver <i>handwheel 1</i> to Manual device 2 of Axis 1.
14:	1 11 3 1 setmanpara	Select channel 1 of manual driver assigned above (<i>handwheel 1</i>) to feed Manual device 2 of Axis 1.

	Command	Description
15:	7 1 <i>setmanctrl</i> <small>(p. 276)</small>	Enable <i>Manual device 0</i> , <i>Manual device 1</i> , and <i>Manual device 2</i> of Axis 1.
16:	1 2 <i>setmanctrl</i>	Enable <i>Manual device 0</i> of Axis 2.

Introduction to trigger function



trigger overview

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.

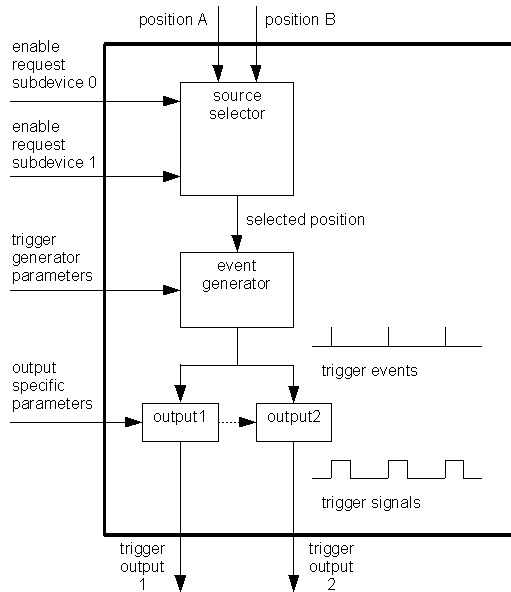
Each DeltaStar interface features one trigger unit which provides *one* trigger output channel and *one* triggered position capture channel. Both channels can independently be enabled under individual specification of the target axis device (and thus, via **Sensor assignment** (p. 398), the target DeltaStar subdevice. The overview on the preceeding page depicts the standard and optional routings. If needed, the routing can be changed by **Sensor assignment**).

The DeltaStar offers one trigger signal input (**trigger input**) for position capture and two trigger signal outputs (**trigger output 1**, **trigger output 2**). One more input and two more outputs are physically existent, but void.

1. Output trigger function

The Hydra output trigger concept distinguishes generation of trigger events from the generation of the actual signal output as shown above. The trigger output function can be enabled for *one subdevice at a time*.

The **source selector** determines the trigger position source. Whenever subdevice 0 is being enabled for trigger output, position A will be significant afterwards. Whenever subdevice 1 is being enabled for trigger output, position B will be significant afterwards. See **Trigger mode** (p. 460) and **Trigger event setup** (p. 458) on how to enable the trigger function at a specified subdevice.



trigger output channel

The **trigger event generator** puts out a single trigger event whenever certain conditions (mostly concerning the currently measured slide or rotor position) are met. The *trigger generator parameters* **Trigger mode** and **Trigger event setup** provide 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 *output specific parameters* such as **Trigger output polarity** (p. 469), **Trigger output pulse width** (p. 471), and **Trigger output delay** (p. 467). The overall result will be the output of several synchronized trigger output signals. In addition, **trigger output 2** has a special function if put into direction mode via **Trigger mode**.



Note that between consecutive trigger events, a gap of 2.5 μs *min.* must be maintained.



Note that for compatibility purpose,

- the Venus interpreter holds one complete output trigger parameter set for each Axis device
- trigger parameter transmission does not depend on trigger activation, but all parameters are forwarded to the DeltaStar interface immediately

The DeltaStar interface, however, holds only one trigger parameter set. This means all parameters entered will immediately take effect regardless to the given Axis / subdevice selection. There is no implicate check as to consistency of momentary trigger data with any command. The proper way to use the trigger output feature is to

- first, transmit all output specific parameters for the target subdevice
- second, set the trigger mode for the target subdevice (effective only if setting differs from current one; trigger generator will then be enabled implicitly)
- third, if required, transmit trigger generator parameters for the target subdevice; trigger generator will be enabled implicitly
- fourth, initiate move at target axis

With each change of the target subdevice,

- **a currently running trigger sequence must have finished before the first new parameter is set**
- **all trigger parameters have to be set anew before trigger activation**



Note that whatever configuration options the Hydra firmware may offer, support is depending on the respective DeltaStar logic/firmware revision.

Output trigger examples

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** (p. 398)), 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 μ s wide trigger pulses, starting at 10 mm, stopping at 90 mm, delayed by 8 μ s; 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. 471)	Set trigger pulse width at output 1 at axis 1 to 5 μ s.
2:	8 1 1 settroutdelay (p. 467)	Set trigger delay at output 1 at axis 1 to 8 μ s.
3:	0 1 1 settroutpol (p. 470)	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. 465)	Prepare axis 1 to generate equidistant trigger pulses at output 1 and the direction signal at output 2.
6:	10 90 9 1 settrpara (p. 458)	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), and arm trigger.
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** (p. 398)).

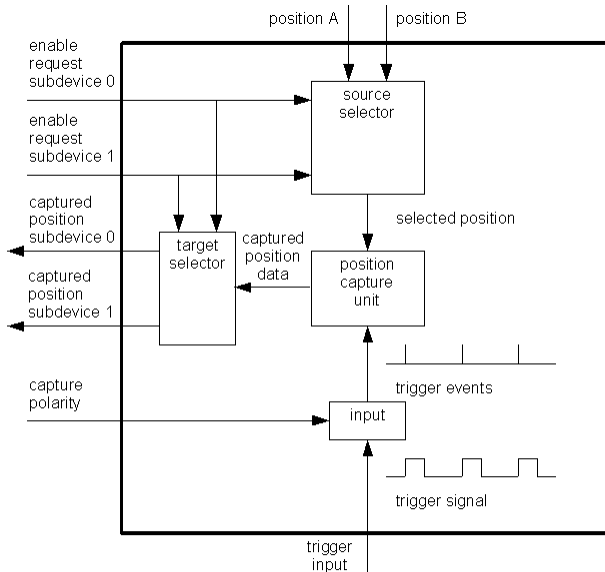
Task: Put out a continuous 500 Hz signal at both trigger outputs, 3 μ s pulsewidth at both outputs, active high at output 1, inverted and delayed by 7 μ s at output 2.

	Command	Description
1:	3 1 1 settroutpw (p. 471)	Set trigger pulse width at output 1 at axis 1 to 3 μ s.
2:	3 2 1 settroutpw	Set trigger pulse width at output 2 at axis 1 to 3 μ s.
3:	7 2 1 settroutdelay (p. 467)	Set trigger delay at output 2 at axis 1 to 7 μ s.
4:	0 1 1 settroutpol (p. 470)	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 (p. 465)	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 position capture function can be enabled for *one subdevice at a time*. An external trigger signal has to be provided at the trigger input.



triggered position capture channel

The **source** and **target selectors** determine the trigger position source and captured position target. Whenever subdevice 0 is being enabled for position capture, position A will be significant afterwards. Captured position data will then be forwarded to subdevice 0. Whenever subdevice 1 is being enabled for position capture, position B will be significant afterwards. Captured position data will then be forwarded to subdevice 1. See **Trigger capture mode**

(p. 448) on how to enable the trigger function at a specified subdevice.

With the position capture function armed at either subdevice, the **position capture unit** watches the trigger signal **input**. Whenever a trigger event (for definition of input trigger events see **Trigger capture polarity** (p. 450)) occurs at the input, the current position value at the armed 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. 452) .

Triggered position capture example

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** (p. 398)). Current position at axis 1 is 0. Velocity and acceleration settings are 250 mm/s and 1000 mm/s², respectively. External active low trigger signal is provided at the trigger input. 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 settrinp <small>ol</small> (p. 451)	Match trigger input to the active low trigger source by making it sensitive to falling edges.
2:	10 1 settrin <small>size</small> (p. 445)	Prepare capture buffer to record up to 10 position values.
3:	1 1 settrin (p. 448)	Arm trigger capture function.
4:	100 1 nm (p. 314)	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 gettrin <small>index</small> (p. 447)	Query number of captured position values. Should be 7 at least.

Query captured positions. We assume perfect action.

	Command	Description
1:	0 1 gettrin <small>pos</small> (p. 452)	Reply: 0.000000
2:	1 1 gettrin <small>pos</small>	Reply: 6.250000
3:	2 1 gettrin <small>pos</small>	Reply: 25.000000
4:	3 1 gettrin <small>pos</small>	Reply: 50.000000

	Command	Description
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 <small>(p. 455)</small>	Store text file of the above record.

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

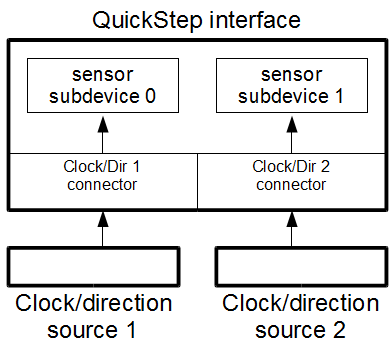
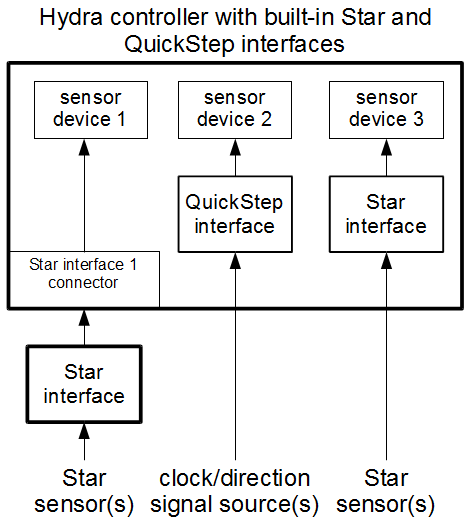
To store the record in a text file on the host PC, first open command line box and change to the target directory. Then type in (*one line*):

```
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.

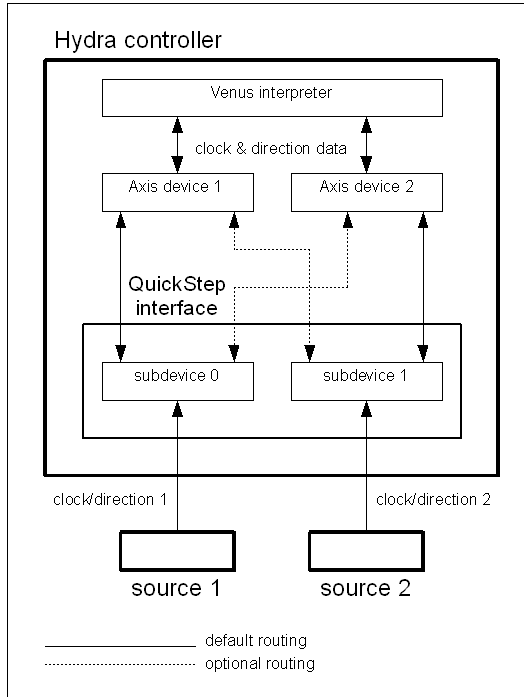
Introduction to clock and direction operation

Clock and direction controlled motion function



If your Hydra controller is equipped with a QuickStep clock/direction interface, axis motion can be controlled by external clock/direction signal sources. QuickStep utilizes the Star interface for data transmission and is therefore compliant with the Star sensor interface family. As opposed to the sensor interfaces, however, QuickStep must always be connected to Star interface port 2, as this the only

port to support clock/direction function. Each QuickStep interface supports up to 2 clock/direction channels, called *sensor subdevices* in the style of the Star sensor interface family. Each source is connected to either of the Clock/Dir input connectors which are invariably linked to the *sensor subdevices* as shown above.



clock/direction overview

Each of the *sensor subdevices* is assigned to either of both axes via **Sensor assignment** (p. 398). The overview above depicts the standard and optional routings. If need be, the signal sources can be rerouted just as position sensors, with the sole precondition that the *sensor device* index is always 2. The clock/direction function can be accessed under specification of the target axis device (and thus, via

the routing, the target QuickStep subdevice). See ***Clock and direction function*** (p. 316) and ***Clock and direction width*** (p. 318) on parameterization.

Example

Preconditions: The Hydra controller is equipped with a QuickStep clock/direction interface (at Star interface port 2).

Task: Clock/direction source 1 is to drive axis 2. Way to go per incoming clock pulse is 1 μm .

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

	Command	Description
1:	0 2 setcdfunc <small>(p. 316)</small>	Disable clock/direction control function at axis 2.
2:	0 2 2 setassignment <small>(p. 399)</small>	Destine "sensor subdevice" 0 (clock/direction source 1), "sensor device" 2 (Star interface port 2), to drive axis 2.
3:	1000 2 setcdwidth <small>(p. 319)</small>	Set motion distance per incoming clock pulse at axis 2 to 1 μm .
4:	2 2 setcdfunc	Enable clock/direction control function at axis 2. Position alignment is implicitly done prior to activation.

Result: Clock/direction source 1 will drive axis 2 at 1 μm per clock pulse until function is being disabled.

Introduction to digital I/O processing

I/O organization

All Venus accessible digital I/O targets are organized into two sets of I/O groups, one being assigned to the Controller device and the other one being assigned to the Axis devices, and featuring separate sets of parameters and commands. An I/O group consists of a specified number of *digital outputs* and a specified number of *event sources*.

A *digital output* can be incorporated by

- a digital output pin *or*
- an LED at an external CAN device

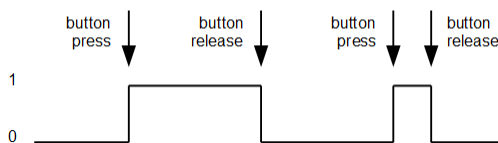
An *event source* can consist in

- a digital input pin *or*
- a pushbutton or key at an external CAN device *or*
- an internal condition

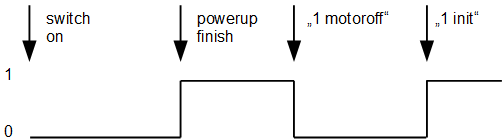
In this context, a pushbutton can be imagined as a digital input pin that goes high whenever the button is pressed and vice versa. An internal condition can be imagined as a digital input that goes high whenever the respective condition is true. The current status of the *event source* is referred to as the *event state*.

Examples:

1. pushbutton event state:



2. „axis ready“ condition event state:



All I/O targets work in the same way, regardless of group membership or assignment.

Controller device I/O

With the Controller I/O group set, a single group is not tied to any axis, but represents one self-contained device featuring I/O functionality, such as the Hydra controller, a connected CAN joystick etc. Each I/O group can be addressed by its individual *group index*. If applicable, a *source index* allows for additional access to a single event source (= pushbutton or digital input pin) or digital output.

Available groups at the time:

group index	destination	event sources	digital outputs
1	controller	6 x digital input pin	3 x digital output pin
2	CAN joystick 1	6 x pushbutton	8 x LED
3	CAN handwheel 1	4 x pushbutton	1 x LED
4..5	reserved	-	-
6	timer group	8 x timer	-

Axis device I/O

The axis device I/O group set does not handle any physically available input or output, but some definite axis dependent *internal conditions*, with one group assigned to each individual axis. These conditions are processed in the same manner as digital inputs, and therefore being qualified as I/O targets as well. Each group is addressed by the *group index*, which equals the device index of the axis it belongs to. If applicable, a *source index* allows for additional access to a single event source (= internal condition).

Available conditions per group/axis at the time:

event source index	condition																														
1	<i>axis powered up</i> true once the axis has passed the powerup sequence																														
2	<i>axis ready</i> true whenever the Axis status (p. 417) register bit 10 value is 0																														
3	<i>axis resting on target</i> truth table: <table><tr><th>Axis status bit 0</th><th>Axis status bit 5</th><th>Target window (p. 412)</th><th>Time on target (p. 414)</th><th>condition value</th></tr><tr><td>1</td><td>X</td><td>X</td><td>X</td><td>0</td></tr><tr><td>0</td><td>0</td><td>> 0</td><td>> 0</td><td>0</td></tr><tr><td>0</td><td>X</td><td>X</td><td>= 0</td><td>1</td></tr><tr><td>0</td><td>X</td><td>= 0</td><td>X</td><td>1</td></tr><tr><td>0</td><td>1</td><td>> 0</td><td>> 0</td><td>1</td></tr></table>	Axis status bit 0	Axis status bit 5	Target window (p. 412)	Time on target (p. 414)	condition value	1	X	X	X	0	0	0	> 0	> 0	0	0	X	X	= 0	1	0	X	= 0	X	1	0	1	> 0	> 0	1
Axis status bit 0	Axis status bit 5	Target window (p. 412)	Time on target (p. 414)	condition value																											
1	X	X	X	0																											
0	0	> 0	> 0	0																											
0	X	X	= 0	1																											
0	X	= 0	X	1																											
0	1	> 0	> 0	1																											

event source index	condition
4	<i>reserved</i>
5	<i>processing Calibration move</i> (p. 284)
6	<i>processing Range measure move</i> (p. 306)
7	<i>reference mark found</i>

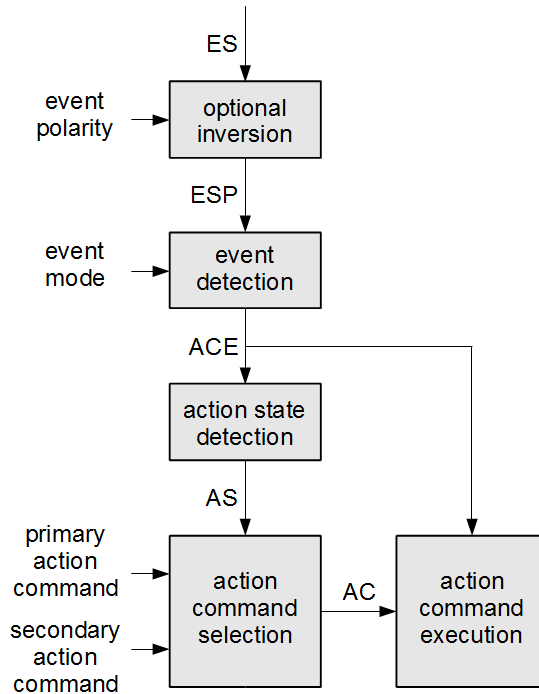
Action command function

The action command feature allows for automatic execution of user definable Venus command strings (*action commands*, *AC*) upon the occurrence of certain events (*action command events*, *ACE*). For this purpose, each I/O group features two sets of executable Venus command strings, labeled "primary" and "secondary", respectively, each holding one command string entry for each *event source*. These strings can be defined by the user like any other storable Venus parameter.

There are several user options:

- inversion (e.g. button press vs. button release) => *polarity* parameter
- event selection (e.g. switch vs. key function) => *mode* parameter
- initial behaviour => *initial action mask* parameter (will be addressed in detail later)
- alternation between the two different command strings (toggle function)

The following block chart overviews action command operation.



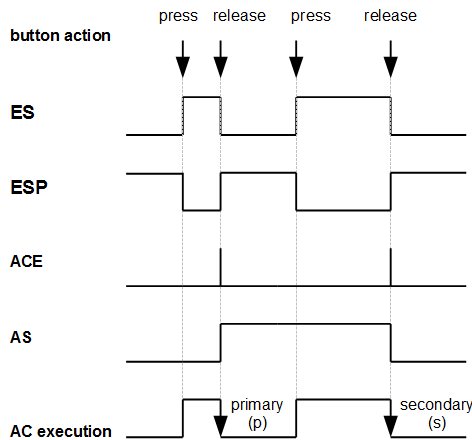
Action command processing flow chart

An *ACE* occurs whenever an *event source* changes state and meets certain additional conditions. In order to determine which changes will actually make *ACE*, the *event state* (*ES*) is first passed through an optional inversion (done if *polarity* is 1), the result being *ESP*. If *mode* is 1, every *ESP* change will make an *ACE*; otherwise, negative (1 to 0) level transitions will be discarded.

In order to select the command to be executed (primary vs. secondary), the *action state* (*AS*) is formed out of *ACE*. It changes state whenever an *ACE* occurs. That means with each *ACE*, the primary (*AS* = 0) and secondary (*AS* = 1) command strings for the

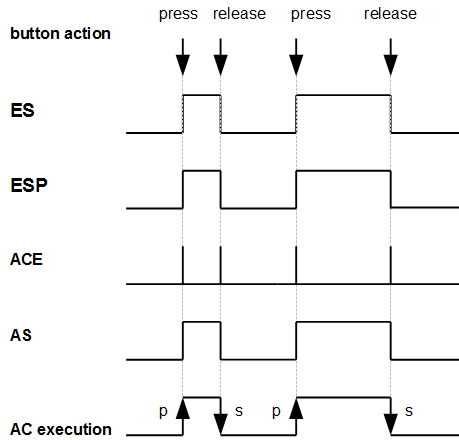
corresponding *event source* will be executed alternately (command execution "toggles"). If empty, however, the secondary command string will be replaced by the primary command string (i.e. "toggle" function is off).

The following signal chart shows an example with a pushbutton event source, mode being 0 (switch function) and polarity being 1 (i.e. command events follow pushbutton releases). The *AC execution* chart simultaneously shows the event state, the resulting actions (p / s = execution of primary / secondary command), and the signal transitions (down / up) upon which they occur.



Action command example (mode = 0, polarity = 1)

A similar example, but mode being 1 (key function, command events follow each pushbutton press *and* release) and polarity being 0 (i.e. press => primary, release => secondary command execution):

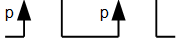
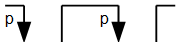
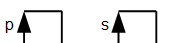
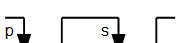
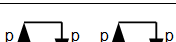
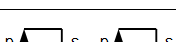
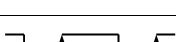


Action command example (mode = 1, polarity = 0)







Note that event occurrence to command execution delay can amount to 25 ms max.

The following chart summarizes the *AC execution* charts of all parameter constellations available. You will find the examples above correspond to patterns *2b* and *4a*, respectively.

event mode	secondary action command	event polarity	action command execution chart *	pattern
0	no	0		1a
0	no	1		1b
0	yes	0		2a
0	yes	1		2b
1	no	any		3
1	yes	0		4a
1	yes	1		4b

*caption:

-  primary action command executed on positive level transition / pushbutton press
-  primary action command executed on negative level transition / pushbutton release
-  secondary action command executed on positive level transition / pushbutton press
-  secondary action command executed on negative level transition / pushbutton release

Action command configuration

Venus I/O access

Brief I/O command overview.

Direct I/O access

Direct I/O commands focus on a single event source or output target for reading or writing, using the group and target indexes. See ***Event state*** (p. 186) and ***Internal event state*** (p. 215) for state query of a single event source. See ***Digital output state*** (p. 168) for setting the level of a single output.

I/O register access

All I/O access commands apart from the above work on register basis, i.e. each I/O group holds one register for the item in question. Each single event source of the respective group is represented by one single bit in that register, the bit order following the target enumeration.

For instance, the ***Event polarity register - CAN joystick 1*** (p. 184) holds one polarity bit for each joystick pushbutton, bit 0 representing pushbutton 1.

I/O registers per group and where to find them in particular:

parameter	controller device I/O	axis device I/O
event mode	<i>Event mode entry</i> (p. 174)	<i>Internal event mode register</i> (p. 209)
event polarity	<i>Event polarity entry</i> (p. 180)	<i>Internal event polarity register</i> (p. 212)
initial action mask*	<i>Initial action mask entry</i> (p. 188)	<i>Internal initial action mask register</i> (p. 217)
event detect	<i>Event detect register</i> (p. 172)	<i>Internal event detect register</i> (p. 207)
action state	<i>Action state register</i> (p. 165)	<i>Internal action state register</i> (p. 203)

*covered in section "Initial event generation"

Setting of action commands

Like the direct I/O commands, the setting of action command strings focuses on a single event source, additionally using a third index specifying whether the primary (0) or secondary (1) action command string is to be set. See ***Action command entry*** (p. 159) and ***Internal action command entry*** (p. 198) for detailed description.

Action command examples

Example 1

The Venus parameterization will now be explained by means of some practical applications, also showing I/O access.

Preconditions: The Hydra controller is equipped with a CAN joystick (manual driver 0).

Task: Display statuses of all limit switches by joystick LEDs 1 through 4 (on => engaged, off => disengaged). All other controller inputs are presumed to be ineffective as to action commands.

Preparation:

At first, we determine the command strings for LED switching. Joystick LEDs are accessed using group index 2, output indexes 1 through 4 (LED 1...4). Command strings are as follows:

LED access:

output group	group index	output	output index	command string (x = LED status [0 1])
CAN joystick 1	2	LED 1	1	„x 1 2 setdoutst“
		LED 2	2	„x 2 2 setdoutst“
		LED 3	3	„x 3 2 setdoutst“
		LED 4	4	„x 4 2 setdoutst“

Then we assign the LED access command strings to the limit switch inputs in a comprehensible order. In order to illuminate each LED at the time of corresponding limit switch engagement (and vice versa), we select "switch

on" for primary and "switch off" for secondary action command, and choose active high polarity (the same result could as well be achieved by swapping primary and secondary commands and choosing active low polarity).

Action command assignment:

input group	group index (g)	input	input index (i)	primary command string (p)	secondary command string (s)
controller	1	Axis 1 cal sw.	5	„1 1 2 setdoutst“ (LED 1 on)	„0 1 2 setdoutst“ (LED 1 off)
		Axis 1 rm sw.	6	„1 2 2 setdoutst“ (LED 2 on)	„0 2 2 setdoutst“ (LED 2 off)
		Axis 2 cal sw.	2	„1 3 2 setdoutst“ (LED 3 on)	„0 3 2 setdoutst“ (LED 3 off)
		Axis 2 rm sw.	3	„1 4 2 setdoutst“ (LED 4 on)	„0 4 2 setdoutst“ (LED 4 off)

Now we set the command strings via Venus. With reference to the symbol letters *g*, *i*, *p*, *s* from the table header, syntax is as follows:

1. Primary: *p 0 i g setactcmd*
2. Secondary: *s 1 i g setactcmd*

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

	Command	Description
1:	"1 1 2 setdoutst" 0 5 1 setactcmd (p. 160)	Assign "LED 1 on" to axis 1 cal switch engagement.
2:	"0 1 2 setdoutst" 1 5 1 setactcmd	Assign "LED 1 off" to axis 1 cal switch release.
3:	"1 2 2 setdoutst" 0 6 1 setactcmd	Assign "LED 2 on" to axis 1 rm switch engagement.
4:	"0 2 2 setdoutst" 1 6 1 setactcmd	Assign "LED 2 off" to axis 1 rm switch release.

	Command	Description
5:	"1 3 2 setdoutst" 0 2 1 setactcmd	Assign "LED 3 on" to axis 2 cal switch engagement.
6:	"0 3 2 setdoutst" 1 2 1 setactcmd	Assign "LED 3 off" to axis 2 cal switch release.
7:	"1 4 2 setdoutst" 0 3 1 setactcmd	Assign "LED 4 on" to axis 2 rm switch engagement.
8:	"0 4 2 setdoutst" 1 3 1 setactcmd	Assign "LED 4 off" to axis 2 rm switch release.

At last, we set the *polarity* and *mode* register values. As said above, polarity is to be set to active high (0) at all limit switch inputs. In order to provide LED changing with every change of switch state, *mode* is to be set to 1 at all limit switch inputs.

Mode and polarity register calculation:

input group	group index	input	input index	bit value	polarity	mode
controller	1	general purpose 1	1	1	any (0)	any (0)
		axis 2 cal switch	2	2	0	1
		axis 2 rm switch	3	4	0	1
		general purpose 2	4	8	any (0)	any (0)
		axis 1 cal switch	5	16	0	1
		axis 1 rm switch	6	32	0	1

polarity register value = 0 (all bits 0)

mode register value = $32 + 16 + 4 + 2 = 54$

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

	Command	Description
1:	0 1 setevtpol _(p. 181)	Set event sensitivity to active high at all controller inputs.
2:	54 1 setevtmode _(p. 175)	Set event mode to key function at all limit switch inputs.

The overall configuration would match AC execution pattern *4a* (s. above).

Example 2

Preconditions: The Hydra controller is equipped with a CAN joystick (manual driver 0).

Task: On every pressure of joystick 1 pushbutton 1, the machine is to move to cartesian coordinates (0 / 0) and (10 mm / 20 mm) alternately. On every pressure of joystick 1 pushbutton 2, the machine is to perform a vectorial relative move by (1 mm / 1 mm) cartesian distance. All other joystick 1 pushbuttons are presumed to be ineffective as to action commands.

Preparation:

At first, we determine the command strings for moving. This is obviously "0 0 m", "10 20 m", and "1 1 r". Then we assign the command strings to the joystick pushbuttons 1 and 2.

Action command assignment:

input group	group index (g)	push button	push button index (i)	primary command string (p)	secondary command string (s)
joystick 1	2	pushbutton 1	1	„0 0 m“	„10 20 m“
		pushbutton 2	2	„1 1 r“	-

Now we set the command strings via Venus. With reference to the symbol letters *g*, *i*, *p*, *s* from the table header, syntax is as follows:

1. Primary: *p 0 i g setactcmd*
2. Secondary: *s 1 i g setactcmd*

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

	Command	Description
1:	"0 0 m" 0 1 2 setactcmd (p. 160)	Assign move to origin to primary pressure of joystick pushbutton 1.
2:	"10 20 m" 1 1 2 setactcmd	Assign move to (10 mm / 20 mm) to secondary pressure of joystick pushbutton 1.
3:	"1 1 r" 0 2 2 setactcmd	Assign relative move by (1 mm / 1 mm) to each pressure of joystick pushbutton 2.

At last, we set the *polarity* and *mode* register values. In order to provide for command execution on pushbutton pressure (discarding pushbutton release), polarity is to be set to active high (0) and *mode* is to be set to 0 at both event sources.

polarity register value = 0 (all bits 0)

mode register value = 0 (all bits 0)

	Command	Description
1:	0 2 <i>setevtpol</i> _(p. 181)	Set event sensitivity to active high at all joystick pushbuttons.
2:	0 2 <i>setevtmode</i> _(p. 175)	Set event mode to key function at all joystick pushbuttons.

The overall configuration of pushbutton 1 would match AC execution pattern 2a; the overall configuration of pushbutton 2 would match AC execution pattern 1a (s. above).

Example 3

Task: Display "axis ready" status via TTL outputs 1 and 2; polarity be active low, i.e. axis is ready if respective output is low. All other internal conditions are presumed to be ineffective as to action commands.

Preparation:

At first, we determine the command strings for output control. Controller outputs are accessed using group index 1, output indexes 4 and 5 (TTL 1/2). Command strings are as follows:

TTL output access:

output group	group index	output	output index	command string (x = output status [0/1])
controller	1	TTL 1	4	„x 4 1 setdoutst“
		TTL 2	5	„x 5 1 setdoutst“

Then we assign the move command strings to the respective internal conditions. We select "set" for primary and "clear" for secondary action command, and choose active low polarity (the same result could also be achieved by swapping primary and secondary commands and choosing active high polarity).

Action command assignment:

event group	group index (g)	int. cond.	source index (i)	primary command string (p)	secondary command string (s)
axis 1	1	axis ready	1	„1 4 1 setdoutst“ (set TTL1)	„0 4 1 setdoutst“ (clear TTL1)
axis 2	2	axis ready	1	„1 5 1 setdoutst“ (set TTL2)	„0 5 1 setdoutst“ (clear TTL2)

Now we set the command strings via Venus. With reference to the symbol letters *g*, *i*, *p*, *s* from the table header, syntax is as follows:

1. Primary: *p 0 i g setactcmd*
2. Secondary: *s 1 i g setactcmd*

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

	Command	Description
1:	"1 4 1 setdoutst" 0 1 1 setactcmdint (p. 200)	Assign "set TTL 1" to "axis 1 ready" event.
2:	"0 4 1 setdoutst" 1 1 1 setactcmdint	Assign "clear TTL 1" to "axis 1 not ready" event.
3:	"1 5 1 setdoutst" 0 1 2 setactcmdint	Assign "set TTL 2" to "axis 2 ready" event.
4:	"0 5 1 setdoutst" 1 1 2 setactcmdint	Assign "clear TTL 2" to "axis 2 not ready" event.

At last, we set the *polarity* and *mode* register values. As said above, polarity is to be set to active low (1) at both event sources. In order to provide change of output state with every change of the respective internal condition, *mode* is to be set to 1 at both event sources, too.

Mode and polarity register calculation:

event group	group index	internal condition	source index	bit value	polarity	mode
axis 1	1	axis ready	1	2	1	1
axis 2	2	axis ready	1	2	1	1

polarity register value = 2
mode register value = 2

	Command	Description
1:	2 1 setevtpolint (p. 212)	Set event sensitivity to active low at "axis 1 ready" condition.
2:	2 2 setevtpolint	Set event sensitivity to active low at "axis 2 ready" condition.
3:	2 1 setevtmoint (p. 210)	Set event mode to key function at "axis 1 ready" condition.
4:	2 2 setevtmoint	Set event mode to key function at "axis 2 ready" condition.

The overall configuration would match AC execution pattern 2*b* (s. above).

Initial event generation

As an individual option with each event source, Hydra provides generation of a one-shot action command event - derived from the respective initial logical state - when event detection starts after powerup. This is not always applicable, but useful with applications where some sort of state consistency is crucial.

For instance, if a joystick LED is to be lit when a calibration switch input goes from inactive to active and vice versa, it is obviously reasonable to have that LED switched on initially if the calibration switch is active from the start of event detection. On the other hand, if a move is to be performed whenever a high to low level transition occurs at a particular controller input, initial event generation would lead to an automatic one-shot execution of the move command if that input is low at the start of event detection. This might either be wanted or unwanted, which is for the user to decide.

To cover this, each event group features an *initial action mask register* along with the *event mode* and *event polarity* registers. If a particular register bit is 1, initial event generation at the corresponding event source will be active. Otherwise, initial action will be skipped (which is the default setting).

The following chart summarizes the initial action performance for all configuration and input state constellations available. The *pattern* entry refers to the configuration patterns 1a...4b defined above.

pattern*	initial action mask	initial logical state	initial command execution
1a 2a	any	0	none
	1	1	primary
1b 2b	1	0	primary
	any	1	none
3	0	any	none
	1	any	primary
4a	0	any	none
	1	0	secondary
		1	primary
4b	0	any	none
	1	0	primary
		1	secondary

*see *Action command configuration* table above

Initial action configuration

Initial event example

Example 4

Preconditions: The Hydra controller is equipped with a CAN joystick (manual driver 0). The joystick event group is parameterized as given in *Example 1*.

Problem: During operation, joystick LEDs 1 through 4 will display statuses of all limit switches (on => engaged, off => disengaged). Yet if a switch is initially engaged, the LED status will remain off and not be altered until the next switch engagement occurs.

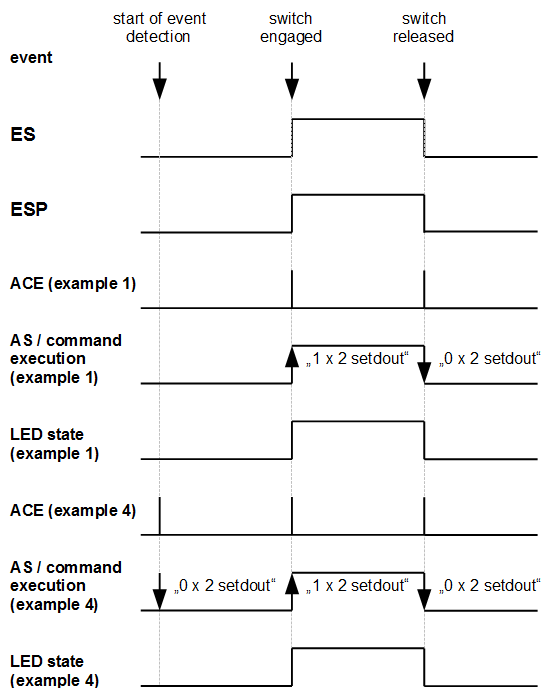
Preparation: We utilize the *Initial action mask register - controller* (p. 193) to incorporate the initial switch engagement and keep switch and LED statuses consistent. Input indexes are 2, 3, 5, and 6, corresponding bits are 1, 2, 4, and 5; so respective bit mask is $2 + 4 + 16 + 32 = 54$.

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

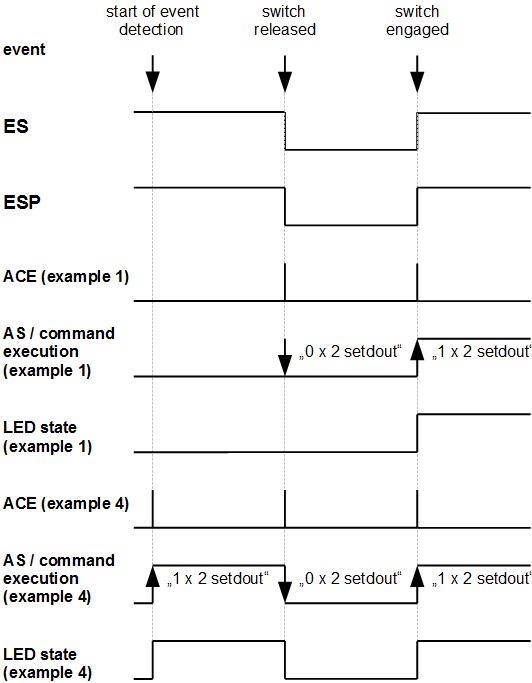
	Command	Description
1:	54 1 <i>setinitactmask</i> (p. 189)	Set bits 1, 2, 4 and 5 in <i>Initial action mask register - controller</i> .
2:	<i>save</i> (p. 242)	Store configuration.
3:	<i>reset</i> (p. 121)	Reset controller.

Result: Statuses of joystick LEDs and limit switches will be consistent from the start.

The following charts depict the action command operation of the controller with examples 1 and 4 after powerup, x being the respective LED index.



**Example 1/4 signal chart (mode = 1,
polarity = 0, initial event state = 0)**

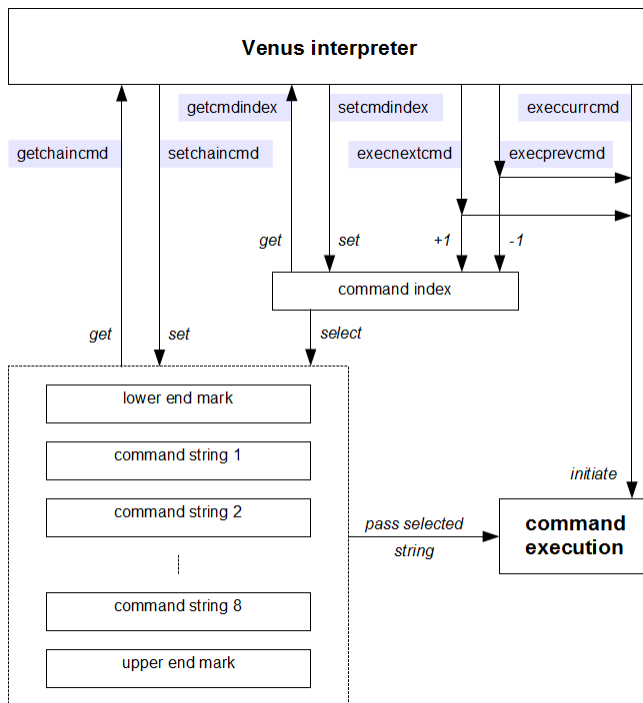


Example 1/4 signal chart (mode = 1, polarity = 0, initial event state = 1)

Introduction to command chain function

A specialty of the Hydra Venus interpreter which needs particular description is the **Command chain** feature. This feature comprises 8 specific string lists (**command chains**) which allow for storing up to 8 command strings each. By application of dedicated Venus commands, these command strings can be triggered one by one, following the list in either ascending or descending order.

This is especially helpful when used in conjunction with the **Action command** feature (see chapter "Introduction to digital I/O processing"), e.g. for raising or lowering a Venus parameter value step by step upon button pressure, or perform individually triggered moves to several invariant stops.



Command chain principle

Above, the **Command chain** principle is depicted.

- The *command strings* to be executed in the chain can directly and individually be accessed by **getchaincmd** (p. 236) and **setchaincmd** (p. 236).
- From all *command strings* defined in the chain, the *command index* selects the one which is up for decoding and execution (1 to 8).
- By application of **execnextcmd** (p. 237), **execprevcmd** (p. 238) or **execcurrcmd** (p. 238), the *command index* is first either incremented or decremented by one or left unaltered, then the selected *command string* will be decoded and executed.
- The *command index* can directly be accessed by **getcmdindex** (p. 240) and **setcmdindex** (p. 240).
- The *command string* array is enframed by *end marks* which mark the array bounds, and where no strings can be deposited. Any string in the array left empty will replace the respective mark. By **setcmdindex**, the *command index* can be set to initially focus a command string or either of these marks (default is 0 = lower end mark). Subsequently, command string execution will provide that the *command index* does not reach these borders again, until **setcmdindex** is applied anew.

Please check the individual command descriptions before going on with the following examples.

Example 1

Preconditions: Start position be 0. Command chain 3 be empty (unused), command index be 0 (=> point to start target).

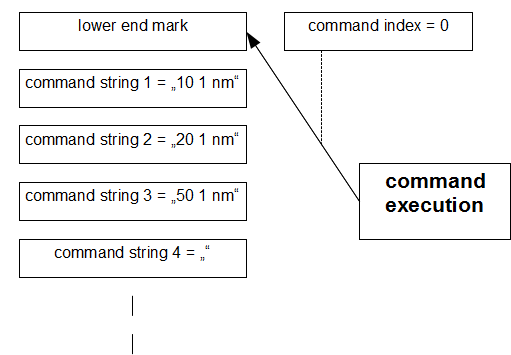
Task: Axis 1 is to be moved to definite stop stations at position coordinates 10 mm, 20 mm, and 50mm, utilizing the command chain feature. Execution of **execnextcmd** (p. 237) is to target next stop in positive direction. Execution of **execprevcmd** (p. 238) is to target next stop in negative direction.

Preparation: We utilize the **nm** (p. 314) command, embedded into command chain 3, for moving to stops.

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

	Command	Description
1:	"10 1 nm" 1 3 setchaincmd (p. 236)	Assign move to first stop at 10 mm to command string 1 of command chain 3.
2:	"20 1 nm" 2 3 setchaincmd	Assign move to second stop at 20 mm to command string 2 of command chain 3.
3:	"50 1 nm" 3 3 setchaincmd	Assign move to third stop at 50 mm to command string 3 of command chain 3.

Result step 1: Position is still 0, command chain 3 is configured and in default state.

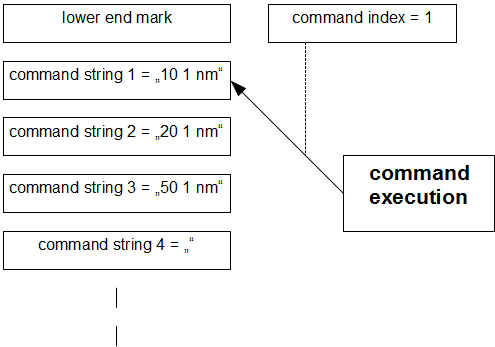


Command chain example step 1

Task: Move axis 1 to first stop.

	Command	Description
1:	3 <i>execnextcmd</i>	Increment command index by one and execute command string 1, i.e. move axis 1 to first stop.

Result step 2: Position is 10 mm, command index is 1.

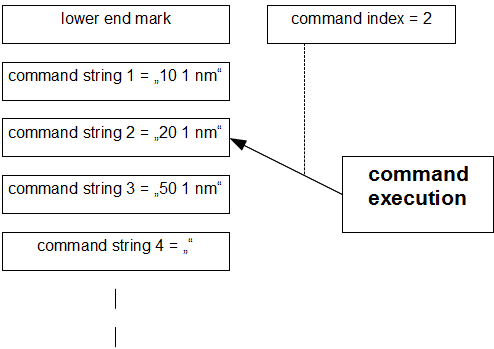


Command chain example step 2

Task: Move axis 1 to next stop in positive direction.

	Command	Description
1:	3 <i>execnextcmd</i>	Increment command index by one and execute command string 2, i.e. move axis 1 to second stop.

Result step 3: Position is 20 mm, command index is 2.

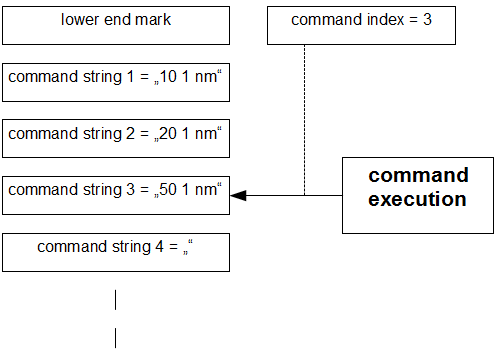


Command chain example step 3

Task: Move axis 1 to next stop in positive direction.

	Command	Description
1:	3 <i>execnextcmd</i>	Increment command index by one and execute command string 3, i.e. move axis 1 to third stop.

Result step 4: Position is 50 mm, command index is 3.

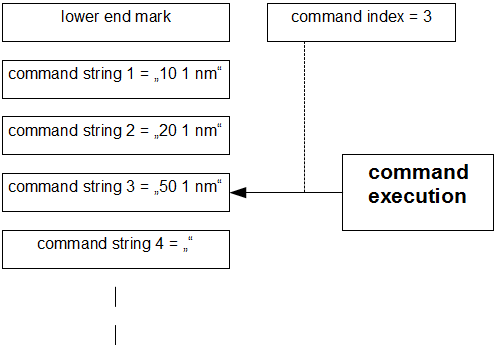


Command chain example step 4

Task: Move axis 1 to next stop in positive direction.

	Command	Description
1:	3 <i>execnextcmd</i>	Next indexed command string (command string 4) is empty => command is void.

Result step 5: Position is 50 mm, command index is 3.

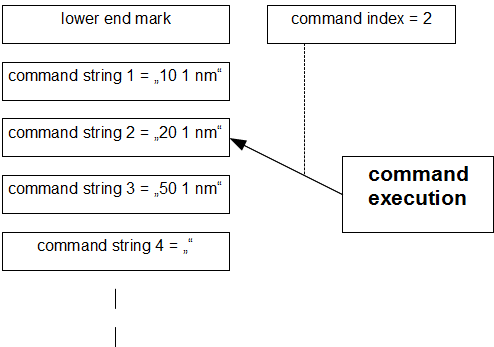


Command chain example step 5

Task: Move axis 1 to next stop in negative direction.

	Command	Description
1:	3 <i>execprevcmd</i>	Decrement command index by one and execute command string 2, i.e. move axis 1 to second stop.

Result step 6: Position is 20 mm, command index is 2.

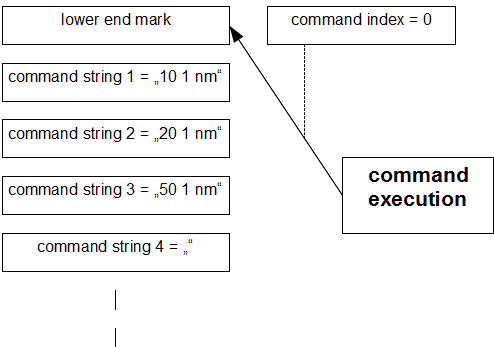


Command chain example step 6

Task: Reset command chain 3 to start anew.

	Command	Description
1:	0 3 <i>setcmdindex</i> (p. 240)	Set command index to 0.

Result step 7: Position is 20 mm, command index is 0.

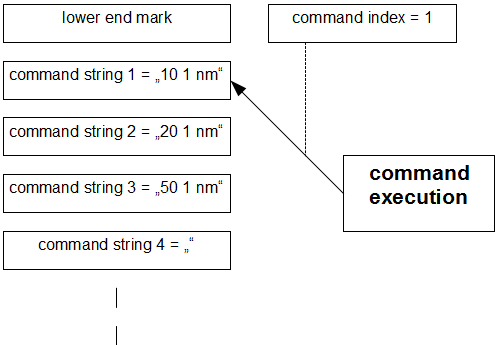


Command chain example step 1

Task: Move axis 1 to first stop.

	Command	Description
1:	3 <i>execnextcmd</i>	Increment command index by one and execute command string 1, i.e. move axis 1 to first stop.

Result step 8: Position is 10 mm, command index is 1.



Command chain example step 2

Example 1a

Task: Have joystick pushbuttons 1 and 2 trigger the commands instead of the host (pushbutton 1 -> positive direction, pushbutton 2 -> negative direction).

Preconditions: CAN joystick is connected.

	Command	Description
1:	"3 execnextcmd" 0 1 2 <i>setactcmd</i> (p. 160)	Write "3 execnextcmd" to primary action command string of CAN joystick 1 pushbutton 1.

	Command	Description
2:	"3 execprevcmd" 0 2 2 setactcmd	Write "3 execprevcmd" to primary action command string of CAN joystick 1 pushbutton 2.

Result: With every push of pushbutton 1, axis 1 will go to location of next stop in positive direction. With every push of pushbutton 2, axis 1 will go to location of next stop in negative direction.

Example 2

Preconditions: CAN handwheel is connected. Command chain 5 be empty (unused).

Task: After powerup, **Velocity** (p. 143) at axis 2 is to be 40 mm/s. It is to be raised at steps of 30 mm/s up to 100 mm/s max. whenever handwheel pushbutton 1 is being pushed. It is to be lowered at steps of 30 mm/s down to 10 mm/s min. whenever handwheel pushbutton 2 is being pushed.

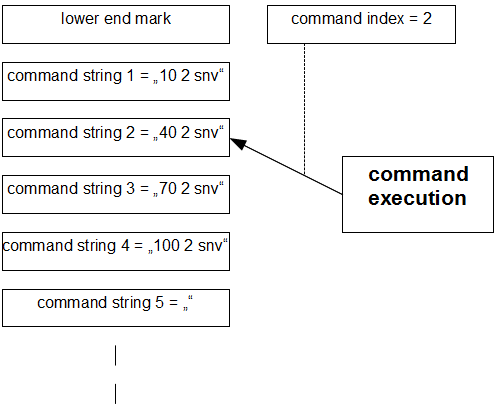
Preparation: We utilize **snv** (p. 144) and command chain 5 for velocity alteration.

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

	Command	Description
1:	40 2 snv	Set Velocity at axis 2 to 40 mm/s.
2:	"10 2 snv" 1 5 setchaincmd (p. 236)	Assign "10 2 snv" to command string 1 of command chain 5.
3:	"40 2 snv" 2 5 setchaincmd	Assign "40 2 snv" to command string 2 of command chain 5.
4:	"70 2 snv" 3 5 setchaincmd	Assign "70 2 snv" to command string 3 of command chain 5.

	Command	Description
5:	"100 2 snv" 4 5 setchaincmd	Assign "100 2 snv" to command string 4 of command chain 5.
6:	2 5 setcmdindex <small>(p. 240)</small>	Set command index of command chain 5 to 2.
7:	"5 execnextcmd" 0 1 3 setactcmd	Write "5 execnextcmd" to primary action command string of CAN handwheel 1 pushbutton 1.
8:	"5 execprevcmd" 0 2 3 setactcmd	Write "5 execprevcmd" to secondary action command string of CAN handwheel 1 pushbutton 2.
9:	save <small>(p. 242)</small>	Save all Venus parameters.
10:	reset <small>(p. 121)</small>	Initiate hardware reset.

Result: After powerup, command chain 5 is configured as required; the default command index is 2 (indexed command: "40 2 snv") to provide consistency with axis 2 **Velocity** setting which is 40 mm/s. Handwheel pushbutton 1 will increase, pushbutton 2 will decrease **Velocity**.



Command chain example 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

Device count.....	115	Serial number.....	122
Device class.....	113	Supply voltage.....	124
Machine error.....	116	Controller name.....	110
Controller version.....	111	CPU temperature.....	112
Reset.....	121	Parameter stack.....	118
Time of day.....	126	Software type.....	123

ControllerIO

Serial communication.....	132	Network.....	129
---------------------------	-----	--------------	-----

Dynamics

Vector acceleration.....	139	Vector velocity.....	141
--------------------------	-----	----------------------	-----

InputOutput

Emergency switch.....	170	Action command - controller (secondary).....	157
Manual driver scan.....	222	Event polarity register - CAN joystick 1.....	184
Manual driver status.....	223	Event mode register - CAN joystick 1.....	178
Manual driver balance.....	220	Initial action mask register - CAN joystick 1.....	192
Event detect register.....	172	Action command - CAN joystick 1 (primary).....	151
Action state register.....	165	Action command - CAN joystick 1 (secondary).....	153
Event polarity entry.....	180	Event polarity register - CAN handwheel 1.....	183
Event mode entry.....	174	Event mode register - CAN handwheel 1.....	177
Initial action mask entry.....	188	Initial action mask register - CAN handwheel 1.....	191
Action command entry.....	159	Action command - CAN handwheel 1 (primary).....	147
Event polarity register - controller.....	185	Action command - CAN handwheel 1 (secondary).....	149
Event mode register - controller.....	179	Event state.....	186
Initial action mask register - controller.....	193	Digital output state.....	168
Action command - controller (primary).....	155		

Interpreter

Configuration storage.....	242	Command chain 1.....	227
Interpreter error.....	249	Command chain 2.....	228
Error decoder.....	247	Command chain 3.....	229
User strings.....	259	Command chain 4.....	230
User doubles.....	255	Command chain 5.....	231
User integers.....	257	Command chain 6.....	232
Command chain entry.....	235	Command chain 7.....	233
Command chain index entry....	239	Command chain 8.....	234
Command chain execution.....	237		

Motion

Vector move.....	331	Controller reference move.....	320
------------------	-----	--------------------------------	-----

Status and position

Controller status..... 425 Controller position..... 423

Axis [Device 1..2]

Controller

Controller identification.....	108	Version.....	127
--------------------------------	-----	--------------	-----

Dynamics

Acceleration.....	135	Stop deceleration.....	137
Velocity.....	143		

InputOutput

Internal action state register....	203	Internal event polarity register.....	212
Internal action command entry.....	198	Internal event mode register....	209
Internal action command (primary).....	194	Internal initial action mask register.....	217
Internal action command (secondary).....	196	Internal event detect register...	207
Internal event state.....	215		

Interpreter

Configuration storage.....	245	Parameter stack.....	253
Interpreter error.....	251		

Manual operation

Manual device entry.....	264	Manual device parameters 2...	271
Manual device parameters 0...	267	Manual device parameters 3...	273
Manual device parameters 1...	269	Manual motion control.....	275

Mechanic

Pitch.....	281	Machine.....	279
------------	-----	--------------	-----

Mechanic setup

Hardware limits.....	291	Calibration velocity.....	289
Initial limits.....	293	Range measure velocity.....	308
Motion direction.....	295	Reference velocity.....	310
Calibration move.....	284	Position origin.....	300
Range measure move.....	306	Position origin configuration....	303
Calibration switch distance.....	287	Motion function.....	297

Motion

Parameterised absolute move.....	323	Relative move.....	329
Parameterised relative move....	324	Reference move.....	327
Random move.....	325	Clock and direction function....	316
Move abortion.....	322	Clock and direction width.....	318
Absolute move.....	314		

Motor

Initial motor power state.....	341	Motor voltage minimum.....	370
Motor pole pairs.....	366	Motor voltage gradient.....	368
Motor form.....	352	Motor current limit.....	346
Motor parameters.....	357	Motor phase current.....	362
Motor optimization stage.....	355	Absolute motor current.....	337
Motor dissipation.....	350	Motor brake.....	343
Motor phase number.....	364	Auto commutation.....	339
Motor current shift.....	348		

Safety functions

Motor restart.....	374	Motor powerdown.....	373
--------------------	-----	----------------------	-----

Sensoric

Sensor amplitudes.....	380	Sensor temperature.....	385
Sensor cache.....	383	Scale period.....	377

Servo

Servo control.....	407	Sensor assignment.....	398
Adaptive positioning control....	388	Sensor status.....	405
Positioning control mode.....	395	Target window.....	412
Positioning control freeze.....	392	Time on target.....	414

Status and position

Axis status.....	417	Device position.....	429
Position display selection.....	431		

Switches and reference mark

Switch status.....	441	Reference configuration.....	434
Switch configuration.....	438	Reference status.....	436

Trigger

Trigger event setup.....	458	Trigger pulse width.....	473
Trigger mode.....	460	Trigger capture mode.....	448
Trigger status.....	475	Trigger capture polarity.....	450
Trigger output delay.....	467	Trigger capture buffer size.....	445
Trigger output pulse width.....	471	Trigger capture index.....	447
Trigger output polarity.....	469	Trigger capture position.....	452
Trigger delay.....	456	Trigger capture position file.....	454

Sensor [Device 3]

This is merely a position measurement facility. No motor can be connected to this device. However, since firmware version 3.0, this device has been given the full Axis device command set to ease the use with driver software. This also means there is full axis set of (usually void) parameters which can be set and read without causing the controller or driver software to pause or hangup. Move commands will be ignored.

Controller

Controller identification.....	108	Version.....	127
--------------------------------	-----	--------------	-----

Dynamics

Acceleration.....	135	Stop deceleration.....	137
Velocity.....	143		

InputOutput

Internal action state register....	203	Internal event polarity register.....	212
Internal action command entry.....	198	Internal event mode register....	209
Internal action command (primary).....	194	Internal initial action mask register.....	217
Internal action command (secondary).....	196	Internal event detect register...	207
Internal event state.....	215		

Interpreter

Configuration storage.....	245	Parameter stack.....	253
Interpreter error.....	251		

Manual operation

Manual device entry.....	264	Manual device parameters 2...	271
Manual device parameters 0...	267	Manual device parameters 3...	273
Manual device parameters 1...	269	Manual motion control.....	275

Mechanic

Pitch.....	281	Machine.....	279
------------	-----	--------------	-----

Mechanic setup

Hardware limits.....	291	Calibration velocity.....	289
Initial limits.....	293	Range measure velocity.....	308
Motion direction.....	295	Reference velocity.....	310
Calibration move.....	284	Position origin.....	300
Range measure move.....	306	Position origin configuration....	303
Calibration switch distance.....	287	Motion function.....	297

Motion

Parameterised absolute move.....	323	Relative move.....	329
Parameterised relative move...	324	Reference move.....	327
Random move.....	325	Clock and direction function....	316
Move abortion.....	322	Clock and direction width.....	318
Absolute move.....	314		

Motor

Initial motor power state.....	341	Motor voltage minimum.....	370
Motor pole pairs.....	366	Motor voltage gradient.....	368
Motor form.....	352	Motor current limit.....	346
Motor parameters.....	357	Motor phase current.....	362
Motor optimization stage.....	355	Absolute motor current.....	337
Motor dissipation.....	350	Motor brake.....	343
Motor phase number.....	364	Auto commutation.....	339
Motor current shift.....	348		

Safety functions

Motor restart.....	374	Motor powerdown.....	373
--------------------	-----	----------------------	-----

Sensoric

Sensor amplitudes.....	380	Sensor temperature.....	385
Sensor cache.....	383	Scale period.....	377

Servo

Servo control.....	407	Sensor assignment.....	398
Adaptive positioning control....	388	Sensor status.....	405
Positioning control mode.....	395	Target window.....	412
Positioning control freeze.....	392	Time on target.....	414

Status and position

Axis status.....	417	Device position.....	429
Position display selection.....	431		

Switches and reference mark

Switch status.....	441	Reference configuration.....	434
Switch configuration.....	438	Reference status.....	436

Trigger

Trigger event setup.....	458	Trigger pulse width.....	473
Trigger mode.....	460	Trigger capture mode.....	448
Trigger status.....	475	Trigger capture polarity.....	450
Trigger output delay.....	467	Trigger capture buffer size.....	445
Trigger output pulse width.....	471	Trigger capture index.....	447
Trigger output polarity.....	469	Trigger capture position.....	452
Trigger delay.....	456	Trigger capture position file.....	454

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

sensorstate

Description

state of the sensor

UNDEFINED=-1

isOK=0

SENSORERROR=1

LOWAMPLITUDE=2

LOWQUALITY=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

deltaStarType = 8

quickStepType = 9

deltaStarEcoType = 10

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

String containing name, type and firmware revision of the device with the specified index.

Commands

nidentify..... 108

Properties

Type: [string](#)

nidentify

Returns the ***Controller identification***.

Syntax

{device} ***nidentify***

Reply

[identifystring]

Examples

Example

	Command	Description
1:	3 <i>nidentify</i>	Return <i>Controller identification</i> of device 3.

Controller name



read-only

Hydra name: "hydra".

Commands

identify..... 110

Properties

Type: [string](#)

identify

Returns the ***Controller name***.

Syntax

identify

Reply

[identifystring]

Examples

Example

	Command	Description
1:	<i>identify</i>	Returns <i>Controller name</i> .

Controller version



read-only

CPU firmware revision.

Commands

version (getversion) 111

Properties

Type: [double](#)

version (getversion)

Returns the actual **Controller version**.

Syntax

version

Reply

[version]

Examples

Example

	Command	Description
1:	<i>version</i>	Returns Controller version .

CPU temperature



read-only

CPU temperature.

Commands

getcputemp..... 112

Properties

Type: double
Unit: °C

getcputemp

Returns the actual *CPU temperature*.

Syntax

getcputemp

Reply

[temperature]

Examples

Example

	Command	Description
1:	<i>getcputemp</i>	Returns <i>CPU temperature</i> .

Device class

Type of a device specified by device number.

Commands

getdeviceclass..... 113

Properties

Name	Type	Description	
[devicenr]	int	numerical device index	
[class code]	int		
		value	description
		0	controller device
		1	axis device
		2	sensor device

getdeviceclass

Returns the ***Device class***.

Syntax

[devicenr] ***getdeviceclass***

Reply

[class code]

Examples

Example

	Command	Description
1:	2 <i>getdeviceclass</i>	Return <i>Device class</i> of device 2.

Device count



read-only

Number of devices minus the controller device.

Commands

getaxc..... 115

Properties

Type: [int](#)

getaxc

Returns the actual *Device count*.

Syntax

getaxc

Reply

[axiscount]

Examples

Example

	Command	Description
1:	<i>getaxc</i>	Returns <i>Device count</i> .

Machine error



read-only

Any error other than ***Interpreter error (Axis)***; normally a hardware error, e.g. a position sensor or CAN controller error. Each single error event results in an error code pushed on a stack. The machine error stack can be read out one by one.

Description of all error codes:

code	description
0	no machine errors
12	motor overcurrent
13	following error
23	Ilt overflow
30	CAN controller error
31	CAN initialization error
32	CAN device version mismatch
33	CAN joystick offset range violation
100	EEPROM checksum error
101	no sensor available
102	sensor not ok
103	sensor position invalid
104	EEPROM write error

Commands

gme (getmerror) 117

Properties

Name	Type
[devicenr]	int
[machine error]	int

gme (getmerror)

Returns the most recent ***Machine error*** (p. 116) code and removes it from the machine error stack. Returns 0 if the stack is empty. The command ***merrordecode*** (p. 248) returns the error description of the code in a string.

Syntax

[devicent] ***gme***

Reply

[machine error]

Examples

	Command	Description
1:	1 <i>gme</i>	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

clear.....118
gsp.....119

Properties

Type: [int](#)

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

Examples

Example

	Command	Description
1:	<i>clear</i>	Clear <i>Parameter stack</i> .

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:	<i>gsp</i> <small>(p. 119)</small>	Returns 0.
2:	0 2	Push 2 parameter values on stack without entering a command string.
3:	<i>gsp</i>	Returns 2.
4:	<i>clear</i> <small>(p. 118)</small>	Clears stack.
5:	<i>gsp</i>	Returns 0 again.

Reset

Controller hardware reset.

Commands

reset..... 121

reset

Initiates **Reset**.



Volatile data will be lost. Execute **save** or **csave** (p. 243) before execution if needed.

Syntax

reset

Examples

Example

	Command	Description
1:	<i>reset</i>	Initiate Reset .

Serial number



read-only

Serial number of controller hardware.

Commands

getserialno..... 122

Properties

Type: [string](#)

getserialno

Returns ***Serial number***.

Syntax

getserialno

Reply

[number]

Examples

Example

	Command	Description
1:	<i>getserialno</i>	Returns <i>Serial number</i> .

Software type



read-only

Commands

getsoftwaretype..... 123

Properties

Type: [string](#)

getsoftwaretype

Returns the ***Software type***.

Syntax

getsoftwaretype

Reply

[type]

Supply voltage



read-only

Supply voltage and type of the motor power stage.

Commands

getusupply..... 124

Properties

Name	Type
[value]	double
[type]	string

getusupply

Returns the **Supply voltage**.

Syntax

getusupply

Reply

[value] [type]

Examples

Example

	Command	Description
1:	<i>getusupply</i>	Returns <i>Supply voltage</i> .

Time of day



read-only

Greenwich mean time.

Commands

gettime..... 126

Properties

Type: [string](#)

gettime

Returns the actual *Time of day*.

Syntax

gettime

Reply

[time]

Examples

Example

	Command	Description
1:	<i>gettime</i>	Returns <i>Time of day</i> .

Version



read-only

Same as **Controller version** (p. 111) . Needs a specified device index, but returns the same result. Compatibility purpose only.

Commands

nversion (getnversion) 127

Properties

Type: double

nversion (getnversion)

Returns the actual **Version**.

Syntax

{device} **nversion**

Reply

[versionnumber]

Examples

Example

	Command	Description
1:	2 nversion	Return Version .

ControllerIO

Network



storable

TCP/IP network.



Regard that for proper TCP/IP operation, the TCP socket option TCP_NODELAY must not be activated by a user PC host software.

Commands

getnetpara.....	129
setnetpara.....	130

Properties

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

getnetpara

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

Syntax

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

Reply

[*Value*]

Examples

Example

	Command	Description
1:	0 <i>getnetpara</i>	Returns controller IP address.

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 <i>setnetpara</i>	Set IP address.
2:	<i>save</i> <small>(p. 242)</small>	Save all controller parameters.

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

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

setbaudrate..... 132
getbaudrate..... 133

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

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

getbaudrate

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

Syntax

[*i*] **getbaudrate**

Reply

[*Value*]

Examples

Example

	Command	Description
1:	1 getbaudrate	Returns baud rate at RS232 port 1.

Dynamics

Acceleration



This states defines how the next move ramps up. The acceleration is normally defined in mm/s². Minimum setting is 1 µm/s²; maximum setting is 500 m/s² (50G).

Commands

gna (getnaccel) 135
 sna (setnaccel) 136

Properties

Type: double

Unit: mm/s²

gna (getnaccel)

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

Syntax

{device} **gna**

Reply

[acceleration]

Examples

Example

	Command	Description
1:	2 <i>gna</i>	Returns <i>Acceleration</i> at axis 2.

sna (setnaccel)

Sets the *Acceleration*.

Syntax

[acceleration] {device} *sna*

Examples

Example

	Command	Description
1:	500 2 <i>sna</i>	Set <i>Acceleration</i> at axis 2 to 0.5 m/s².

Stop deceleration



storable

Deceleration for immediate halt used upon

- touch of either end switch during any move
- **Move abortion** (p. 322)
- *Ctrl+C* shortcut



Replaced by **Acceleration** (p. 135) setting for braking slope calculation whenever that one is currently higher. This does not affect the parameter setting, though.

Commands

ssd (setstopdecel)	137
gsd (getstopdecel)	138

Properties

Type: [double](#)

Unit: [mm/s²](#)

ssd (setstopdecel)

Sets the **Stop deceleration**.

Syntax

[stop_deceleration] {device} **ssd**

Examples

Example

	Command	Description
1:	2000 1 <i>ssd</i>	Set <i>Stop deceleration</i> at axis 1 to 2 m/s².

gsd (getstopdecel)

Returns the current setting of the *Stop deceleration*.

Syntax

{device} *gsd*

Reply

[stop_deceleration]

Examples

Example

	Command	Description
1:	2 <i>gsd</i>	Return <i>Stop deceleration</i> at axis 2.

Vector acceleration



storable

Vectorial acceleration used with **Vector move**^(p. 331).

Commands

sa (setaccel)	139
ga (getaccel)	140

Properties

Type: double
Unit: mm/s²

sa (setaccel)

Sets the **Vector acceleration**.

Syntax

[acceleration] **sa**

Examples

Example

	Command	Description
1:	1000 sa	Set Vector acceleration to 1 m/s ² .

ga (getaccel)

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

Syntax

ga

Reply

[acceleration]

Vector velocity



storable

Vectorial velocity used with **Vector move** (p. 331) .

Commands

sv (setvel)	141
gv (getvel)	142

Properties

Type: double
Unit: mm/s

sv (setvel)

Sets the **Vector velocity**.

Syntax

[velocity] **sv**

Examples

Example

	Command	Description
1:	50 sv	Set Vector velocity to 50 mm/s.

gv (getvel)

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

Syntax

gv

Reply

[velocity]

Velocity



storable

Velocity used by all programmed moves but

- **Calibration move** (p. 284)
- **Range measure move** (p. 306)
- **Reference move** (p. 327)
- **Vector reference move**

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

Commands

gnv (getnvel)	143
snv (setnvel)	144

Properties

Type: [double](#)

gnv (getnvel)

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

Syntax

{device} **gnv**

Reply

[velocity]

Examples

Example

	Command	Description
1:	1 <i>gnv</i>	Returns <i>Velocity</i> at axis 1.

snv (setnvel)

Sets the *Velocity*.

Syntax

[velocity] {device} *snv*

Examples

Example

	Command	Description
1:	50 1 <i>snv</i>	Set <i>Velocity</i> at axis 1 to 50 mm/s.

InputOutput

Before use of the I/O functions, please take at a look at introductory chapter "Introduction to digital I/O processing".

Action command - CAN handwheel 1 (primary)



storable

Set of user definable commands strings executed on certain events at the CAN handwheel 1 pushbuttons, with one string assigned to each pushbutton. Execution conditions are:

event mode *	secondary action command defined ***	event polarity **	execution upon
0	yes	0	every other push
0	yes	1	every other release
0	no	0	every push
1	yes		
0	no	1	every release
1	yes		
1	no	any	every push and every release

***s. Event mode register - CAN handwheel 1** (p. 177)

****s. Event polarity register - CAN handwheel 1** (p. 183)

*****s. Action command - CAN handwheel 1 (secondary)**

(p. 149)



See also introductory chapter "Introduction to digital I/O processing" for general information on how CAN handwheel pushbuttons are processed.

Properties

i	Name	Type
1	[pushbutton 1]	string

<i>i</i>	Name	Type
2	[pushbutton 2]	string
3	[pushbutton 3]	string
4	[pushbutton 4]	string

Action command - CAN handwheel 1 (secondary)



Set of user definable commands strings with one string assigned to each input. On certain events at any of the the CAN handwheel 1 pushbuttons, the respective command string is executed alternately with the respective command string from **Action command - CAN handwheel 1 (primary)** (p. 147) . Execution conditions are:

event mode *	event polarity **	execution upon
0	0	every other push
0	1	every other release
1	0	every release
1	1	every push

*s. **Event mode register - CAN handwheel 1** (p. 177)

s. **Event polarity register - CAN handwheel 1 (p. 183)



See also introductory chapter "Introduction to digital I/O processing" for general information on how CAN handwheel pushbuttons are processed.

Properties

i	Name	Type
1	[pushbutton 1]	string
2	[pushbutton 2]	string
3	[pushbutton 3]	string
4	[pushbutton 4]	string

Action command - CAN joystick 1 (primary)



Set of user definable commands strings executed on certain events at the CAN joystick 1 pushbuttons, with one string assigned to each pushbutton. Execution conditions are:

event mode *	secondary action command defined ***	event polarity **	execution upon
0	yes	0	every other push
0	yes	1	every other release
0	no	0	every push
1	yes		
0	no	1	every release
1	yes		
1	no	any	every push and every release

***s. Event mode register - CAN joystick 1** (p. 178)

****s. Event polarity register - CAN joystick 1** (p. 184)

*****s. Action command - CAN joystick 1 (secondary)**
(p. 153)



See also introductory chapter "Introduction to digital I/O processing" for general information on how CAN joystick pushbuttons are processed.

Properties

i	Name	Type
1	[pushbutton 1]	string
2	[pushbutton 2]	string

<i>i</i>	Name	Type
3	[pushbutton 3]	string
4	[pushbutton 4]	string
5	[pushbutton 5]	string
6	[pushbutton 6]	string

Action command - CAN joystick 1 (secondary)



Set of user definable commands strings with one string assigned to each input. On certain events at any of the the CAN joystick 1 pushbuttons, the respective command string is executed alternately with the respective command string from **Action command - CAN joystick 1 (primary)** (p. 151) . Execution conditions are:

event mode *	event polarity **	execution upon
0	0	every other push
0	1	every other release
1	0	every release
1	1	every push

***s. Event mode register - CAN joystick 1** (p. 178)

****s. Event polarity register - CAN joystick 1** (p. 184)



See also introductory chapter "Introduction to digital I/O processing" for general information on how CAN joystick pushbuttons are processed.

Properties

i	Name	Type
1	[pushbutton 1]	string
2	[pushbutton 2]	string
3	[pushbutton 3]	string
4	[pushbutton 4]	string

<i>i</i>	Name	Type
5	[pushbutton 5]	string
6	[pushbutton 6]	string

Action command - controller (primary)



Set of user definable commands strings executed on certain events at the digital controller inputs, with one string assigned to each input. Execution conditions are:

event mode *	secondary action command defined ***	event polarity **	execution upon
0	yes	0	every other rising edge
0	yes	1	every other falling edge
0	no	0	every rising edge
1	yes		
0	no	1	every falling edge
1	yes		
1	no	any	every rising and every falling edge

*s. **Event mode register - controller** (p. 179)

s. **Event polarity register - controller (p. 185)

***s. **Action command - controller (secondary)** (p. 157) .



See also introductory chapter "Introduction to digital I/O processing" for general information on how digital inputs are processed.

Properties

i	Name	Type	Description
1	[input 1]	string	condition true if input set to high level
2	[input 2]	string	condition true if input set to high level
3	[input 3]	string	condition true if input set to high level
4	[input 4]	string	condition true if input set to high level

<i>i</i>	Name	Type	Description
5	[input 5]	string	
6	[input 6]	string	

Action command - controller (secondary)



storable

Set of user definable commands strings with one string assigned to each input. On certain events at any of the the digital controller inputs, the respective command string is executed alternately with the respective command string from **Action command - controller (primary)** (p. 155). Execution conditions are:

event mode *	event polarity **	execution upon
0	0	every other rising edge
0	1	every other falling edge
1	0	every falling edge
1	1	every rising edge

*s. **Event mode register - controller** (p. 179)

s. **Event polarity register - controller (p. 185)



See also introductory chapter "Introduction to digital I/O processing" for general information on how digital inputs are processed.

Properties

i	Name	Type
1	[input 1]	string
2	[input 2]	string
3	[input 3]	string
4	[input 4]	string

<i>i</i>	Name	Type
5	[input 5]	string
6	[input 6]	string

Action command entry

Entry which allows for access to a specified **Action command** of a specified event source. The target I/O group is selected by the *group index*, whereas the *source index* allows for choice of one individual event source out of the group. Either the primary or the secondary command is focused, depending on the *command index*.



Several single command lines may be concatenated to one string. Use the space character (0x20) for separation. Overall string length is limited to 255 characters, though.

group index	event index *	command index	target command string set
1	1...6	0	Action command - controller (primary) <small>(p. 155)</small>
1	1...6	1	Action command - controller (secondary) <small>(p. 157)</small>
2	1...6	0	Action command - CAN joystick 1 (primary) <small>(p. 151)</small>
2	1...6	1	Action command - CAN joystick 1 (secondary) <small>(p. 153)</small>
3	1...4	0	Action command - CAN handwheel 1 (primary) <small>(p. 147)</small>
3	1...4	1	Action command - CAN handwheel 1 (secondary) <small>(p. 149)</small>

* **[i]** column entry in properties table of target string set



See also introductory chapter "Introduction to digital I/O processing" for general information on how digital inputs and CAN device pushbuttons are processed.

Commands

setactcmd.....	160
getactcmd.....	163

Properties

Name	Type	Description
[group index]	int	specifies target I/O group
[event index]	int	specifies target event source
[command index]	int	specifies target command string entry
[value]	string	command string; will take up to 255 characters.

setactcmd

Sets the **Action command** for the specified event source.

Syntax

[value] [command index] [event index] [group index] **setactcmd**

Examples

Example 1

Task:

Primary action for pushbutton 3 at CAN joystick 1 is to open the position control loop at axis 1.

Primary action for pushbutton 4 at CAN joystick 1 is to open the position control loop at axis 2.

Secondary action for pushbutton 3 at CAN joystick 1 is to close the position control loop at axis 1.

Secondary action for pushbutton 4 at CAN joystick 1 is to close the position control loop at axis 2.

Preparation:

I/O group index for CAN joystick 1 is 2.

Pushbutton 3 index is 3.

Pushbutton 4 index is 4.

Command strings for the actions above in their order of appearance:

"0 1 setcloop"

"0 2 setcloop"

"1 1 setcloop"

"1 2 setcloop"

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

	Command	Description
1:	"0 1 setcloop" 0 3 2 <i>setactcmd</i>	Set primary action for pushbutton 3.
2:	"0 2 setcloop" 0 4 2 <i>setactcmd</i>	Set primary action for pushbutton 4.
3:	"1 1 setcloop" 1 3 2 <i>setactcmd</i>	Set secondary action for pushbutton 3.

	Command	Description
1:	"1 2 setcloop" 1 4 2 setactcmd	Set secondary action for pushbutton 4.

Result:

Pushbutton 3 at CAN joystick 1 will alternately open and close the position control loop at axis 1.

Pushbutton 4 at CAN joystick 1 will alternately open and close the position control loop at axis 2.

Example 2

Task:

Like above, but additionally show loop status via joystick LEDs 1 and 2.

Preparation:

Altered command strings:

"0 1 setcloop 0 1 2 setdoutst"

"0 2 setcloop 0 2 2 setdoutst"

"1 1 setcloop 1 1 2 setdoutst"

"1 2 setcloop 1 2 2 setdoutst"

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

	Command	Description
1:	"0 1 setcloop 0 1 2 setdoutst" 0 3 2 setactcmd	Set primary action for pushbutton 3.
2:	"0 2 setcloop 0 2 2 setdoutst" 0 4 2 setactcmd	Set primary action for pushbutton 4.

	Command	Description
1:	"1 1 setcloop 1 1 2 setdoutst" 1 3 2 setactcmd	Set secondary action for pushbutton 3.
2:	"1 2 setcloop 1 2 2 setdoutst" 1 4 2 setactcmd	Set secondary action for pushbutton 4.

Result:

Pushbutton 3 at CAN joystick 1 will additionally show closed loop state at axis 1 by LED 1 illumination.
Pushbutton 4 at CAN joystick 1 will additionally show closed loop state at axis 2 by LED 2 illumination.

getactcmd

Returns the **Action command** currently set for the specified event source.

Syntax

[command index] [event index] [group index] **getactcmd**

Reply

[value]

Examples

Example

Task: Inquire secondary *Action command* of controller input 4.

Preparation: Controller I/O group index is 1, input 4 source index is 4.

	Command	Description
1:	1 4 1 <i>getactcmd</i>	Returns secondary <i>Action command</i> of controller input 4.

Action state register



Register that is responsible for the primary vs. secondary action command selection with the I/O group specified by the *group index*. Simultaneously shows which command is next to be executed for all event sources of the group. Each event source is represented by one bit.

group index	group	representative bits	event sources
1	controller	0...5	digital inputs 1...6
2	CAN joystick 1	0...5	pushbuttons 1...6
3	CAN handwheel 1	0...3	pushbuttons 1...4

bit value	secondary command defined	next command
0	insignificant	primary
1	yes	secondary
1	no	primary



See also introductory chapter "Introduction to digital I/O processing" for general information on how digital inputs or CAN device pushbuttons are processed.

Commands

getactst.....	166
setactst.....	167

Properties

Name	Type
[group index]	int
[value]	int

getactst

Returns the current content of the specified **Action state register**.

Syntax

[group index] **getactst**

Reply

[value]

Examples

Example

Task: Query content of CAN handwheel 1 **Action state register** (p. 165) .

Preparation: CAN handwheel 1 I/O group index is 3.

	Command	Description
1:	3 getactst	Returns content of CAN handwheel 1 Action state register .

setactst



Sets the content of specified **Action state register**.

To be handled with care! This command is designed to enable the user to set the action command logic to a definite state during normal operation. It is the user's responsibility to provide consistency of the register content with the current logical states of associated internal conditions (considering the configuration). If consistency is not provided, this may lead to improper action command handling.

Syntax

[value] [group index] **setactst**

Examples

Example

Task: Set action state of CAN handwheel 1 pushbutton 3 to 1, all other internal conditions' action states to 0.

Preparation: Bit 2 (corresponding to pushbutton 3) is to be set. All other bits must be cleared. So register value is 4. CAN handwheel 1 I/O group index is 3.

	Command	Description
1:	4 3 setactst	Sets content of CAN handwheel 1 Action state register to 4.

Result: CAN handwheel 1 pushbutton 3 will perform the corresponding secondary action command string upon next action command event. All other CAN handwheel 1 pushbuttons will perform the corresponding primary action command string upon next action command event.

Digital output state



Logical state at specified digital output. The target I/O group is selected by the *group index*, whereas the *output index* allows for choice of one output out of the group.

output type	value description
digital output	0 => low level
	1 => high level
LED	0 => off
	1 => on

Outputs supported at the time:

group index	destination	output index range	outputs
0	internal	none	none
1	controller	3...5	3 => open drain output 4 => TTL output 1* 5 => TTL output 2*
2	CAN joystick 1	1...8	LED1...LED8
3	CAN handwheel 1	1	LED1

* hardware I/O, invariably configured as output

Commands

setdoutst..... 169

Properties

Name	Type
[group index]	int
[output index]	int
[value]	int

setdoutst

Sets *Digital output state* as specified.

Syntax

[value] [output index] [group index] ***setdoutst***

Examples

Example

	Command	Description
1:	1 3 2 <i>setdoutst</i>	Turn on joystick 1 LED 3.
2:	0 1 2 <i>setdoutst</i>	Turn off joystick 1 LED 1.
3:	1 3 1 <i>setdoutst</i>	Set the Hydra open drain output to high level.

Emergency switch



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

getemsw..... 170
setemsw..... 171

Properties

Type: **int**

value	enable state
0	disabled
1	enabled

getemsw

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

Syntax

getemsw

Reply

[mask]

Examples

Example

	Command	Description
1:	<i>getemsw</i>	Returns <i>Emergency switch</i> enable state. Reply be 1.

Result: Emergency switch is enabled.

setemsw

Sets the *Emergency switch* configuration.

Syntax

[mask] *setemsw*

Examples

Example

	Command	Description
1:	0 <i>setemsw</i>	Disables emergency switch.

Event detect register



read-only

Register that simultaneously shows the occurrence of action command events at all event sources of an I/O group specified by the *group index*. Each event source is represented by one bit. A set bit indicates at least one action command event having occurred at the respective event source since last query.

group index	group	representative bits	event sources
1	controller	0...5	digital inputs 1...6
2	CAN joystick 1	0...5	pushbuttons 1...6
3	CAN handwheel 1	0...3	pushbuttons 1...4



See also introductory chapter "Introduction to digital I/O processing" for general information on how digital inputs and CAN device pushbuttons are processed.

Commands

getevtdet..... 173

Properties

Name	Type
[group index]	int
[value]	int

getevtdet

Returns the actual content of the *Event detect register*.

Syntax

[group index] *getevtdet*

Reply

[value]

Examples

Example

Task: Query content of CAN joystick 1 *Event detect register*.

Preparation: CAN joystick 1 I/O group index is 2.

	Command	Description
1:	2 <i>getevtdet</i>	Returns content of CAN joystick 1 <i>Event detect register</i> . Reply be 5.

Result: With bits 0 and 2 set in the register (value = 5), action command events have occurred at CAN joystick 1 pushbuttons 1 and 3 since last query.

Event mode entry

Entry which allows for access to the **Event mode register** of a specified I/O group. The target group is selected by the *group index*.

group index	target register
1	Event mode register - controller (p. 179)
2	Event mode register - CAN joystick 1 (p. 178)
3	Event mode register - CAN handwheel 1 (p. 177)



See also introductory chapter "Introduction to digital I/O processing" for general information on how digital inputs and CAN device pushbuttons are processed.



The proper way of changing the register content is to modify the content, store parameters, and then reset the controller. Immediate continuation of normal operation after changing may lead to inconsistencies and is not recommended.

Commands

getevtmode.....	175
setevtmode.....	175

Properties

Name	Type
[group index]	int
[value]	int

getevtmode

Returns the current **Event mode register** content of the specified I/O group.

Syntax

[group index] **getevtmode**

Reply

[value]

Examples

Example

Task: Query content of **Event mode register - CAN joystick 1** (p. 178) .

Preparation: Internal I/O group index is 2.

	Command	Description
1:	2 getevtmode	Returns content of Event mode register - CAN joystick 1 .

setevtmode

Sets the **Event mode register** content for the specified I/O group.

Syntax

[value] [group index] **setevtmode**

Examples

Example

Task: Set the function of pushbuttons 3 and 4 at CAN joystick 1 to "key".

Preparation: Bits 2 (pushbutton 3) and 3 (pushbutton 4) are to be set. Register value is $4 + 8 = 12$. CAN joystick I/O group index is 2.

Command		Description
1:	12 2 <i>setevtmode</i>	Sets content of <i>Event mode register - CAN joystick 1</i> to 12.

Result: Pushbuttons 3 and 4 will trigger an action command with every push *and* every release event, while all other pushbuttons will trigger action commands with every push *or* every release event, depending on their individual polarity settings.

Event mode register - CAN handwheel 1



storable

Register that contains the mode settings of all pushbuttons at CAN handwheel 1. Each pushbutton is represented by one bit indicating which events (changes of state) at the respective pushbutton will trigger an **Action command** :

bit setting	Action command triggered by
0	every push <i>or</i> every release, depending on polarity setting
1	every push <i>and</i> every release

representative bits	event sources
0...3	pushbuttons 1...4



See also introductory chapter "Introduction to digital I/O processing" for general information on how CAN handwheel pushbuttons are processed.

Properties

Type: `int`

Event mode register - CAN joystick 1



storable

Register that contains the mode settings of all pushbuttons at CAN joystick 1. Each pushbutton is represented by one bit indicating which events (changes of state) at the respective pushbutton will trigger an **Action command** :

bit setting	Action command triggered by
0	every push <i>or</i> every release, depending on polarity setting
1	every push <i>and</i> every release

representative bits	event sources
0...5	pushbuttons 1...6



See also introductory chapter "Introduction to digital I/O processing" for general information on how CAN joystick pushbuttons are processed.

Properties

Type: [int](#)

Event mode register - controller



storable

Register that contains the mode settings of all digital controller inputs. Each input is represented by one bit indicating which events (changes of state) at the respective input will trigger an **Action command** :

bit setting	Action command triggered by
0	every rising <i>or</i> every falling edge, depending on polarity setting
1	every rising <i>and</i> every falling edge

representative bits	event sources
0...5	digital inputs 1...6



See also introductory chapter "Introduction to digital I/O processing" for general information on how digital inputs are processed.

Properties

Type: [int](#)

Event polarity entry

Entry which allows for access to the **Event polarity register** of a specified I/O group. The target group is selected by the *group index*.

group index	target register
1	Event polarity register - controller (p. 185)
2	Event polarity register - CAN joystick 1 (p. 184)
3	Event polarity register - CAN handwheel 1 (p. 183)



See also introductory chapter "Introduction to digital I/O processing" for general information on how digital inputs and CAN device pushbuttons are processed.



The proper way of changing the register content is to modify the content, store parameters, and then reset the controller. Immediate continuation of normal operation after changing may lead to inconsistencies and is not recommended.

Commands

getevtpol.....	181
setevtpol.....	181

Properties

Name	Type
[group index]	int
[value]	int

getevtpol

Returns the current *Event polarity register* content of the specified I/O group.

Syntax

[group index] *getevtpol*

Reply

[value]

Examples

Example

Task: Query content of *Event polarity register - CAN handwheel 1* (p. 183) .

Preparation: CAN handwheel 1 I/O group index is 3.

	Command	Description
1:	3 <i>getevtpol</i>	Returns content of <i>Event polarity register - CAN handwheel 1</i> .

setevtpol

Sets the *Event polarity register* content for the specified I/O group.

Syntax

[value] [group index] *setevtpol*

Examples

Example

Task: Invert the logical state at digital controller inputs 1 and 4.

Preparation: Bits 0 (input 1) and 3 (input 4) are to be set. So register value is $1 + 8 = 9$. I/O group index for the controller is 1.

	Command	Description
1:	9 1 <i>setevtpol</i>	Sets content of <i>Event polarity register - controller</i> (p. 185) to 9.

Result: Input 1 and 4 signals will be inverted before further processing, while all other input signals will be processed directly.

Event polarity register - CAN handwheel 1



storable

Register that contains the polarity settings of all pushbuttons at CAN handwheel 1. Each pushbutton is represented by one bit. A set bit indicates that the respective pushbutton state is being inverted before further processing.

representative bits	event sources
0...3	pushbuttons 1...4



See also introductory chapter "Introduction to digital I/O processing" for general information on how CAN handwheel pushbuttons are processed.

Properties

Type: [int](#)

Event polarity register - CAN joystick 1



Register that contains the polarity settings of all pushbuttons at CAN joystick 1. Each pushbutton is represented by one bit. A set bit indicates that the respective pushbutton state is being inverted before further processing.

representative bits	event sources
0...5	pushbuttons 1...6



See also introductory chapter "Introduction to digital I/O processing" for general information on how CAN joystick pushbuttons are processed.

Properties

Type: [int](#)

Event polarity register - controller



storable

Register that contains the polarity settings of all digital controller inputs. Each input is represented by one bit. A set bit indicates that the respective input state is being inverted before further processing.

representative bits	event sources
0...5	digital inputs 1...6



See also introductory chapter "Introduction to digital I/O processing" for general information on how digital inputs are processed.

Properties

Type: [int](#)

Event state



read-only

Current logical state of a specified event source. The target I/O group is selected by the *group index*, whereas the *input index* allows for choice of one event source out of the group. The value is raw, independent on any polarity or other software configuration.

source type	value description
digital input	0 => low level 1 => high level
pushbutton	0 => released 1 => pressed



See also introductory chapter "Introduction to digital I/O processing" for general information on how digital inputs and CAN device pushbuttons are processed.

Commands

getevtst..... 187

Properties

Name	Type
[group index]	int
[source index]	int
[value]	int

getevtst

Returns the specified event source's actual **Event state**.

Syntax

[source index] [group index] **getevtst**

Reply

[value]

Examples

Example

Task: Query current **Event state** of pushbutton 5 at CAN joystick 1.

Preparation: CAN joystick 1 I/O group index is 2, pushbutton 5 event source index is 5.

	Command	Description
1:	5 2 getevtst	Returns Event state of pushbutton 5 at CAN joystick 1. Reply be 1.

Result: With reply being 1, CAN joystick 1 pushbutton 5 is pressed at the time of query.

Initial action mask entry

Entry which allows for access to the **Initial action mask register** of a specified I/O group. The target group is selected by the *group index*.

group index	target register
1	Initial action mask register - controller (p. 193)
2	Initial action mask register - CAN joystick 1 (p. 192)
3	Initial action mask register - CAN handwheel 1 (p. 191)



See also introductory chapter "Introduction to digital I/O processing" for general information on how digital inputs and CAN device pushbuttons are processed.



Due to its nature, the setting will only show effect after **Configuration storage** (p. 242) and **Reset** (p. 121) .

Commands

getinitactmask.....	189
setinitactmask.....	189

Properties

Name	Type
[group index]	int
[value]	int

getinitactmask

Returns the current ***Initial action mask register*** content of the specified I/O group.

Syntax

[group index] ***getinitactmask***

Reply

[value]

Examples

Example

Task: Query content of ***Initial action mask register - controller*** (p. 193) .

Preparation: Controller I/O group index is 1.

	Command	Description
1:	1 <i>getinitactmask</i>	Returns content of <i>Initial action mask register - controller</i> .

setinitactmask

Sets the ***Initial action mask register*** content for the specified I/O group.

Syntax

[value] [group index] ***setinitactmask***

Examples

Example

Task: Activate the initial event generation at pushbuttons 2 and 4 at CAN handwheel 1 (being inactive at other pushbuttons).

Preparation: Bits 1 (pushbutton 2) and 3 (pushbutton 4) are to be set. Register value is $2 + 8 = 10$. CAN handwheel I/O group index is 3.

Command		Description
1:	10 3 <i>setinitactmask</i>	Sets content of <i>Initial action mask register</i> - <i>CAN handwheel 1</i> <small>(p. 191)</small> to 10.

Result: Initial action event generation will be active and defined commands will be executed after powerup at pushbuttons 2 and 4.

Initial action mask register - CAN handwheel 1



storable

Register that contains the initial action mask settings of all pushbuttons at CAN handwheel 1. Each condition is represented by one bit indicating which signal sources will generate a state event and trigger an initial **Action command** at the start of event detection:

bit setting	initial action
0	no
1	yes

representative bits	event sources
0...3	pushbuttons 1...4



See also introductory chapter "Introduction to digital I/O processing" for general information on how CAN handwheel pushbuttons are processed.

Properties

Type: [int](#)

Initial action mask register - CAN joystick 1



storable

Register that contains the initial action mask settings of all pushbuttons at CAN handwheel 1. Each condition is represented by one bit indicating which signal sources will generate a state event and trigger an initial **Action command** at the start of event detection:

bit setting	initial action
0	no
1	yes

representative bits	event sources
0...5	pushbuttons 1...6



See also introductory chapter "Introduction to digital I/O processing" for general information on how CAN joystick pushbuttons are processed.

Properties

Type: [int](#)

Initial action mask register - controller



Register that contains the initial action mask settings of all digital controller inputs. Each condition is represented by one bit indicating which signal sources will generate a state event and trigger an initial **Action command** at the start of event detection:

bit setting	initial action
0	no
1	yes

representative bits	event sources
0...5	digital inputs 1...6



See also introductory chapter "Introduction to digital I/O processing" for general information on how digital inputs are processed.

Properties

Type: [int](#)

Internal action command (primary)



Set of user definable command strings executed on certain state changes of internal conditions, with one string assigned to each condition. Execution conditions are:

internal event mode *	secondary internal action command defined ***	internal event polarity **	execution upon
0	yes	0	every other false->true transition
0	yes	1	every other true->>false transition
0	no	0	every false->true transition
1	yes		
0	no	1	every true->>false transition
1	yes		
1	no	any	any transition

*S. *Internal event mode register* (p. 209)

**S. *Internal event polarity register* (p. 212)

***S. *Internal action command (secondary)* (p. 196) .



See also introductory chapter "Introduction to digital I/O processing" for general information on how internal conditions are processed.

Properties

i	Name	Type	Description
1	[condition 1]	string	true if axis is powered up (powerup done and motion enabled)
2	[condition 2]	string	true if axis is ready, false if busy

<i>i</i>	Name	Type	Description
3	[condition 3]	string	true if axis is resting on target position; false if performing move or position not in target window

Internal action command (secondary)



storable

Set of user definable command strings with one string assigned to each input. On certain state changes of internal conditions, the respective command string is executed alternately with the respective command string from **Internal action command (primary)** (p. 194) . Execution conditions are:

internal event mode *	internal event polarity **	execution upon
0	0	every other false->true transition
0	1	every other true->>false transition
1	0	every true->>false transition
1	1	every false->true transition

*s. **Internal event mode register** (p. 209)

s. **Internal event polarity register (p. 212)



See also introductory chapter "Introduction to digital I/O processing" for general information on how internal conditions are processed.

Properties

i	Name	Type	Description
1	[condition 1]	string	true if axis is powered up (powerup done and motion enabled)
2	[condition 2]	string	true if axis is ready; false if busy
3	[condition 3]	string	true if axis is resting on target position; false if performing move or position not in target window

Internal action command entry

Entry which allows for access to a specified **Internal action command** of a specified internal condition. The *source index* allows for choice of one individual event source out of the group. Either the primary or the secondary command is focused, depending on the *command index*.



Several single command lines may be concatenated to one string. Use the space character (0x20) for separation. Overall string length is limited to 255 characters, though.

event index *	command index	target command string set
1...2	0	Internal action command (primary) (p. 194)
1...2	1	Internal action command (secondary) (p. 196)

* **[i]** column entry in properties table of target string set



See also introductory chapter "Introduction to digital I/O processing" for general information on how internal conditions are processed.

Commands

getactcmdint.....	199
setactcmdint.....	200

Properties

Name	Type	Description
[event index]	int	specifies target event source
[command index]	int	specifies target command string entry
[value]	string	command string; will take up to 255 characters.

getactcmdint

Returns the specified ***Internal action command*** currently set for the specified internal condition.

Syntax

[value] [command index] [event index] {device} ***getactcmdint***

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

Reply

[value]

Examples

Example

Task: Inquire secondary *Internal action command* of "axis 1 ready" condition.

Preparation: Axis index is 1, "axis ready" condition index is 2.

Command		Description
1:	1 2 1 <i>getactcmdint</i>	Returns secondary <i>Internal action command</i> of "axis 1 ready" condition.

setactcmdint

Sets the specified *Internal action command* for the specified internal condition.

Syntax

[value] [command index] [event index] {device} *setactcmdint*

Examples

Example

Task:

CAN joystick 1 LED 1 is to show "axis 1 powered up" status.

CAN joystick 1 LED 2 is to show "axis 1 ready" status.

CAN joystick 1 LED 5 is to show "axis 2 powered up" status.

CAN joystick 1 LED 6 is to show "axis 2 ready" status.

Preparation:

I/O group index for CAN joystick 1 is 2.

Axis indexes are 1 and 2, respectively.

"Axis powered up" condition index is 1.

"Axis ready" condition index is 2.

Command strings for the actions above in their order of appearance:

"1 1 2 setdoutst" (switch on CAN joystick 1 LED 1)

"0 1 2 setdoutst" (switch off CAN joystick 1 LED 1)

"1 2 2 setdoutst" (switch on CAN joystick 1 LED 2)

"0 2 2 setdoutst" (switch off CAN joystick 1 LED 2)

"1 5 2 setdoutst" (switch on CAN joystick 1 LED 5)

"0 5 2 setdoutst" (switch off CAN joystick 1 LED 5)

"1 6 2 setdoutst" (switch on CAN joystick 1 LED 6)

"0 6 2 setdoutst" (switch off CAN joystick 1 LED 6)

In order to show condition states correctly, conditions' modes must be set to key function, using **setevtmoint**.

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

First setting the action command strings...

	Command	Description
1:	"1 1 2 setdoutst" 0 1 1 setactcmdint	Set primary action for "axis 1 powered up" => switch on CAN joystick LED 1.
2:	"0 1 2 setdoutst" 1 1 1 setactcmdint	Set secondary action for "axis 1 powered up" => switch off CAN joystick LED 1.
3:	"1 2 2 setdoutst" 0 2 1 setactcmdint	Set primary action for "axis 1 ready" => switch on CAN joystick LED 2.
4:	"0 2 2 setdoutst" 1 2 1 setactcmdint	Set secondary action for "axis 1 ready" => switch off CAN joystick LED 2.
5:	"1 5 2 setdoutst" 0 1 2 setactcmdint	Set primary action for "axis 2 powered up" => switch on CAN joystick LED 5.
6:	"0 5 2 setdoutst" 1 1 2 setactcmdint	Set secondary action for "axis 2 powered up" => switch off CAN joystick LED 5.
7:	"1 6 2 setdoutst" 0 2 2 setactcmdint	Set primary action for "axis 2 ready" => switch on CAN joystick LED 6.
8:	"0 6 2 setdoutst" 1 2 2 setactcmdint	Set secondary action for "axis 2 ready" => switch off CAN joystick LED 6.

Now setting the corresponding mode registers...

	Command	Description
1:	3 1 setevtmoint	Set modes for "axis 1 powered up" and "axis 1 ready" to key function.
2:	3 2 setevtmoint	Set modes for "axis 2 powered up" and "axis 2 ready" to key function.

Internal action state register



Register that is responsible for the primary vs. secondary action command selection with the internal conditions. Simultaneously shows which command is next to be executed for all event sources of the group. Each event source is represented by one bit.

bit	condition
0	axis powered up
1	axis ready
2	axis resting on target

bit value	secondary command defined	next command
0	insignificant	primary
1	yes	secondary
1	no	primary



See also introductory chapter "Introduction to digital I/O processing" for general information on how internal conditions are processed.

Commands

setactstint.....	204
getactstint.....	205

Properties

Type: [int](#)

setactstint

Sets the content of *Internal action state register*.



To be handled with care! This command is designed to enable the user to set the action command logic to a definite state during normal operation. It is the user's responsibility to provide consistency of the register content with the current logical states of associated event sources (considering the configuration). If consistency is not provided, this may lead to improper action command handling.

Syntax

[value] {device} ***setactstint***

Examples

Example

Task: Set action state of "axis 2 ready" condition to 1, all other internal conditions' action states at axis 2 to 0.

Preparation: Bit 1 ("axis 2 ready" => internal condition 2) is to be set. All other bits must be cleared. So register value is 2. Axis index is 2.

	Command	Description
1:	2 2 <i>setactstint</i>	Sets content of internal <i>Internal action state register</i> at axis 2 to 2.

Result: "Axis 2 ready" condition will perform the corresponding secondary action command string upon next action command event. All other internal conditions will perform the corresponding primary action command string upon next action command event.

getactstint

Returns the current content of the specified *Internal action state register*.

Syntax

{device} *getactstint*

Reply

[value]

Examples

Example

Task: Query content of axis 2 *Internal action state register* (p. 203) .

Preparation: Axis index is 2.

	Command	Description
1:	2 <i>getactstint</i>	Returns content of axis 2 <i>Internal action state register</i> .

Internal event detect register



Register that simultaneously shows the occurrence of action command events at all internal conditions. Each event source is represented by one bit. A set bit indicates at least one action command event having occurred at the respective event source since last query.

bit	condition
0	axis powered up
1	axis ready
2	axis resting on target



See also introductory chapter "Introduction to digital I/O processing" for general information on how internal conditions are processed.

Commands

getevtdetint..... 207

Properties

Type: [int](#)

getevtdetint

Returns the actual content of the *Internal event detect register*.

Syntax

{device} *getevtdetint*

Reply

[value]

Examples

Example

Task: Query content of axis 1 *Internal event detect register*.

Preparation: Axis index is 1.

	Command	Description
1:	1 <i>getevtdetint</i>	Returns content of axis 1 <i>Internal event detect register</i> . Reply be 2.

Result: With bit 1 set in the register (value = 2), action command events have occurred at "axis 1 ready" condition since last query.

Internal event mode register



Register that contains the mode settings of all internal conditions. Each condition is represented by one bit indicating which events (changes of state) at the respective condition will trigger an ***Internal action command*** :

bit setting	Action command triggered by
0	every false-to-true <i>or</i> every true-to-false transition, depending on polarity setting
1	every true-to-false <i>and</i> every false-to-true transition

bit	condition
0	axis powered up
1	axis ready
2	axis resting on target



See also introductory chapter "Introduction to digital I/O processing" for general information on how internal conditions are processed.

Commands

getevtmodeint.....	210
setevtmodeint.....	210

Properties

Type: **int**

getevtmodeint

Returns the current *Internal event mode register* content.

Syntax

{device} *getevtmodeint*

Reply

[value]

Examples

Example

Task: Query content of *Internal event mode register* (p. 209) at axis 2 .

Preparation: Axis index is 2.

	Command	Description
1:	2 <i>getevtmodeint</i>	Returns content of <i>Internal event mode register</i> at axis 2.

setevtmodeint

Sets the *Internal event mode register* content.

Syntax

[value] {device} *setevtmodeint*

Examples

Example

Task: Set the function of "axis 1 ready" and "axis 1 powered up" conditions to "key".

Preparation: Bits 0 ("axis powered up") and 1 ("axis ready") are to be set. Register value is $1 + 2 = 3$. Axis index is 1.

Command		Description
1:	3 1 <i>setevtmodeint</i>	Sets content of <i>Internal event mode register</i> at axis 1 to 3.

Result: "axis 1 ready" and "axis 1 powered up" conditions will trigger an action command with every logical state transition, while all other conditions will trigger action commands with every true-to-false *or* every false-to-true transition, depending on their individual polarity settings.

Internal event polarity register



Register that contains the polarity settings of all internal conditions. Each condition is represented by one bit. A set bit indicates that the respective condition's logical state is being inverted before further processing.

bit	condition
0	axis powered up
1	axis ready
2	axis resting on target



See also introductory chapter "Introduction to digital I/O processing" for general information on how internal conditions are processed.

Commands

setevtpolint.....	212
getevtpolint.....	213

Properties

Type: [int](#)

setevtpolint

Sets the *Internal event polarity register* content.

Syntax

[value] {device} *setevtpolint*

Examples

Example

Task: Invert the logical state of "axis 2 ready" condition (making "axis 2 busy").

Preparation: Bit 1 ("axis ready" condition) is to be set. So register value is 2. Axis index is 2.

	Command	Description
1:	2 2 <i>setevtpolint</i>	Sets content of <i>Event polarity register - controller</i> to 2.

Result: "axis 2 ready" condition will be inverted before further processing, while all other internal conditions at axis 2 will be processed directly.

getevtpolint

Returns the current *Internal event polarity register* content.

Syntax

{device} *getevtpolint*

Reply

[value]

Examples

Example

Task: Query content of *Internal event polarity register* (p. 212) at axis 1.

Preparation: Axis index is 1.

Command		Description
1:	1 <i>getevtpolint</i>	Returns content of <i>Internal event polarity register</i> at axis 1.

Internal event state



Current logical state of a specified event source. The *source index* allows for choice of the target event source. The value is raw, independent on any polarity or other software configuration.

source type	value description
condition	0 => false
	1 => true

source index	target condition
1	axis powered up
2	axis ready
3	axis resting on target



See also introductory chapter "Introduction to digital I/O processing" for general information on how internal conditions are processed.

Commands

getevtstint..... 216

Properties

Name	Type
[source index]	int
[value]	int

getevtstint

Returns the current ***Internal event state*** of specified condition.

Syntax

[source index] {device} ***getevtstint***

Reply

[value]

Examples

Example

Task: Query current ***Internal event state*** of "axis 2 ready" condition.

Preparation: Axis index is 2, "axis ready" condition index is 2.

	Command	Description
1:	2 2 <i>getevtstint</i>	Returns <i>Internal event state</i> of "axis 2 ready" condition. Reply be 1.

Result: With reply being 1, axis 2 is ready at the time of query.

Internal initial action mask register



Register that contains the initial action mask settings of all internal conditions. Each condition is represented by one bit indicating which signal sources will generate a state event and trigger an initial ***Internal action command*** at the start of event detection:

bit setting	initial action
0	no
1	yes

bit	condition
0	axis powered up
1	axis ready
2	axis resting on target



See also introductory chapter "Introduction to digital I/O processing" for general information on how internal conditions are processed.

Commands

setinitactmaskint.....	218
getinitactmaskint.....	218

Properties

Type: **int**

setinitactmaskint

Sets the *Internal initial action mask register* content for the specified I/O group.

Syntax

[value] {device} *setinitactmaskint*

Examples

Example

Task: Activate the initial event generation at "axis 2 ready" condition (being inactive at other internal conditions).

Preparation: Bits 1 ("axis ready") is to be set. Register value is 2. Axis index is 2.

Command		Description
1:	2 2 <i>setinitactmaskint</i>	Sets content of <i>Internal initial action mask register</i> at axis 2 to 2.

Result: Initial action event generation will be active and defined commands will be executed after powerup at "axis 2 ready" condition.

getinitactmaskint

Returns the current *Internal initial action mask register* content.

Syntax

{device} *getinitactmaskint*

Reply

[value]

Examples

Example

Task: Query content of *Internal initial action mask register* at axis 1.

Preparation: Axis index is 1.

	Command	Description
1:	1 <i>getinitactmaskint</i>	Returns content of <i>Internal initial action mask register</i> at axis 1.

Manual driver balance



Provides for CAN joystick offset compensation and origin-related symmetry at all channels. A balanced joystick will generate

- specified maximum positive output at maximum positive elongation
- specified maximum negative output at maximum negative elongation
- zero output at zero elongation (i.e. with handle in rest position)

driver index	destination
0	CAN joystick 1
1	CAN handwheel 1 (void)



Joystick handle must be left in rest position while balancing. If measured offset exceeds 20% of full range in either direction, respective joystick channel will automatically be disabled, overriding the user settings.



Ineffective with CAN handwheel.

Commands

manbal.....221

Properties

Type: [int](#)

manbal

Performs **Manual driver balance** at specified driver.

Syntax

[driver index] **manbal**

Examples

Example

	Command	Description
1:	0 manbal	Balances CAN joystick 1.

Manual driver scan

Scan for manual drivers newly connected at the CAN interface during operation.

Commands

scanman.....222

scanman

Initiates **Manual driver scan**.

Syntax

scanman

Examples

	Command	Description
1:	scanman	Initiate Manual driver scan .

Manual driver status

Current / last status at specified manual driver; normally 1.
Indicates error if less than 1.

value	description
3	initial check pending
2	version check passed
1	ready / operating
0	CAN node missing
-1	CAN controller error
-2	wheel velocity error
-3	CAN not initialized
-4	version check failed
-5	internal error



See also introductory chapter "Introduction to manual operation" for general information.

Commands

getmanst.....224

Properties

Name	Type
[index]	int
[value]	int

getmanst

Returns the actual **Manual driver status**.

Syntax

[index] **getmanst**

Reply

[value]

Examples

Example 1

Command		Description
1:	0 getmanst	Request status at joystick 1.

Interpreter

Before use of the command chain feature, please take a look at introductory chapter "Introduction to command chain function".

Command chain 1



Table containing *command* strings executed in accordance with the *command index* which provides command selection. See **Command chain execution** (p. 237) on how to use the chain and access the index. See **Command chain entry** (p. 235) on how to access the individual command strings.



See also introductory chapter "Introduction to command chain function" for general information.

Properties

<i>i</i>	Name	Type
0	[command index]	int
1	[command 1]	string
2	[command 2]	string
3	[command 3]	string
4	[command 4]	string
5	[command 5]	string
6	[command 6]	string
7	[command 7]	string
8	[command 8]	string

Command chain 2



Table containing *command* strings executed in accordance with the *command index* which provides command selection. See **Command chain execution** (p. 237) on how to use the chain and access the index. See **Command chain entry** (p. 235) on how to access the individual command strings.



See also introductory chapter "Introduction to command chain function" for general information.

Properties

<i>i</i>	Name	Type
0	[command index]	int
1	[command 1]	string
2	[command 2]	string
3	[command 3]	string
4	[command 4]	string
5	[command 5]	string
6	[command 6]	string
7	[command 7]	string
8	[command 8]	string

Command chain 3



Table containing *command* strings executed in accordance with the *command index* which provides command selection. See **Command chain execution** (p. 237) on how to use the chain and access the index. See **Command chain entry** (p. 235) on how to access the individual command strings.



See also introductory chapter "Introduction to command chain function" for general information.

Properties

<i>i</i>	Name	Type
0	[command index]	int
1	[command 1]	string
2	[command 2]	string
3	[command 3]	string
4	[command 4]	string
5	[command 5]	string
6	[command 6]	string
7	[command 7]	string
8	[command 8]	string

Command chain 4



Table containing *command* strings executed in accordance with the *command index* which provides command selection. See **Command chain execution** (p. 237) on how to use the chain and access the index. See **Command chain entry** (p. 235) on how to access the individual command strings.



See also introductory chapter "Introduction to command chain function" for general information.

Properties

<i>i</i>	Name	Type
0	[command index]	int
1	[command 1]	string
2	[command 2]	string
3	[command 3]	string
4	[command 4]	string
5	[command 5]	string
6	[command 6]	string
7	[command 7]	string
8	[command 8]	string

Command chain 5



Table containing *command* strings executed in accordance with the *command index* which provides command selection. See **Command chain execution** (p. 237) on how to use the chain and access the index. See **Command chain entry** (p. 235) on how to access the individual command strings.



See also introductory chapter "Introduction to command chain function" for general information.

Properties

<i>i</i>	Name	Type
0	[command index]	int
1	[command 1]	string
2	[command 2]	string
3	[command 3]	string
4	[command 4]	string
5	[command 5]	string
6	[command 6]	string
7	[command 7]	string
8	[command 8]	string

Command chain 6



Table containing *command* strings executed in accordance with the *command index* which provides command selection. See **Command chain execution** (p. 237) on how to use the chain and access the index. See **Command chain entry** (p. 235) on how to access the individual command strings.



See also introductory chapter "Introduction to command chain function" for general information.

Properties

<i>i</i>	Name	Type
0	[command index]	int
1	[command 1]	string
2	[command 2]	string
3	[command 3]	string
4	[command 4]	string
5	[command 5]	string
6	[command 6]	string
7	[command 7]	string
8	[command 8]	string

Command chain 7



Table containing *command* strings executed in accordance with the *command index* which provides command selection. See **Command chain execution** (p. 237) on how to use the chain and access the index. See **Command chain entry** (p. 235) on how to access the individual command strings.



See also introductory chapter "Introduction to command chain function" for general information.

Properties

<i>i</i>	Name	Type
0	[command index]	int
1	[command 1]	string
2	[command 2]	string
3	[command 3]	string
4	[command 4]	string
5	[command 5]	string
6	[command 6]	string
7	[command 7]	string
8	[command 8]	string

Command chain 8



Table containing *command* strings executed in accordance with the *command index* which provides command selection. See **Command chain execution** (p. 237) on how to use the chain and access the index. See **Command chain entry** (p. 235) on how to access the individual command strings.



See also introductory chapter "Introduction to command chain function" for general information.

Properties

<i>i</i>	Name	Type
0	[command index]	int
1	[command 1]	string
2	[command 2]	string
3	[command 3]	string
4	[command 4]	string
5	[command 5]	string
6	[command 6]	string
7	[command 7]	string
8	[command 8]	string

Command chain entry

Entry which allows for access to a specified command string of a specified **Command chain** . The target chain is selected by the *chain index*, whereas the *command index* allows for choice of one individual command string out of the chain.



Note that the *command index* is stored along with the chain commands when a **Configuration storage** (p. 242) is done.



Several single command lines may be concatenated to one string. Use the space character (0x20) for separation. Overall string length is limited to 255 characters, though.



See also introductory chapter "Introduction to command chain function" for general information.

Commands

getchaincmd.....	236
setchaincmd.....	236

Properties

Name	Type
[chain index]	int
[command index]	int
[command string]	string

getchaincmd

Returns the command string currently set at the specified index of the specified **Command chain** .

Syntax

[command index] [chain index] **getchaincmd**

Reply

[command string]

setchaincmd

Sets the *command string* at the specified space of the specified **Command chain** .

Syntax

[command string] [command index] [chain index] **setchaincmd**

Command chain execution



Entry which provides several commands for use of the **Command chain** feature. See individual command description for further detail.



Note that the *command index* is stored along with the chain commands when a **Configuration storage** (p. 242) is done.



See also introductory chapter "Introduction to command chain function" for general information.

Commands

execnextcmd.....	237
execcurr cmd.....	238
execprevcmd.....	238

Properties

Type: [int](#)

execnextcmd

Increments *command index* of specified **Command chain** by one, then triggers execution of indexed *command string*.



The command is void whenever

- the command index prior to execution is greater than 7 or
- the command string one step above the currently indexed one is empty

Syntax

[chain index] ***execnextcmd***

execcurrCmd

Triggers execution of currently indexed *command string*.



The command is void whenever the currently indexed command string is empty.

Syntax

[chain index] ***execcurrCmd***

execprevCmd

Decrements *command index* of specified ***Command chain*** by one, then triggers execution of indexed *command string*.



The command is void whenever

- the command index prior to execution is smaller than 2 *or*
- the command string one step below the currently indexed one is empty

Syntax

[chain index] ***execprevCmd***

Command chain index entry

Entry which allows for access to the *command index* of a specified **Command chain** . The target chain is selected by the *chain index*. The *command index* can be set from 0 to 9, although only 8 command strings can be stored in the chain. The 0 and 9 settings correspond to the *start and end marks* of the chain where no command string can be deposited, just to make sure *command strings* 1 and 8 can be accessed from the start.



Note that the *command index* is stored along with the chain commands when a **Configuration storage** (p. 242) is done.



See also introductory chapter "Introduction to command chain function" for general information.

chain index	target chain
1	Command chain 1 (p. 227)
2	Command chain 2 (p. 228)
3	Command chain 3 (p. 229)
4	Command chain 4 (p. 230)
5	Command chain 5 (p. 231)
6	Command chain 6 (p. 232)
7	Command chain 7 (p. 233)
8	Command chain 8 (p. 234)

command index	description
0	lower end mark
1..8	command string selection 1..8
9	upper end mark

Commands

getcmdindex..... 240

setcmdindex..... 240

Properties

Name	Type
[chain index]	int
[command index]	int

getcmdindex

Returns current *command index* of a specified **Command chain** .

Syntax

[chain index] **getcmdindex**

Reply

[command index]

setcmdindex

Sets *command index* of a specified **Command chain** .
The command chain will target the newly indexed command string afterwards.



Consider that the next execution of ***execnextcmd***^(p. 237) or ***execprevcmd***^(p. 238) will alter the *command index* prior to command string evaluation. Set it accordingly.

Syntax

[command index] [chain index] ***setcmdindex***

Configuration storage (Controller)

Storage of non-volatile parameters. Does not extend to content of **Sensor cache** (p. 383) .

Commands

save.....	242
csave.....	243

save

Performs overall **Configuration storage** of all devices.



Note that the storage process will

- halt the Venus interpreter (no processing of further Venus commands at any axis during execution)
- take several seconds to execute

Never switch off the controller during parameter storage to avoid data loss! The proper way to check if the controller has completed storage is to repeatedly transmit an inquiry command such as **nst** (p. 420) and wait until the Venus interpreter starts replying to the inquiries. Then the storage process has finished, and the controller may be reset or switched off.



For storage of data from **Sensor cache** (p. 383), additional execution of **savecache** (p. 383) is required.

Syntax

save

Examples

Example

	Command	Description
1:	save	Initiate Configuration storage of all devices.

csave

Performs **Configuration storage** of controller device.



Note that the storage process will

- halt the Venus interpreter (no processing of further Venus commands at any axis during execution)
- take several seconds to execute

Never switch off the controller during parameter storage to avoid data loss! The proper way to check if the controller has completed storage is to repeatedly transmit an inquiry command such as **nst** and wait until the Venus interpreter starts replying to the inquiries. Then the storage process has finished, and the controller may be reset or switched off.



For storage of data from **Sensor cache**, additional execution of **savecache** is required.

Syntax

csave

Examples

Example

	Command	Description
1:	<i>csave</i>	Initiate <i>Configuration storage</i> of controller device.

Configuration storage (Axis)

Storage of non-volatile parameters. Does not extend to content of **Sensor cache**.

Commands

nsave.....245

nsave

Performs **Configuration storage** of specified axis.



Note that the storage process will

- halt the Venus interpreter (no processing of further Venus commands at any axis during execution)
- take several seconds to execute

Never switch off the controller during parameter storage to avoid data loss! The proper way to check if the controller has completed storage is to repeatedly transmit an inquiry command such as **nst** and wait until the Venus interpreter starts replying to the inquiries. Then the storage process has finished, and the controller may be reset or switched off.



For storage of data from **Sensor cache**, additional execution of **savecache** is required.

Syntax

{device} **nsave**

Examples

Example

	Command	Description
1:	1 <i>nsave</i>	Initiate <i>Configuration storage</i> of axis 1.

Error decoder

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

Commands

errordecode.....	247
merrordecode.....	248

Properties

Name	Type
[errorcode]	int
[errorstring]	string

errordecode

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

Syntax

[errorcode] ***errordecode***

Reply

[errorstring]

Examples

Example

	Command	Description
1:	2000 <i>merrordecode</i>	Returns "undefined command".

merrordecode

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

Syntax

[errorcode] ***merrordecode***

Reply

[errorstring]

Examples

Example

	Command	Description
1:	13 <i>merrordecode</i>	Returns "following error".

Interpreter error (Controller)



read-only

Any error caused by erroneous Venus communication; normally a command format, a command syntax, or a parameter range error. Each single error event results in an error code pushed on a stack. The interpreter error stack can be read out one by one.

Description of all error codes:

code	description
0	no error
4	internal error
100	devicenumber out of range
101	stack underflow or cmd not found at 0
1001	wrong parameter type
1002	stack underflow - too few parameters on stack
1003	parameter out of range
1004	move out of limits requested
1009	parameter stack overflow
2000	undefined command
3000	no configuration file available
3001	error in configuration file, please check it with the style sheet

Commands

ge..... 250

Properties

Type: [int](#)

ge

Returns the most recent ***Interpreter error*** (p. 249) code and removes it from the interpreter error stack. Returns 0 if the stack is empty. The command ***errordecode*** (p. 247) returns the error description of the code in a string.

Syntax

ge

Reply

[errorregister]

Examples

Example

	Command	Description
1:	<i>ge</i>	Returns <i>Interpreter error</i> .

Interpreter error (Axis)



read-only

Any error caused by erroneous Venus communication; normally a command format, a command syntax, or a parameter range error. Each single error event results in an error code pushed on a stack. The interpreter error stack can be read out one by one.

Description of all error codes:

code	description
0	no error
4	internal error
100	devicenumber out of range
101	stack underflow or cmd not found at 0
1001	wrong parameter type
1002	stack underflow - too few parameters on stack
1003	parameter out of range
1004	move out of limits requested
1009	parameter stack overflow
2000	undefined command
3000	no configuration file available
3001	error in configuration file, please check it with the style sheet

Commands

gne..... 252

Properties

Type: [int](#)

gne

Returns the actual *Interpreter error*.

Syntax

{device} *gne*

Reply

[errorregister]

Examples

Example

	Command	Description
1:	1 <i>gne</i>	Return <i>Interpreter error</i> at axis 1.

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

<code>nclear</code>	253
<code>ngsp</code>	254

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***

Examples

Example

	Command	Description
1:	2 <i>nclear</i>	Clears axis <i>Parameter stack</i> .

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]

Examples

Example

	Command	Description
1:	2 <i>ngsp</i>	Returns number of parameters on <i>Parameter stack</i> .

User doubles



storable

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

Commands

getvardbl.....	256
setvardbl.....	256

Properties

<i>i</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

getvardbl

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

Syntax

[*i*] *getvardbl*

Reply

[*Value*]

Examples

Example

	Command	Description
1:	5 <i>getvardbl</i>	Returns entry at index 5 of <i>User doubles</i> .

setvardbl

Sets the the specified *User doubles* entry.

Syntax

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

Examples

Example

	Command	Description
1:	123.456 6 <i>setvardbl</i>	Writes 123.456 to entry 6 of <i>User doubles</i> .

User integers



storable

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

Commands

setvarint.....	257
getvarint.....	258

Properties

<i>i</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**

Examples

Example

	Command	Description
1:	321 2 setvarint	Writes 321 to entry 2 of User integers .

getvarint

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

Syntax

[*i*] **getvarint**

Reply

[*Value*]

Examples

Example

	Command	Description
1:	8 getvarint	Returns entry at index 8 of User integers .

User strings



storable

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

Commands

getvarstring.....	260
setvarstring.....	260

Properties

<i>i</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

getvarstring

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

Syntax

[*i*] *getvarstring*

Reply

[*Value*]

Examples

Example

	Command	Description
1:	3 <i>getvarstring</i>	Returns entry at index 3 of <i>User strings</i> .

setvarstring

Sets the the specified *User strings* entry.

Syntax

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

Examples

Example

	Command	Description
1:	"mystring" <i>setvarstring</i>	4 Writes "mystring" to entry 4 of <i>User strings</i> .

Manual operation

Before operating the controller manually, please take a look at introductory chapter "Introduction to manual operation".

Manual device entry

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** set.



Note that the pushbuttons of a manual driver will only work as long as at least one channel of that particular driver is assigned to one or more enabled manual devices. Routing and enable state have no impact on LED function, though. Use *mode* entry of corresponding **Manual device parameters** set in order to disable position data generation only.



See also introductory chapter "Introduction to manual operation" for general information.

Commands

setmanpara.....	265
getmanpara.....	265

Properties

Name	Type	Description
[control index]	int	selection index of target manual device 1 : Manual device parameters 0 (p. 267) 2 : Manual device parameters 1 (p. 269) 3 : Manual device parameters 2 (p. 271) 4 : Manual device parameters 3 (p. 273)
[parameter index]	int	selection index of target parameter (see respective 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*.

Syntax

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

Examples

Example

	Command	Description
1:	100 1 1 1 <i>setmanpara</i>	Sets the <i>velocity</i> at the 1st manual device of axis 1 to 100 mm/s.
2:	2 6 1 2 <i>setmanpara</i>	Sets the <i>input function</i> 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 accepted. If it is omitted, though, proper execution is still granted, but the Venus interpreter will report an *Interpreter error* (code 1002). This is due to the specific parameter stack handling.

Reply

[parameter value]

Examples

Example

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

Manual device parameters 0



storable

Properties of manual device 0.



A programmed move unconditionally overrides manual motion.



See also introductory chapter "Introduction to manual operation" for general information.

Properties

<i>i</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 invariably 323584 with SMC CAN handwheel
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: enabled until next programmed move; disabled afterwards 2: enabled until next programmed move and afterwards
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)
10	[input driver]	int	-	0: joystick 1

<i>i</i>	Name	Type	Unit	Description
11	[input channel]	int	-	joystick - 0: horizontal, 1: vertical, 2: rotational handwheel - 0: front rotor, 1: rear rotor

Manual device parameters 1



storable

Properties of manual device 1.



A programmed move unconditionally overrides manual motion.



See also introductory chapter "Introduction to manual operation" for general information.

Properties

<i>i</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 invariably 323584 with SMC CAN handwheel
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: enabled until next programmed move; disabled afterwards 2: enabled until next programmed move and afterwards
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)
10	[input driver]	int	-	0: joystick 1

<i>i</i>	Name	Type	Unit	Description
11	[input channel]	int	-	joystick - 0: horizontal, 1: vertical, 2: rotational handwheel - 0: front rotor, 1: rear rotor

Manual device parameters 2



storable

Properties of manual device 2.



A programmed move unconditionally overrides manual motion.



See also introductory chapter "Introduction to manual operation" for general information.

Properties

<i>i</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 invariably 323584 with SMC CAN handwheel
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: enabled until next programmed move; disabled afterwards 2: enabled until next programmed move and afterwards
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)
10	[input driver]	int	-	0: joystick 1

<i>i</i>	Name	Type	Unit	Description
11	[input channel]	int	-	joystick - 0: horizontal, 1: vertical, 2: rotational handwheel - 0: front rotor, 1: rear rotor

Manual device parameters 3



storable

Properties of manual device 3.



A programmed move unconditionally overrides manual motion.



See also introductory chapter "Introduction to manual operation" for general information.

Properties

<i>i</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 invariably 323584 with SMC CAN handwheel
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: enabled until next programmed move; disabled afterwards 2: enabled until next programmed move and afterwards
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)
10	[input driver]	int	-	0: joystick 1

<i>i</i>	Name	Type	Unit	Description
11	[input channel]	int	-	joystick - 0: horizontal, 1: vertical, 2: rotational handwheel - 0: front rotor, 1: rear rotor

Manual motion control



storable

Integrated activation state of all manual devices. Allows for bitwise enabling / disabling.

Bit number	target	description
0	manual device 0	0: disabled 1: enabled
1	manual device 1	0: disabled 1: enabled
2	manual device 2	0: disabled 1: enabled
3	manual device 3	0: disabled 1: enabled



Note that the pushbuttons of a manual driver will only be effective as long as that particular driver is enabled at one axis at least. Enable state has no impact on LED function, though. To maintain pushbutton operation while position data generation is disabled, set corresponding enable bit and use *mode* entry of corresponding **Manual device parameters** set for disabling.



See also introductory chapter "Introduction to manual operation" for general information.

Commands

setmanctl.....	276
getmanctl.....	276

Properties

Type: **int**
bit coded => see above

setmanctrl

Sets the *Manual motion control*.

Syntax

[enable state] {device} *setmanctrl*

Examples

Example

	Command	Description
1:	3 2 <i>setmanctrl</i>	Enable <i>Manual device 0</i> and <i>Manual device 1</i> at Axis 2, disable others.

getmanctrl

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

Syntax

{device} *getmanctrl*

Reply

[enable state]

Examples

Example

	Command	Description
1:	1 <i>getmanctrl</i>	Returns <i>Manual motion control</i> at axis 1.

Mechanic

Machine



storable

Commands

getmacpara.....	279
setmacpara.....	280

Properties

<i>i</i>	Name	Type
1	[inertia]	double

getmacpara

Returns the current setting of the ***Machine***.

Syntax

[*i*] {device} ***getmacpara***

Reply

[*Value*]

setmacpara

Sets the ***Machine***.

Syntax

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

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. 366)

Commands

getpitch..... 281
setpitch..... 282

Properties

Type: double

Unit: mm

getpitch

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

Syntax

{device} ***getpitch***

Reply

[pitch]

Examples

Example

	Command	Description
1:	1 <i>getpitch</i>	Returns <i>Pitch</i> at axis 1.

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 <i>setpitch</i>	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. 287) 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. 429))
- setting of **Hardware limits** (p. 291) :
 - *lowerlimit* will be set to 0
 - *upperlimit* will be set to *upperlimit of Initial limits* (p. 293)
- motor optimization (if configured accordingly)



Upon execution of **Ctrl+C** or **nabort** (p. 322), the slide/rotor will halt immediately. Just as with regular move termination, the following additional operation is done:

- definition of actual position as new origin of the reference coordinate system, i.e nominal position is 0 afterwards (see **Device position**)
- setting of **Hardware limits**:
 - *lowerlimit* will be set to 0
 - *upperlimit* will be set to *upperlimit of Initial limits*

Note, however, that this is done at the moment of stop request. So after reaching standstill, the slide/rotor will be located slightly below the origin, outside the newly defined motion range (depending on **Velocity** (p. 143) , **Stop deceleration** (p. 137) and **Acceleration** (p. 135)).



See ***Motion function*** (p. 297) on move configuration. See also introductory chapter "Motor commutation handling" on motor optimization.

Commands

ncal (ncalibrate)286

ncal (ncalibrate)

Initiates a ***Calibration move***.

Syntax

{device} ***ncal***

Examples

Example

	Command	Description
1:	1 <i>ncal</i>	Initiates <i>Calibration move</i> at axis 1.

Calibration switch distance



storable

Distance from "Cal" end switch at which **Calibration move** (p. 284) terminates.

Commands

getncalswdist.....287
setncalswdist.....288

Properties

Type: double
Unit: mm

getncalswdist

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

Syntax

{device} **getncalswdist**

Reply

[dist from switch]

Examples

Example

	Command	Description
1:	2 <i>getncalswdist</i>	Returns <i>Calibration switch distance</i> at axis 2.

setncalswdist

Sets the *Calibration switch distance*.

Syntax

[dist from switch] {device} *setncalswdist*

Examples

Example

	Command	Description
1:	1 1 <i>setncalswdist</i>	Set axis 1 <i>Calibration switch distance</i> to 1 mm.

Calibration velocity



Velocity at which a **Calibration move** (p. 284) will run. Set separately for motion towards switch and backward motion, respectively.

Commands

getncalvel..... 289
setncalvel..... 290

Properties

<i>i</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

{device} **getncalvel**

Reply

[vel into switch] [vel out of switch]

Examples

Example

	Command	Description
1:	1 <i>getncalvel</i>	Returns <i>Calibration velocity</i> at axis 1.

setncalvel

Sets the *Calibration velocity*.

Syntax

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

Examples

Example

	Command	Description
1:	3 2 <i>setncalvel</i>	Set axis 2 <i>Calibration velocity</i> to 3 mm/s.

Hardware limits



Limits of available motion range, applying to all moves except **Calibration move** (p. 284) and **Range measure move** (p. 306). 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. 293).

Commands

getnlimit.....	291
setnlimit.....	292

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]

Examples

Example

	Command	Description
1:	2 <i>getnlimit</i>	Return <i>Hardware limits</i> at axis 2.

setnlimit

Sets the *Hardware limits*.

Syntax

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

Examples

Example

	Command	Description
1:	-50 200 1 <i>setnlimit</i>	Set axis 1 <i>Hardware limits</i> to -50 (<i>lower limit</i>) and 200 (<i>upper limit</i>).

Initial limits



Defines storable values for the initialization of the **Hardware limits** (p. 291) . 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. 284) .



Regard that the initial limits will not be matched upon change of the **Position origin** (p. 300) .



Setting both initial limits to 0 will reset them to their defaults (+/- 200 m).

Commands

setinilimit.....	293
getinilimit.....	294

Properties

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

setinilimit

Sets the **Initial limits**.

Syntax

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

Examples

Example

	Command	Description
1:	-70 70 1 <i>setinilimit</i>	Set axis 1 <i>Initial limits</i> to -70 (<i>lower limit</i>) and 70 (<i>upper limit</i>).

getinilimit

Returns the current setting of the *Initial limits*.

Syntax

{device} *getinilimit*

Reply

[lowerlimit] [upperlimit]

Examples

Example

	Command	Description
1:	1 <i>getinilimit</i>	Returns <i>Initial limits</i> at axis 1.

Motion direction



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

setmotiondir.....	295
getmotiondir.....	296

Properties

Type: [int](#)

setmotiondir

Sets the ***Motion direction***.



Since firmware version 3.000:

Note that a change of the ***Motion direction*** setting will not take effect immediately. The requested configuration will only be effective on next ***Reset***.

Syntax

[direction] {device} **setmotiondir**

Examples

Example

	Command	Description
1:	1 1 setmotiondir	Sets axis 1 Motion direction reverse (in reference to default).
2:	save	Stores the complete parameters set.

getmotiondir

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

Syntax

{device} **getmotiondir**

Reply

[direction]

Examples

Example

	Command	Description
1:	1 getmotiondir	Returns Motion direction at axis 1.

Motion function



storable

Defines the respective activation states of **Calibration move** (p. 284), **Range measure move** (p. 306) and **Reference move** (p. 327) by a bit-coded value. If the respective bit is low, each corresponding move request will be discarded. Also defines if automatic motor optimization is done and, if so (i.e. if corresponding bit value is high), the home location for motor optimization.



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



Note that motor optimization is only useful with linear and AC motors. See also introductory chapter "Motor commutation handling" on motor optimization. Inquire **Motor optimization stage** (p. 355) for information on what step to take next.

Bit number	description		
0	Calibration move activation		
1	Range measure move activation		
2	Reference move activation		
3	Motor optimization activation		
4	Motor optimization selection		
	bit value	home location	initiating move
	0	cal limit switch	Calibration move
	1	reference mark	Reference move

Commands

getmotionfunc.....	298
setmotionfunc.....	298

Properties

Type: [int](#)

getmotionfunc

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

Syntax

{device} ***getmotionfunc***

Reply

[mask]

Examples

Example

	Command	Description
1:	1 <i>getmotionfunc</i>	Returns <i>Motion function</i> at axis 1.

setmotionfunc

Sets the ***Motion function***.

Syntax

[mask] {device} **setmotionfunc**

Examples

Example

Task: Disable *Range measure move* (p. 306), but leave *Calibration move* (p. 284) and *Reference move* (p. 327) enabled. Use *Reference move* for motor optimization.

	Command	Description
1:	29 1 setmotionfunc	Sets bits 4, 3, 2, and 0 and clears bit 1 in the register.

Result: Any *Range measure move* request will be ignored afterwards. Subsequent to each *Reference move*, a motor optimization will take place.

Position origin



storable

Defines location of position origin relative to its initial location.

Commands

getnpos.....	300
setnpos.....	301

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. 301)/**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. 303)) 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



Configures the handling of the **Position origin** (p. 300) . The only parameter to be configured at the time is the effective direction, i.e. the sign of the origin.

Commands

getorgconfig.....	303
setorgconfig.....	304

Properties

<i>i</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**

Parameter index *i* is for expandability purpose only. For the time being, *sign* is the only accessible parameter (1 is only valid *i* value).

Reply

[*Value*]

Examples

Example

Command		Description
1:	1 2 <i>setorgconfig</i>	Return <i>sign</i> entry of <i>Position origin configuration</i> at axis 2.

setorgconfig

Sets the *Position origin configuration*.

Syntax

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

Parameter index *i* is for expandability purpose only. For the time being, *sign* is the only accessible parameter (1 is only valid *i* value).

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 <small>(p. 429)</small>	Returns -10.000000.
6:	2 np	Returns 10.000000.
7:	1 getnpos <small>(p. 300)</small>	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. 291)) to actual position



Upon execution of *Ctrl+C* or ***nabort*** (p. 322), the device will halt immediately. The upper hardware limit will immediately be set to the current nominal position at the moment of stop request. So after reaching standstill, the slide/rotor will be located slightly outside the newly defined motion range (depending on ***Velocity*** (p. 143) , ***Stop deceleration*** (p. 137) and ***Acceleration*** (p. 135)).



See ***Motion function*** (p. 297) on move configuration.

Commands

nrm (nrangemeasure)306

nrm (nrangemeasure)

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

Syntax

{device} ***nrm***

Examples

Example

	Command	Description
1:	2 <i>nrm</i>	Initiates <i>Range measure move</i> at axis 2.

Range measure velocity



Velocity at which a **Range measure move** (p. 306) will run. Set separately for motion towards switch and motion out of switch, respectively.

Commands

setnrmvel.....	308
getnrmvel.....	309

Properties

<i>i</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**

Examples

Example

	Command	Description
1:	5 1 <i>setnrmvel</i>	Set axis 1 <i>Range measure velocity</i> to 5 mm/s.

getnrmvel

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

Syntax

{device} *getnrmvel*

Reply

[vel into switch] [vel out of switch]

Examples

Example

	Command	Description
1:	2 <i>getnrmvel</i>	Returns <i>Range measure velocity</i> at axis 2.

Reference velocity



Defines motion velocity for reference position mark detection. The setting takes effect upon execution of *refmove* (p. 320) and *nrefmove* (p. 328).

Commands

getnrefvel.....310
setnrefvel.....311

Properties

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

getnrefvel

Returns the current setting of the *Reference velocity*.

Syntax

{device} *getnrefvel*

Reply

[forward velocity] [backward velocity]

Examples

Example

	Command	Description
1:	1 <i>getnrefvel</i>	Returns <i>Reference velocity</i> at axis 1.

setnrefvel

Sets the *Reference velocity*.

Syntax

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

Examples

Example

	Command	Description
1:	20 1 2 <i>setnrefvel</i>	Set axis 2 <i>forward Reference velocity</i> to 20 mm/s.



Motion

Before use of clock and direction interface, please take a look at introductory chapter "Introduction to clock and direction operation".

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.



Upon execution of *Ctrl+C* or ***nabort*** (p. 322), the slide/rotor will halt immediately.

Commands

nm (nmove) 314

Properties

Type: double

nm (nmove)

Initiates an ***Absolute move***.

Syntax

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

Examples

Example

	Command	Description
1:	10 1 <i>nm</i>	Initiate a move at axis 1 to coordinate 10 mm.

Clock and direction function



For use with QuickStep clock/direction interface only. Enable state of clock/direction interface. When enabled, target position is exclusively determined by the QuickStep counter; programmed and manual moves will be disabled. Offers optional position alignment to match the counter position to the current nominal position when activated. For changing the position to clock ratio, see ***Clock and direction width*** (p. 318) .

Commands

setcdfunc.....	316
getcdfunc.....	317

Properties

Type: [int](#)

value	description
0	programmed/manual operation
1	clock/direction operation, no position alignment upon activation
2	clock/direction operation, position alignment upon activation

setcdfunc

Enables or disables the ***Clock and direction function***.

Syntax

[enabled] {device} ***setcdfunc***

Examples

Example

	Command	Description
1:	2 1 <i>setcdfunc</i>	Enable the <i>Clock and direction function</i> at axis 1. Counter position is aligned before taking up clock and direction operation.

getcdfunc

Returns current enable state of ***Clock and direction function***.

Syntax

{device} ***getcdfunc***

Reply

[enabled]

Examples

Example

	Command	Description
1:	1 <i>getcdfunc</i>	Returns <i>Clock and direction function</i> enable state at axis 1.

Clock and direction width



For use with QuickStep clock/direction interface only. Position to clock ratio, i.e. distance covered upon reception of a single clock pulse whenever ***Clock and direction function*** (p. 316) is enabled.

Commands

getcdwidth..... 318
setcdwidth..... 319

Properties

Type: [int](#)
Unit: [nm](#)

getcdwidth

Returns the current setting of the ***Clock and direction width***.

Syntax

{device} ***getcdwidth***

Reply

[value]

Examples

Example

	Command	Description
1:	2 <i>getcdwidth</i>	Return <i>Clock and direction width</i> at axis 2.

setcdwidth

Sets the *Clock and direction width*.

Syntax

[value] {device} *setcdwidth*

Examples

Example

	Command	Description
1:	1000 2 <i>setcdwidth</i>	Set <i>Clock and direction width</i> at axis 2 to 1 μ m / pulse.

Controller reference move

Extended version of **Reference move** (p. 327) ; 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. 310) and **Reference configuration** (p. 434) .



Upon execution of **Ctrl+C** or **nabort** (p. 322), the slide(s)/rotor(s) will halt immediately.

Commands

refmove..... 320

Properties

Type: double

Unit: mm

refmove

Initiates a **Controller reference move**.

Syntax

[distance] **refmove**

Examples

Example

	Command	Description
1:	15 1 <i>refmove</i>	Initiate <i>Controller reference move</i> which will find reference mark or cover 15 mm distance at each axis individually.

Move abortion

Immediate stop of currently running move. Braking slope is set according to either **Stop deceleration** (p. 137) or **Acceleration** (p. 135) - whatever setting is higher.

Commands

nabort..... 322

nabort

Executes **Move abortion**.

Syntax

{device} **nabort**

Examples

Example

	Command	Description
1:	1 nabort	Initiate Move abortion at axis 1.

Parameterised absolute move



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

pm..... 323

Properties

Name	Type	Unit	Description
[targetposition]	double	mm	target position of move
[velocity]	double	mm/s	target velocity during move
[acceleration]	double	mm/s ²	acceleration / deceleration
[deceleration]	double	-	noneffective

pm

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

Syntax

{device} ***pm***

Parameterised relative move



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

pr..... 324

Properties

Name	Type	Unit	Description
[distance]	double	mm	move distance to be covered
[velocity]	double	mm/s	target velocity during move
[acceleration]	double	mm/s ²	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 individual move are calculated in a way it will not take more than about 8 seconds. Move velocity will be **Velocity** (p. 143) at most, and move acceleration/deceleration will be **Acceleration** (p. 135) .



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. 286) and **nrm** (p. 306) (in this order).



Upon execution of **Ctrl+C** or **nabort** (p. 322), the slide/rotor will halt immediately.

Commands

nrandmove..... 325

nrandmove

Executes **Random move**.

Syntax

{device} **nrandmove**

Examples

Example

	Command	Description
1:	2 <i>nrandmove</i>	Initiates <i>Random move</i> at axis 2.

Reference move

Relative move covering the given distance. While moving, reference position mark detection is performed according to **Reference configuration** (p. 434). 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. 310) and **Reference status** (p. 436). Upon matching configuration, move will conclude with motor optimization, provided a reference mark was found.



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



Upon execution of **Ctrl+C** or **nabort** (p. 322), the slide/rotor will halt immediately.



See **Motion function** (p. 297) on move configuration. See also introductory chapter "Motor commutation handling" on motor optimization.

Commands

nrefmove..... 328

Properties

Type: double
Unit: mm

nrefmove

Initiates a **Reference move**.

Syntax

[distance] {device} **nrefmove**

Examples

Example

	Command	Description
1:	75 2 nrefmove	Initiates Reference move at axis 2 which targets either reference or position coordinate 75 mm from start position.

Relative move

Move covering the given *distance*. Uses **Velocity**_(p. 143) and **Acceleration**_(p. 135).



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.



Upon execution of *Ctrl+C* or ***nabort***_(p. 322), the slide/rotor will halt immediately.

Commands

nr (nrmove)329

Properties

Type: double
Unit: mm

nr (nrmove)

Executes **Relative move**.



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*

Examples

Example

	Command	Description
1:	-20 1 <i>nr</i>	Initiates Relative move at axis 1 which targets position coordinate -20 mm from start position.

Vector move

Vectorial move, the trajectory course of which is specified by

- the cartesian x and y position components of the target location or move distance
- the **Vector velocity** (p. 141)
- the **Vector acceleration** (p. 139)

The controller will move the object along a straight line approximately.



Upon execution of *Ctrl+C* or **nabort** (p. 322), the slide(s)/rotor(s) will halt immediately.

Example 1

Preconditions: Current position is 0 at both axes.

Task: Move object by a distance of 5 mm with 30° angle against Axis 1 at 10 mm/s target velocity and 100 mm/s² acceleration.

	Command	Description
1:	10 sv <small>(p. 141)</small>	Set the vectorial velocity at 10 mm/s.
2:	100 sa <small>(p. 139)</small>	Set the vectorial acceleration at 100 mm/s ² .
3:	4.330127 2.5 r <small>(p. 334)</small>	Initiate relative move to target location x component is 5 mm * cos(30°) = 4.330127 mm y component is 5 mm * sin(30°) = 2.5 mm

Result: Both axes move to target location simultaneously.

Commands

m.....	333
r.....	334
v2m.....	334
v2r.....	335

Properties

Name	Type	Unit	Description
[y-value]	double	mm/s	x position component (axis 1)
[x-value]	double	mm/s	y position component (axis 2)
[status]	int	-	acknowledge reply on v2m (p. 334) and v2r (p. 335)

m

Initiates an absolute **Vector move**.

Syntax

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

Examples

Example

	Command	Description
1:	25 60 m	Initiates Vector move which targets (axis 1; axis 2) position coordinate (25 mm; 60 mm).

r



Initiates a relative **Vector move**.

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

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

Examples

Example

	Command	Description
1:	-10 20 <i>r</i>	Initiates Vector move which covers (axis 1; axis 2) position coordinate distance (-10 mm; 20 mm).

v2m

Initiates an absolute **Vector move**. Same effect as *m* (p. 333), but returns an acknowledge reply at move start. The reply content is not specified at the time.

Syntax

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

Reply

[status]

Examples

Example

	Command	Description
1:	-5 -30 v2m	Initiates Vector move which targets (axis 1; axis 2) position coordinate (-5 mm; -30 mm).

v2r

Initiates a relative **Vector move**. Same effect as **r** (p. 334), but returns an acknowledge reply at move start. The reply content is not specified at the time.

Syntax

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

Reply

[status]

Examples

Example

	Command	Description
1:	100 40 v2r	Initiates Vector move which covers (axis 1; axis 2) position coordinate distance (100 mm; 40 mm).



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. 362)).

Commands

gi (getcurrent)337

Properties

Type: double

Unit: A

gi (getcurrent)

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

Syntax

{device} ***gi***

Reply

[absolutcurrent]

Examples

Example

	Command	Description
1:	2 <i>gi</i>	Returns <i>Absolute motor current</i> at axis 2.

Auto commutation



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.



1500 μ s has emerged to be a **time** parameter value that works in most cases, whereas the voltage parameter value is motor type specific. To be set to 0 when the controller is used as a stepper, DC or piezo motor drive (**Motor form** (p. 352) set to 0, 4, or 5).

Commands

getamc.....	339
setamc.....	340

Properties

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

getamc

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

Syntax

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

Reply

[*Value*]

Examples

Example

	Command	Description
1:	2 1 getamc	Returns Auto commutation time at axis 1.

setamc

Configures the **Auto commutation**.

Syntax

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

Examples

Example

	Command	Description
1:	4 1 2 setamc	Set Auto commutation voltage at axis 2 to 4 V.

Initial motor power state



Power state of motor after powerup sequence; has no impact on the execution of **Auto commutation** (p. 339) during powerup. If motor is configured to be dead after powerup, power can be recovered by execution of **Motor restart** (p. 374) .

Commands

setinimotorstate..... 341
getinimotorstate..... 342

Properties

Type: [int](#)

value	motor state after powerup
0	dead
1	live

setinimotorstate

Sets the **Initial motor power state**.

Syntax

[state] {device} **setinimotorstate**

Examples

Example

	Command	Description
1:	0 1 <i>setinimotorstate</i>	Set <i>Initial motor power state</i> at axis 1 to 0.
2:	<i>save</i>	Store parameters.

Result: After next powerup sequence, motor current at axis 1 will be zero.

getinimotorstate

Returns the current setting of the *Initial motor power state*.

Syntax

{device} *getinimotorstate*

Reply

[state]

Examples

Example

	Command	Description
1:	2 <i>getinimotorstate</i>	Return <i>Initial motor power state</i> at axis 2.

Motor brake



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. 374) .

output index	destination
1	reserved
2	reserved
3	controller open drain output
4	controller TTL I/O 1 *
5	controller TTL I/O 2 *

* hardware I/O, invariably configured as output



Neither does the brake control output function generally overrule the **Digital output state** (p. 168) function, nor vice versa. Any level settings will be executed. It is up to the user to avoid potential conflicts.



If brakes are needed at both axes, it is strongly recommended to control them by different outputs. Controlling both brakes by the same output is not prohibited, though.

Commands

getbrakefunc.....344

setbrakefunc.....345

Properties

<i>i</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 setting of the specified **Motor brake** configuration entry.

Syntax

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

Reply

[*Value*]

Examples

Example

	Command	Description
1:	1 2 getbrakefunc	Returns the associated output index of the Motor brake at axis 2.

setbrakefunc

Configures the **Motor brake** control function.

Syntax

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

Examples

Example 1

Task: Configure the open drain output to control a motor brake mounted at axis 1.

	Command	Description
1:	3 1 1 setbrakefunc	Assign the open drain output to the axis 1 brake function.
2:	1 0 1 setbrakefunc	Enable the motor brake control function at axis 1.

Motor current limit



Defines the motor current limit used for safety purposes. As soon as the actual **Absolute motor current** (p. 337) 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. 374) .

Commands

getmaxcurrent..... 346
setmaxcurrent.....347

Properties

Type: double
Unit: A

getmaxcurrent

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

Syntax

{device} **getmaxcurrent**

Reply

[maximum_current]

Examples

Example

	Command	Description
1:	2 <i>getmaxcurrent</i>	Returns <i>Motor current limit</i> at axis 2.

setmaxcurrent

Sets the *Motor current limit*.

Syntax

[maximum_current] {device} *setmaxcurrent*

Examples

Example

	Command	Description
1:	10 1 <i>setmaxcurrent</i>	Set <i>Motor current limit</i> at axis 1 to 10 A.

Motor current shift



storable

Motor to rotor angle offset. Value recalculated and replaced by every application of ***Auto commutation*** (p. 339) .



To be set to 0.0 with stepper and piezo motors.



See also introductory chapter "Motor commutation handling".

Commands

setMCShift.....	348
getMCShift.....	349

Properties

Type: [double](#)

setMCShift

Sets the ***Motor current shift*** value.



Regard that the value will be overwritten by every application of ***Auto commutation*** (p. 339) .

Syntax

[current_shift] {device} ***setMCShift***

Examples

	Command	Description
1:	0.25 1 setMCShift	Sets the Motor current shift value at axis 1 to 0.25 (i.e. a quarter of the motor phasewidth).

getMCShift

Returns current **Motor current shift** value.

Syntax

{device} **getMCShift**

Reply

[current_shift]

Examples

	Command	Description
1:	1 getMCShift	Returns current Motor current shift value at axis 1.

Motor dissipation



read-only

Current result of the average dissipation check.



See ***Motor parameters*** (p. 357) on how to parameterize the average dissipation check.

Commands

getdissipation..... 350

Properties

Type: [double](#)

getdissipation

Returns the actual ***Motor dissipation***.

Syntax

{device} ***getdissipation***

Reply

[dissipation]

Examples

Example

	Command	Description
1:	1 <i>getdissipation</i>	Returns <i>Motor dissipation</i> at axis 1.

Motor form



storable

Matches actuation of motor outputs to the connected motor.
Available motor forms are:

value	description
0	stepper motor
1	linear motor
2	alternating current (AC) motor
3	torque motor *
4	direct current (DC) motor
5	piezo motor

*not yet supported



A mismatched setting will prevent proper motor operation.



Note that if Hydra is equipped with a piezo type power stage, it will operate with piezo driven motors meeting the hardware manual specification *only*. In this case, the stored motor form setting will be ineffective; the controller will always replace it by the value 5. If Hydra is equipped with a standard type power stage, it will *not* operate with piezo driven motors. In this case, the controller will block the attempt to set the value to 5.

Commands

getmotor.....	353
setmotor.....	353

Properties

Type: [int](#)

getmotor

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

Syntax

{device} ***getmotor***

Reply

[motorform]

setmotor

Sets the ***Motor form***.



Since firmware version 3.000:

Note that a change of the ***Motor form*** setting will not take effect immediately. The requested configuration will only be effective on next ***Reset***.

Syntax

[motorform] {device} ***setmotor***

Examples

	Command	Description
1:	1 1 <i>setmotor</i>	Sets the <i>Motor form</i> at device 1 to <i>linear motor</i> .
2:	<i>save</i>	Stores the complete parameters set.

After next ***Reset***, device 1 will be configured to drive a linear motor.

Motor optimization stage



read-only

Information on how far the motor optimization preparation has progressed and what step to take next.

value	description
0	home position unknown - execution of Calibration move (p. 284) or Reference move (p. 327) required for detection
1	home position detected - motor to rotor angle offset specification pending - see Motor parameters (p. 357) , <i>optimized angle offset</i> entry
2	motor optimization armed - execution of Calibration move or Reference move required for execution
3	motor optimized

Commands

getmotoptst.....355

Properties

Type: [int](#)

getmotoptst

Returns the actual **Motor optimization stage**.

Syntax

{device} **getmotoptst**

Reply

[state]

Examples

Example

	Command	Description
1:	2 <i>getmotoptst</i>	Returns <i>Motor optimization stage</i> at axis 2.

Motor parameters



storable

Additional parameters for motor matching.

The *optimized angle offset* parameter is the optimum motor to rotor offset used with motor optimization, which usually has to be averaged out in advance from several **Auto commutation** (p. 339) runs with variation of the slide/rotor location and environmental conditions. It is designed to be entered manually. If the motor optimization home position (either calibration limit switch or reference mark location) has not yet been detected after powerup, value entrance will be denied. Otherwise, the entered value will automatically be corrected to match a coordinate system where the optimization home position is the origin, which makes motor optimization independent from the initial motor/slide location, regardless of the measurement system used. As soon as a valid value has been entered, motor optimization is ready to be armed.

The *phase compensation enabled*, *resistance*, and *inductance* parameters are used for the compensation of the motor voltage to current phase shift over motor speed (frequency). The higher the inductance and the lower the resistance setting, the more effect the compensation will have at a given motor speed. Depending on the motor characteristics, this can improve performance considerably. With either of the parameters set to 0, phase compensation will be ineffective.

The *max. average dissipation* and *dissipation averaging period* parameters are used for an average dissipation check over a user-definable period of time. Exceedance of *max. average dissipation* will constitute an emergency-off condition. With either of the parameters set to 0, the dissipation check will be inactive, and the average dissipation will be 0. With *resistance* set to a valid value, dissipation unit will be [W]. With *resistance* set to 0, dissipation unit will be [A²].



Note that motor optimization, as well as phase compensation, is useful with linear and AC motors only. See

also introductory chapter "Motor commutation handling" on motor optimization.



See ***Motion function*** (p. 297) for information on how to select the motor optimization home position and arm/disarm the motor optimization facility. Inquire ***Motor optimization stage*** (p. 355) for information if motor optimization facility is ready for *optimized angle offset* entry.



See ***Motor dissipation*** (p. 350) on how to inquire the current average dissipation.

Commands

setmotorpara..... 359
 getmotorpara..... 360

Properties

i	Name	Type	Unit	Description
1	[optimized angle offset]	double	-	ranging from 0.0 to 1.0
2	[phase compensation enabled]	int	-	0 = disabled 1 = active
3	[resistance]	double	Ohm	winding resistance
4	[inductance]	double	H	winding inductance
5	[max. average dissipation]	double	W or A²	emergency off threshold
6	[dissipation period]	double	s *	dissipation averaging period * 25 ms stepwidth

setmotorpara

Sets the specified ***Motor parameters*** entry.

Syntax

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

Examples

Example

Task: Set winding resistance of motor at axis 1 to 4 Ohms. Additionally activate average dissipation check with a period of 3 seconds; emergency-off threshold is to be 10 Watts.

	Command	Description
1:	4 3 1 setmotorpara	Sets motor winding <i>resistance</i> to 4 Ohms.
2:	10 5 1 setmotorpara	Sets <i>max. average dissipation</i> to 10 Watts.
3:	3 6 1 setmotorpara	Sets <i>dissipation check period</i> to 3 seconds.

getmotorpara

Returns the current setting of the specified **Motor parameters** entry.

Syntax

[*i*] {device} **getmotorpara**

Reply

[*Value*]

Examples

Example

	Command	Description
1:	4 2 <i>getmotorpara</i>	Returns <i>motor winding inductance</i> setting at axis 2.

Motor phase current



read-only

Measured current at 2 motor phases. See also ***Absolute motor current*** (p. 337) .

Commands

gc.....362

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]

Examples

Example

	Command	Description
1:	1 <i>gc</i>	Returns individual phase currents at motor connected to axis 1.

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

setphases.....	364
getphases.....	365

Properties

Type: [int](#)

setphases

Sets the **Motor phase number**.



Since firmware version 3.000:

Note that a change of the **Motor phase number** setting will not take effect immediately. The requested configuration will only be effective on next **Reset**.

Syntax

[phase] {device} **setphases**

Examples

	Command	Description
1:	2 1 <i>setphases</i>	Sets the <i>Motor phase number</i> at device 1 to 2.
2:	<i>save</i>	Stores the complete parameters set.

After next **Reset**, device 1 will be configured to drive a 2-phase motor.

getphases

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

Syntax

{device} *getphases*

Reply

[phase]

Examples

Example

	Command	Description
1:	2 <i>getphases</i>	Returns <i>Motor phase number</i> at axis 2.

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, DC or piezo motor drive (**Motor form** (p. 352) set to 2, 4, or 5).



A mismatched setting will lead to a mismatch between delivered and actually covered move distances in open loop mode, and unbalance the position controller in closed loop mode.

Commands

setpolepairs.....	366
getpolepairs.....	367

Properties

Type: [int](#)

setpolepairs

Sets the **Motor pole pairs**.



Since firmware version 3.000:

Note that a change of the **Motor pole pairs** setting will not take effect immediately. The requested configuration will only be effective on next **Reset**.

Syntax

[polepairs] {device} **setpolepairs**

Examples

	Command	Description
1:	50 2 setpolepairs	Sets the Motor pole pairs at device 2 to 50.
2:	save	Stores the complete parameters set.

After next **Reset**, device 2 will be configured to drive a 50-polepair (i.e. 200-step) motor.

getpolepairs

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

Syntax

{device} **getpolepairs**

Reply

[polepairs]

Examples

Example

	Command	Description
1:	1 getpolepairs	Returns Motor pole pairs at axis 1.

Motor voltage gradient



storable

Velocity gradient of motor phase voltage amplitude. Predominantly significant during faster motion.



To be set to 0.0 with linear, AC, and DC motors in closed loop mode.



Caution - to be handled with care! Parameter value is not compatible with previous SMC controllers! In order to prevent motor and controller hardware from sustaining damage, the respective maximum current ratings have to be met. Use **gc** (p. 362) and **gi** (p. 337) for current observation.

Commands

setumotgrad.....	368
getumotgrad.....	369

Properties

Type: double

setumotgrad

Sets the **Motor voltage gradient**.

Syntax

[voltage_gradient] {device} **setumotgrad**

Examples

Example

	Command	Description
1:	1.2 1 <i>setumotgrad</i>	Set <i>Motor voltage gradient</i> at axis 1 to 1.2.

getumotgrad

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

Syntax

{device} *getumotgrad*

Reply

[voltage_gradient]

Examples

Example

	Command	Description
1:	1 <i>getumotgrad</i>	Returns <i>Motor voltage gradient</i> at axis 1.

Motor voltage minimum



Base rate of motor phase voltage amplitude with stepper and piezo motors. Predominantly significant during standstill or slow motion.



To be left at factory setting with linear, AC, and DC motors.



Caution - to be handled with care! Not compatible with previous SMC controllers where the parameter unit was mV! In order to prevent motor and controller hardware from sustaining damage, the respective maximum current ratings have to be met. Use **gc** (p. 362) and **gi** (p. 337) for current observation.

Commands

getumotmin.....	370
setumotmin.....	371

Properties

Type: double
Unit: V

getumotmin

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

Syntax

{device} *getumotmin*

Reply

[minimum_voltage]

Examples

Example

	Command	Description
1:	2 <i>getumotmin</i>	Returns <i>Motor voltage minimum</i> at axis 2.

setumotmin

Sets the *Motor voltage minimum*.

Syntax

[minimum_voltage] {device} *setumotmin*

Examples

Example

	Command	Description
1:	2.5 1 <i>setumotmin</i>	Set <i>Motor voltage minimum</i> at axis 1 to 2.5 V.

Safety functions

Motor powerdown



Powerdown of motor to zero current. Same impact as a powerdown caused by a **Machine error** (p. 116) . Normal operation can be recovered by execution of **Motor restart** (p. 374) .

Commands

motoroff..... 373

motoroff

Initiates **Motor powerdown**.

Syntax

{device} **motoroff**

Examples

Example

	Command	Description
1:	2 motoroff	Initiate Motor powerdown at axis 2.

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..... 374

init

Upon execution, the following will happen:

- Motor powerup according to actual motor parameters (**Motor voltage minimum** (p. 370) , **Motor voltage gradient** (p. 368)). Unwanted motion is reduced to a minimum.
- Reactivation of positioning controller according to currently requested **Positioning control mode**(p.395) .

Syntax

{device} *init*

Examples

Example

	Command	Description
1:	1 <i>init</i>	Initiate Motor restart at axis 1.



Sensoric

Before use of position sensors, please take at a look at introductory chapter "Position sensor routing".

Scale period

Scale period of position sensor assigned to specified axis.



Written to **Sensor cache** (p. 383); not permanently stored by **Configuration storage** (p. 242) .



Must match the physical sensor period and effective axis motion direction. A mismatched setting will lead to improper position controlled operation.



Immediately effective; should not be changed during position controlled operation. See **Positioning control mode** (p. 395) on how to open and close the position control loop.



Can be set with a negative sign in order to invert the count direction of position measurement.

Commands

getclperiod.....	377
setclperiod.....	378

Properties

Type: double
Unit: mm

getclperiod

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

Syntax

{device} *getclperiod*

Reply

[Period]

Examples

Example

	Command	Description
1:	2 <i>getclperiod</i>	Returns scale period of position sensor assigned to axis 2.

setclperiod

Sets the *Scale period*.

Syntax

[Period] {device} *setclperiod*

Examples

Example

Preconditions: Any subdevice of sensor interface 3 is assigned to axis 1.

Task: Set **Scale period** at sensor interface 3, subdevice 0 to 20 μm .

	Command	Description
1:	0 1 setcloop	Open position control loop at axis 1.
2:	0.02 1 setclperiod	Set Scale period at axis 1 to 20 μm . Value is written to Sensor cache for sensor interface 3.
3:	3 savecache	Store Sensor cache of interface 3 to respective EEPROM.

Result: After next controller reset, **Scale period** will be 20 μm .

Sensor amplitudes



Current values of all tracks at the sensor interface connected to the specified sensor port in sine/cosine track pairs.



The *device* entry does not select a motor axis, but the target sensor port.

Commands

S..... 380
SC..... 381

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

Returns the ***Sensor amplitudes***.



Use the index of the target sensor port to specify the *device*. The raw values of all tracks of all sensors connected to that sensor port will be returned, regardless what motor axes they may be assigned to. The command will always return the maximum possible number of tracks; unused tracks may deliver invalid results.

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

Example

Command		Description
1:	3 S	Returns Sensor amplitudes at Star interface port 3.

SC

Same as **S**_(p. 380), but returns amplitude corrected values instead of raw values.

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]

Examples

Example

Command		Description
1:	2 SC	Returns amplitude corrected Sensor amplitudes at Star interface port 2.

Sensor cache



not storable

Cache holding data read out of a specified position sensor interface EEPROM during powerup sequence. As for user accessible data, contains **Scale period** (p. 377) .

Commands

savecache..... 383

Properties

Type: [int](#)

savecache

Stores specified **Sensor cache** content in respective sensor interface EEPROM and returns *result* value.

result	description
0	storage complete
-1	storage failed



Note that the storage process will

- halt the Venus interpreter (no processing of further Venus commands at any axis during execution)
- take several seconds to execute

Never switch off the controller during parameter storage to avoid data loss! The proper way to check if the controller has completed storage is to repeatedly transmit an inquiry command such as *nst* and wait until the Venus

interpreter starts replying to the inquiries. Then the storage process has finished, and the controller may be reset or switched off.



Use the index of the target sensor port to specify the *device*.

Syntax

{device} **savecache**

Reply

[result]

Examples

Example

	Command	Description
1:	2 savecache	Stores Sensor cache content for sensor interface 2 in respective EEPROM.

Sensor temperature



read-only

Sensor temperature delivered at specified Star interface port.



The *device* entry does not select a motor axis, but the target sensor port.

Commands

gettemp..... 385

Properties

Type: double

Unit: °C

gettemp

Returns the actual **Sensor temperature**.

Syntax

{device} **gettemp**

Reply

[temperature]

Examples

Example

	Command	Description
1:	3 <i>gettemp</i>	Returns <i>Sensor temperature</i> at Star interface port 3.



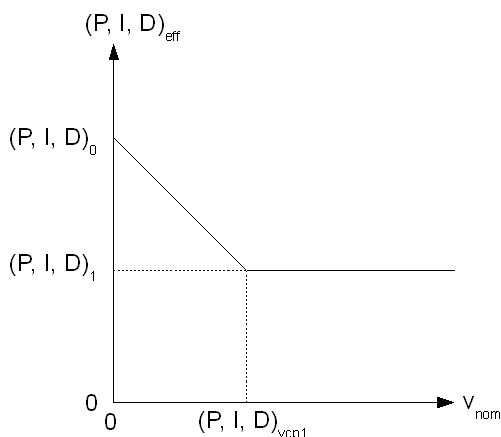
Servo

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. 407) settings are still effective, except for the P, I and D factor settings being replaced by the values calculated here.



Note that when control point velocity is set to 0 in dynamic mode, the respective halt setting will constantly be effective throughout velocity range.

See **Positioning control mode** (p. 395) for positioning controller mode selection.

Commands

getadaptive.....	390
setadaptive.....	391

Properties

i	Name	Type	Unit	Description
1	[P0]	double	-	proportional (P) coefficient at standstill
2	[I0]	double	-	integral (I) coefficient at standstill
3	[D0]	double	-	differential (D) coefficient at standstill
4	[Pvcp1]	double	mm/s	control point velocity for P coefficient
5	[Ivcp1]	double	mm/s	control point velocity for I coefficient
6	[Dvcp1]	double	mm/s	control point velocity for D coefficient
7	[P1]	double	-	final value of P coefficient
8	[I1]	double	-	final value of I coefficient
9	[D1]	double	-	final value of D coefficient

getadaptive

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

Syntax

[i] {device} **getadaptive**

Reply

[*Value*]

Examples

Example

	Command	Description
1:	5 2 <i>getadaptive</i>	Returns integral portion control point velocity of positioning controller at axis 2.

setadaptive

Configures the *Adaptive positioning control*.

Syntax

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

Examples

Example

	Command	Description
1:	0.01 8 2 <i>setadaptive</i>	Set integral portion final value of positioning controller axis 2 to 0.01.

Positioning control freeze

Mode in which the position control is inactive, but - as opposed to open loop mode - remains in the position processing chain, so "freezing" as well as "unfreezing" can be done during normal operation without loss of position. This can be useful with stepper motors to avoid unwanted fluctuation caused by position control during standstill, e.g. with imaging applications.



This feature may also be used in conjunction with the **Action command** feature (internal condition 3) (see introductory chapter "Introduction to digital I/O processing").

Commands

getfreeze.....	392
freeze.....	393

Properties

Type: [int](#)

getfreeze

Returns current **Positioning control freeze** mode activation state.

Syntax

{device} **getfreeze**

Reply

[state]

freeze

Puts the positioning controller into or out of ***Positioning control freeze*** mode.

Syntax

[state] {device} ***freeze***

Examples

Example 1

Direct axis 1 operation.

	Command	Description
1:	1 1 <i>freeze</i>	Activates <i>Positioning control freeze</i> at axis 1.

Example 2

Axis 1 operation, utilizing **Action command** feature with internal condition 3, presuming all other internal conditions be unused. We also presume **Target window** (p. 412) and **Time on target** (p. 414) are set to appropriate values.

	Command	Description
1:	"1 1 freeze " 0 3 1 setactioncmdint	Assign activation of axis 1 Positioning control freeze to primary action command of "axis 1 resting on target" condition.
2:	"0 1 freeze " 1 3 1 setactioncmdint	Assign deactivation of axis 1 Positioning control freeze to secondary action command of "axis 1 resting on target" condition.
3:	0 1 seteventpolint	Set "axis 1 resting on target" polarity bit (along with all other polarity register bits) to active high.
4:	4 1 seteventmodeint	Set "axis 1 resting on target" mode bit to key function (all other register bits being set to default).
5:	4 setinitactmaskint (p. 218)	1 Activate the initial event generation at "axis 1 resting on target" condition (and deactivate at all other internal conditions).

After **Configuration storage (Controller)** and **Reset**, Axis 1 will automatically activate **Positioning control freeze** whenever "axis 1 resting on target" condition is satisfied, and vice versa.

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. 407) 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 adaptive 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. 388).

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

Commands

setcloop..... 396
getcloop..... 396

Properties

Type: [int](#)

setcloop



Sets the **Positioning control mode**.

Since firmware version 3.000:

Note that a change of the mode setting will not take effect immediately. The actual activation of the requested mode is delayed until

- next execution of **Motor restart** (p. 374) *or*
- next **Reset** (p. 121) (when preceded by parameter storage)

Syntax

[state] {device} **setcloop**

Examples

	Command	Description
1:	2 1 setcloop	Sets the device 1 closed loop mode to 2.
2:	1 init (p. 374)	Rebalance position control, close loop and activate adaptive mode at device 1.

The positioning regulator will operate in adaptive mode.

getcloop

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

Syntax

{device} **getcloop**

Reply

[state]

Examples

Example

Command		Description
1:	1 <i>getcloop</i>	Returns <i>Positioning control mode</i> at axis 1.

Sensor assignment



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

setassignment.....	399
getassignment.....	404

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. 396)
- 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. 121) 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

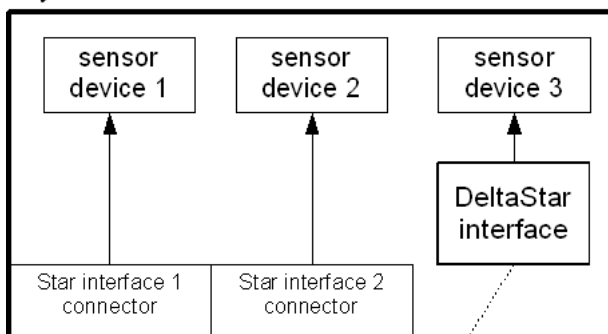
[sensor subdevice] [sensor device] {device} ***setassignment***

Examples

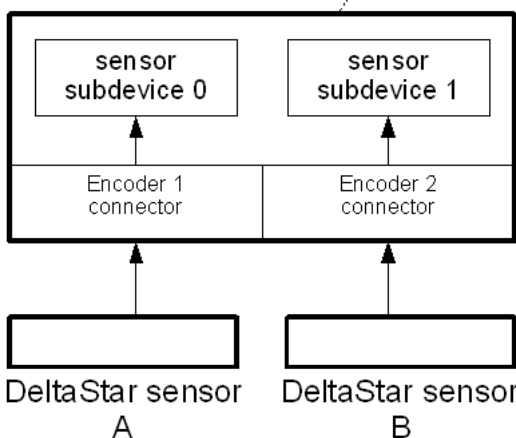
Example 1

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.

Hydra controller with built-in Star interface



DeltaStar interface

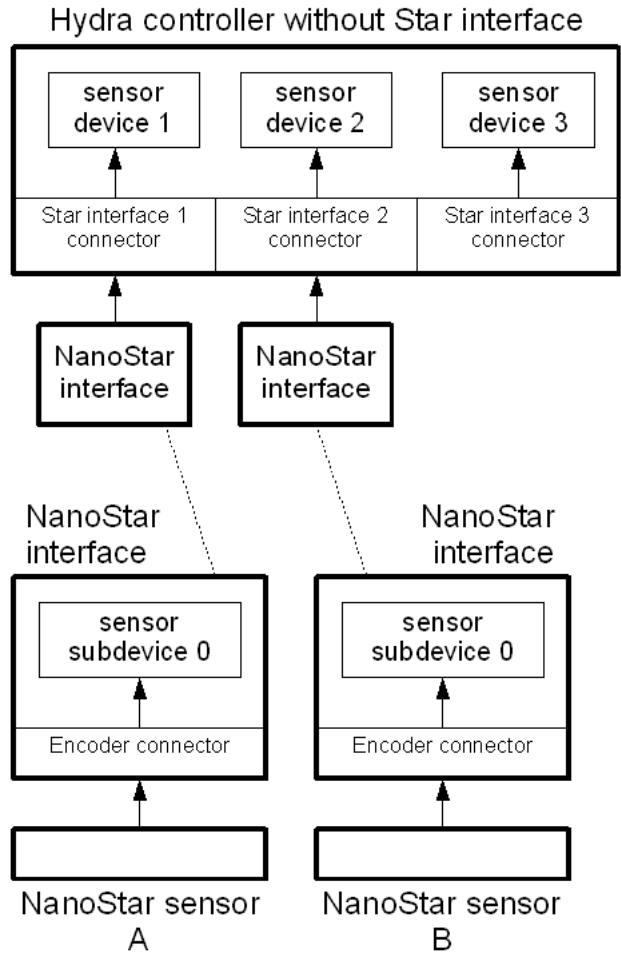


Task: The sensor A is to be assigned to axis device 1, sensor B to axis device 2.

	Command	Description
1:	0 3 1 <i>setassignment</i>	Assign sensor A (subdevice 0, device 3) to axis 1.
2:	1 3 2 <i>setassignment</i>	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 <i>setassignment</i>	Assign sensor A (subdevice 0, device 1) to axis 1.
2:	0 2 2 <i>setassignment</i>	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]

Examples

Example

	Command	Description
1:	1 <i>getassignment</i>	Returns <i>Sensor assignment</i> at axis 1.

Sensor status



read-only

Actual status register of position sensor assigned to specified axis. Basically an error register; with all bits cleared, sensor is working properly. With all bits set, status is undefined.

Bit number	description
0	general error
1	low amplitude
2	low quality
3	no sensor assigned
4	no sensor interface connected
5	reserved
6	position out of range
7	illegal subdevice

Commands

getsensorstatus.....	405
sensorreconnect.....	406

Properties

Type: [int](#)

getsensorstatus

Returns the actual **Sensor status**.

Syntax

{device} **getsensorstatus**

Reply

[sensor status]

Examples

Example

Command		Description
1:	2 <i>getsensorstatus</i>	Returns <i>Sensor status</i> at axis 2.

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. 374)).



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. 388) 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. 346) 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 discontinuity 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:

$$vI_{Max_{eff}} = \text{max I vel} * I_{eff} / I \text{ factor}$$



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



The ***regulation output filter time*** is to be applied to especially oscillation susceptible systems only. Should usually be 0 or left at factory setting to maintain optimum performance.

For activation and deactivation of position regulation in either mode (standard or adaptive), see **Positioning control mode** (p. 395) .

Commands

setsp.....	410
getsp.....	410

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) - <i>void since firmware rev. 3.000</i>
6	[load distance]	double	mm	<i>up to firmware rev. 2.411:</i> motor load distance limit (linear motor drive only) <i>since firmware rev. 3.000:</i> overall output limitation
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)
9	[D threshold]	double	mm	absolute minimum position error to generate differential output
10	[filter time]	double	s	regulation output filter time

setsp

Configures the **Servo control**.

Syntax

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

Examples

Example

Command		Description
1:	5 7 1 setsp	Set maximum position aberration at axis 1 to 5 mm.

getsp

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

Syntax

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

Reply

[*Value*]

Examples

Example

	Command	Description
1:	3 1 <i>getsp</i>	Returns differential portion of positioning controller at axis 1.

Target window



Width of entrance and exit windows for the position settlement indication or in-window bit (see **Axis status** (p. 417) 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. 414), 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

getclwindow.....	412
setclwindow.....	413

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]

Examples

Example

	Command	Description
1:	1 <i>getclwindow</i>	Returns <i>Target window</i> at axis 1.

setclwindow

Sets the *Target window*.

Syntax

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

Examples

Example

	Command	Description
1:	.01 .05 <i>setclwindow</i>	1 Set <i>Target window</i> at axis 1 to 10 μm (entrance window) and 50 μm (exit window).

Time on target



storable

Position settling time for the in-window indication feature (see **Target window** (p. 412)).

Commands

setclwintime.....	414
getclwintime.....	415

Properties

Type: double
Unit: s

setclwintime

Sets the **Time on target**.

Syntax

[value] {device} **setclwintime**

Examples

Example

	Command	Description
1:	0.1 2 setclwintime	Set Time on target at axis 2 to 0.1 s.

getclwintime

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

Syntax

{device} *getclwintime*

Reply

[value]

Examples

Example

Command		Description
1:	2 <i>getclwintime</i>	Returns <i>Time on target</i> at axis 2.

Status and position

Axis status



read-only

Actual status of the axis device. The flags of the 16 bit status word in detail:

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...14	reserved
15	invalid status - reset required **

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

Indication of emergency conditions:

Current engagement of the emergency switch is shown by simultaneous activation of bits 9, 8 and 7:

bit 15	bit 9	bit 8	bit 7	bit 2	bit 0
		1	1	1				

Recent engagement of the emergency switch is shown by bits 8 and 7 (as long as the motor has not been repowered):

bit 15	bit 9	bit 8	bit 7	bit 2	bit 0
		0	1	1				

Emergency condition due to a machine error is shown by simultaneous activation of bits 8 and 2:

bit 15	bit 9	bit 8	bit 7	bit 2	bit 0
			1			1		

If the machine error cause has been removed and all machine errors entries have been popped off the machine error stack, bit 2 will be cleared:

bit 15	bit 9	bit 8	bit 7	bit 2	bit 0
			1			0		

As soon as the respective emergency condition has been removed, the **Motor restart** (p. 374) 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). Bits 8 and 7 will be cleared:

bit 15	bit 9	bit 8	bit 7	bit 2	bit 0
			0	0		0		

The status register is supplemented by a 16 bit machine error extension with **est** (p. 420) and **ast** (p. 421).



For watching the complete system without decoding each singular axis, see **Controller status** (p. 425) .

Commands

est.....	420
nst (nstatus)	420
ast.....	421

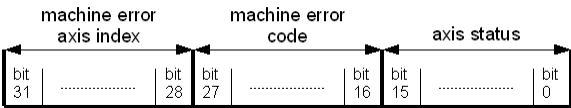
Properties

Type: **int**

est

Like *nst*, delivers the standard **Axis status** in the lower 16 bit portion, but with an axis-independent machine error extension within the upper 16 bit portion. For this purpose, the last entry at the **Machine error** stack is read (*not popped*) and encoded into the extension as follows:

- machine error code (lower 12 bits)
- axis device index (upper 4 bits)



Syntax

{device} **est**

Reply

[status]

Examples

Example

	Command	Description
1:	1 est	Returns extended Axis status at axis 1.

nst (nstatus)

Returns the actual **Axis status** as is.

Syntax

{device} *nst*

Reply

[status]

Examples

Example

	Command	Description
1:	1 <i>nst</i>	Returns <i>Axis status</i> at axis 1.

ast

Same as *est* (p. 420), but synchronized to move execution. Waits until currently running move has finished before status and machine error inquiry; then same function as *est*. Execution of *Ctrl+C* (supplemented by CR/LF with TCP/IP connection) will immediately quit the running move and force status reply.

Syntax

{device} *ast*

Reply

[status]

Examples

Example

	Command	Description
1:	2 <i>ast</i>	Waits until axis 2 is in standstill. Then returns extended <i>Axis status</i> .

Controller position



read-only

Integrated **Device position** (p. 429) of both axis devices. The single positions displayed conform the respective **Position display selection** (p. 431) settings.

Commands

p..... 423

Properties

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

p

Returns the actual **Controller position**.

Syntax

p

Reply

[position X] [position Y]

Examples

Example

	Command	Description
1:	<i>p</i>	Returns current <i>Controller position.</i>

Controller status



read-only

Partially a controller device status indication, partially a bitwise linkage of all **Axis status** (p. 417) 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. 425) registers.

Commands

st (status) 427

Properties

Type: [int](#)

st (status)

Returns the actual **Controller status**.

Syntax

st

Reply

[status]

Examples

Example

	Command	Description
1:	<i>st</i>	Returns <i>Controller status</i> .

Device position



read-only

Actual device position, displayed selectively as the nominal or measured position (see ***Position display selection*** (p. 431)).

Commands

np..... 429

Properties

Type: double

Unit: mm

np

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

Syntax

{device} ***np***

Reply

[position]

Examples

Example

	Command	Description
1:	2 <i>np</i>	Return <i>Device position</i> at axis 2.

Position display selection



storable

Selection of position displayed as **Device position** (p. 429) .

Commands

setselpos.....	431
getselpos.....	432

Properties

Type: [int](#)

value	selection
0	nominal position
1	measured position

setselpos

Sets the **Position display selection**.

Syntax

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

Examples

Example

	Command	Description
1:	1 2 <i>setselpos</i>	Set <i>Position display selection</i> at axis 2 to display of measured position.

getselpos

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

Syntax

{device} *getselpos*

Reply

[position source]

Examples

Example

	Command	Description
1:	1 <i>getselpos</i>	Returns <i>Position display selection</i> at axis 1.

Switches and reference mark

Reference configuration



Defines the mode of operation for reference position mark detection. The setting takes effect upon execution of ***refmove*** (p. 320) and ***nrefmove*** (p. 328).

Commands

getref..... 434
setref..... 435

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]

Examples

Example

	Command	Description
1:	2 <i>getref</i>	Returns <i>Reference configuration</i> at axis 2.

setref

Sets *Reference configuration*.

Syntax

[config] {device} *setref*

Examples

	Command	Description
1:	1 1 <i>setref</i>	Enables device 1 reference position mark detection on falling edges.

Upon execution of *refmove* (p. 320) or 1 *nrefmove* (p. 328), 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. 327) only. Once set, they stay set until next execution of **Reference move**.

Commands

getrefst..... 436

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]

Examples

Example

	Command	Description
1:	1 <i>getrefst</i>	Returns <i>Reference status</i> at axis 1.

Switch configuration



storable

Bit-coded configuration of limit switches. Polarity options are normally open (NO) or normally closed (NC). Masking a limit switch will suppress move events (such as stop/reverse) and **Hardware limits** (p. 291) alteration due to switch engagement.

Bit number	description
0	polarity (0 = NO, 1 = NC)
1	mask (0 = enabled, 1 = masked)



Calibration move (p. 284) and **Range measure move** (p. 306) will not work properly when the respective limit switch is disabled. See **Motion function** (p. 297) on how to disable those move commands.



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



Masking a limit switch does not affect switch engagement indication via **Switch status** (p. 441) .

Commands

setsw.....	439
getsw.....	439

Properties

i	Name	Type
0	[calibration sw.]	int
1	[range measure sw.]	int

setsw

Sets the **Switch configuration**.

Syntax

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

Examples

Example

	Command	Description
1:	0 1 2 setsw	Set Switch configuration of range measure switch at axis 2 to normally open. Enable range measure switch for move event generation and limit alteration.

getsw

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

Syntax

{device} **getsw**

Reply

[calibration sw.] [range measure sw.]

Examples

Example

	Command	Description
1:	1 <i>getsw</i>	Returns <i>Switch configuration</i> of calibration switch at axis 1.

Switch status



not storable

Engagement states of limit switches.

Commands

getswst..... 441

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.]

Examples

Example

	Command	Description
1:	1 <i>getswst</i>	Returns <i>Switch status</i> at axis 1.



Trigger

Before use of the trigger functions, please take a look at introductory chapter "Introduction to trigger function".

Trigger capture buffer size



not storable

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

Commands

settrinsize.....	445
gettrinsize.....	446

Properties

Type: [int](#)

settrinsize

Sets the *Trigger capture buffer size*.

Syntax

[size] {device} **settrinsize**

Examples

Example

	Command	Description
1:	100 1 settrinsize	Configure axis 1 position capture buffer to take up to 100 position values.

gettrinsize

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

Syntax

{device} *gettrinsize*

Reply

[size]

Examples

Example

	Command	Description
1:	1 <i>gettrinsize</i>	Return <i>Trigger capture buffer size</i> at axis 1.

Trigger capture index



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

gettrindex..... 447

Properties

Type: [int](#)

gettrindex

Returns the current *Trigger capture index*.

Syntax

{device} *gettrindex*

Reply

[index]

Examples

Example

	Command	Description
1:	2 <i>gettrindex</i>	Return <i>Trigger capture index</i> at axis 2.

Trigger capture mode



Enable state of the trigger capture function.

value	capture function
0	off
1	on

Commands

settrin.....	448
gettrin.....	449

Properties

Type: [int](#)

settrin

Sets the *Trigger capture mode*.

Syntax

[enabled] {device} **settrin**

Examples

Example

	Command	Description
1:	1 settrin	Enable position capturing at axis 1.

gettrin

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

Syntax

{device} *gettrin*

Reply

[enabled]

Examples

Example

	Command	Description
1:	1 <i>gettrin</i>	Return <i>Trigger capture mode</i> at axis 1.

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

gettrinpole.....	450
settrinpole.....	451

Properties

Type: [int](#)

gettrinpole

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

Syntax

{device} *gettrinpole*

Reply

[input polarity]

Examples

Example

	Command	Description
1:	1 <i>gettrinp</i> ol	Return <i>Trigger capture polarity</i> at axis 1.

settrinp

Sets the *Trigger capture polarity*.

Syntax

[input polarity] {device} *settrinp*ol

Examples

Example

	Command	Description
1:	1 2 <i>settrinp</i> ol	Make position capturing at axis 2 sensitive to falling edges at trigger input.

Trigger capture position



not storable

Position value latched by trigger capture function at a given index (index starting at 0 for 1st captured position value).

Commands

gettrinpos.....452

Properties

Name	Type
[index]	int
[position]	double

gettrinpos

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

Syntax

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

Reply

[position]

Examples

Example

	Command	Description
1:	7 2 <i>gettrinpos</i>	Returns 8th position captured during last capture sequence at axis 2.

Trigger capture position file

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. 455) command. With a TFTP command line tool installed on the host PC, the contents of the file can then 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. 452), this saves communication time, and is particularly useful when a record contains a large number of captured values.



See also introductory chapter "Introduction to command chain function" for general information on how CAN handwheel pushbuttons are processed.

Commands

filetrinpos.....	455
------------------	-----

Properties

Type: [int](#)

filetrnpos

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} *filetrnpos*

Reply

[error code]

Examples

Example

	Command	Description
1:	2 <i>filetrnpos</i>	Write record of last position capture sequence at axis 2 to Hydra RAM file system.

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. 467) instead.

Commands

settrdelay.....	456
gettrdelay.....	456

Properties

Type: double

Unit: 0.5 μ s ticks

settrdelay

Sets the ***Trigger delay***.

Syntax

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

gettrdelay

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

Syntax

{device} ***gettrdelay***

Reply

[delay]

Trigger event setup



not storable

Trigger event generator setup, valid with **equidistant mode** (see **Trigger mode** (p. 460)).

Commands

settrpara.....	458
gettrpara.....	459

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** and arms the trigger. Void if **Trigger mode** (p. 460) is not currently set to **equidistant mode**.



Note that at the time of arming, the slide or rotor position has to be consistent with the specified temporal trigger position order, i.e. the first trigger position has to be located between the current position and the last trigger position.

Syntax

{device} **settrpara**

Examples

Example

	Command	Description
1:	10 20 11 2 <i>settrpara</i>	Arm trigger for a pulse sequence of 11 equidistant pulses between axis 2 coordinates 10 mm (start) and 20 mm (stop).

gettrpara

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

Syntax

{device} *gettrpara*

Reply

[start position] [stop position] [number]

Examples

Example

	Command	Description
1:	2 <i>gettrpara</i>	Return <i>Trigger event setup</i> at axis 2.

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 s_1
- the stop position s_n

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



See **Trigger event setup** (p. 458) for equidistant trigger event configuration.



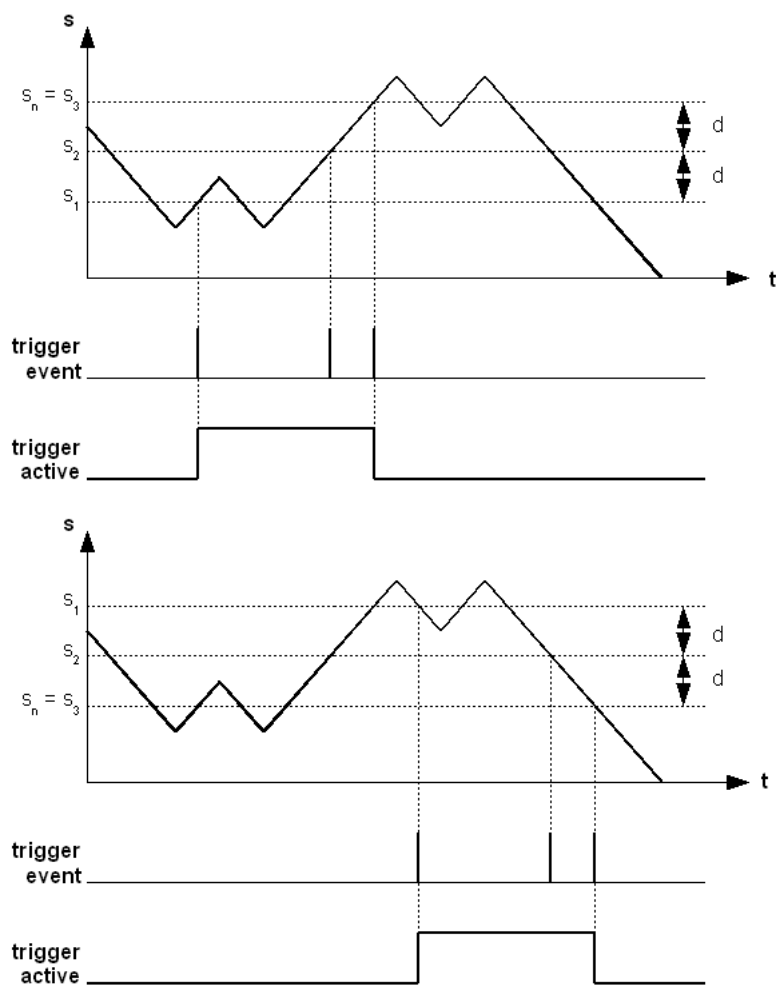
For query whether a trigger sequence is currently active (running), see **Trigger status** (p. 475).



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.



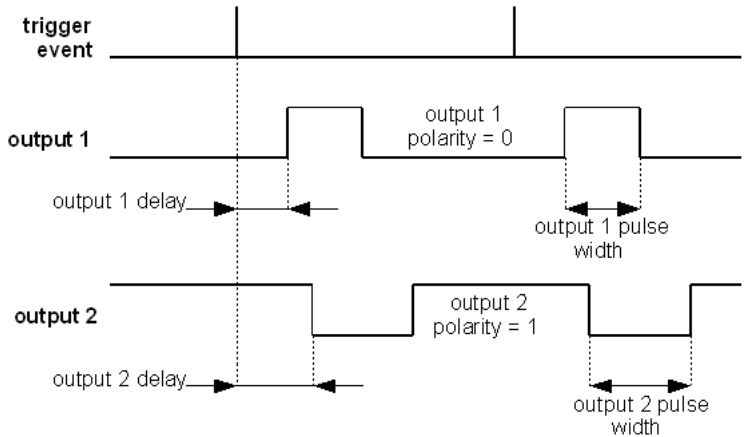
Note that as soon as a trigger sequence has finished or been stopped before reaching the final trigger event, the trigger function has to be reenabled to start the next one.



Equidistant trigger mode principle

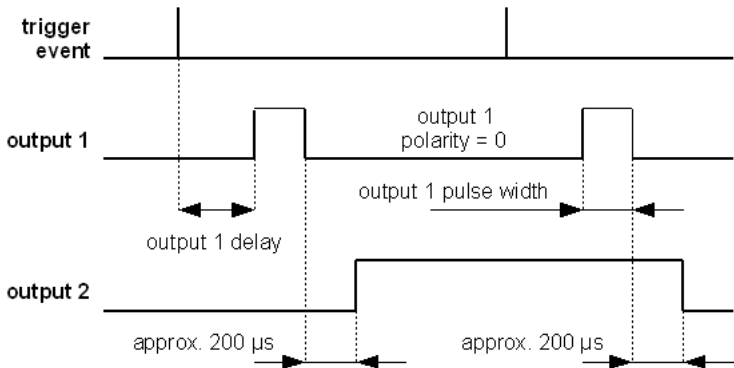
With the **continuous mode**, the trigger event generator puts out continuous asynchronous events at a constant frequency of approx. 500 Hz. It does not need further parameterization.

With the **standard mode**, both outputs operate equally, individually configurable regarding **Trigger output delay** (p. 467) , **Trigger output pulse width** (p. 471) and **Trigger output polarity** (p. 469) .



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

settr.....	465
gettr.....	466

Properties

Type: [int](#)

settr

Sets the *Trigger mode*.

Syntax

[mode] {device} **settr**

Examples

Example

	Command	Description
1:	1 2 <i>settr</i>	Set <i>Trigger mode</i> to equidistant/standard mode for axis 2 trigger sequences.

gettr

Returns the current setting of the *Trigger mode*.

Syntax

{device} *gettr*

Reply

[mode]

Examples

Example

	Command	Description
1:	2 <i>gettr</i>	Return <i>Trigger mode</i> at axis 2.

Trigger output delay



not storable

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

Commands

settroutdelay.....467
gettroutdelay.....468

Properties

<i>i</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**

Examples

Example

	Command	Description
1:	10 1 2 <i>settroutdelay</i>	Set <i>Trigger output delay</i> at output 1 to 10 μ s for axis 2 trigger sequences.

gettroutdelay

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

Syntax

[*i*] {device} *gettroutdelay*

Reply

[*Value*]

Examples

Example

	Command	Description
1:	2 1 <i>gettroutdelay</i>	Returns <i>Trigger output delay</i> of output 1 for axis 2 trigger sequences.

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

gettroutpol.....	469
settroutpol.....	470

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*]

Examples

Example

	Command	Description
1:	1 1 <i>settroutpol</i>	Returns <i>Trigger output polarity</i> of output 1 for axis 1 trigger sequences.

settroutpol

Sets the *Trigger output polarity*.

Syntax

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

Examples

Example

	Command	Description
1:	0 2 1 <i>settroutpol</i>	Set <i>Trigger output polarity</i> at output 2 to active high for axis 1 trigger sequences.

Trigger output pulse width



not storable

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

Commands

settroutpw.....	471
gettroutpw.....	472

Properties

<i>i</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**

Examples

Example

	Command	Description
1:	5 2 2 <i>settroutpw</i>	Set <i>Trigger output pulse width</i> at output 2 to 5 μ s for axis 2 trigger sequences.

gettroutpw

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

Syntax

[*i*] {device} *gettroutpw*

Reply

[*Value*]

Examples

Example

	Command	Description
1:	2 1 <i>gettroutpw</i>	Returns <i>Trigger output pulse width</i> of output 2 for axis 1 trigger sequences.

Trigger pulse width



not storable

Unitary width of pulses at all trigger signal outputs.



For compatibility purpose only - not for further use. Use ***Trigger output pulse width*** (p. 471) instead.

Commands

gettrwidth.....	473
settrwidth.....	474

Properties

Type: double
Unit: 0.5 μ s ticks

gettrwidth

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

Syntax

{device} ***gettrwidth***

Reply

[width]

settrwidth

Sets the *Trigger pulse width*.

Syntax

[width] {device} **settrwidth**

Trigger status



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. 460) . For trigger event configuration, see ***Trigger event setup*** (p. 458) .

Commands

gettrst..... 475

Properties

Type: [int](#)

gettrst

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

Syntax

{device} ***gettrst***

Reply

[status]

Examples

Example

	Command	Description
1:	2 <i>gettrst</i>	Return <i>Trigger status</i> at axis 2.

State Reference

1. Absolute motor current.....	337	39. CPU temperature.....	112
2. Absolute move.....	314	40. Device class.....	113
3. Acceleration.....	135	41. Device count.....	115
4. Action command - CAN handwheel 1 (primary).....	147	42. Device position.....	429
5. Action command - CAN handwheel 1 (secondary).....	149	43. Digital output state.....	168
6. Action command - CAN joystick 1 (primary).....	151	44. Emergency switch.....	170
7. Action command - CAN joystick 1 (secondary).....	153	45. Error decoder.....	247
8. Action command - controller (primary).....	155	46. Event detect register.....	172
9. Action command - controller (secondary).....	157	47. Event mode entry.....	174
10. Action command entry.....	159	48. Event mode register - CAN handwheel 1.....	177
11. Action state register.....	165	49. Event mode register - CAN joystick 1.....	178
12. Adaptive positioning control.....	388	50. Event mode register - controller...	179
13. Auto commutation.....	339	51. Event polarity entry.....	180
14. Axis status.....	417	52. Event polarity register - CAN handwheel 1.....	183
15. Calibration move.....	284	53. Event polarity register - CAN joystick 1.....	184
16. Calibration switch distance.....	287	54. Event polarity register - controller	185
17. Calibration velocity.....	289	55. Event state.....	186
18. Clock and direction function.....	316	56. Hardware limits.....	291
19. Clock and direction width.....	318	57. Initial action mask entry.....	188
20. Command chain 1.....	227	58. Initial action mask register - CAN handwheel 1.....	191
21. Command chain 2.....	228	59. Initial action mask register - CAN joystick 1.....	192
22. Command chain 3.....	229	60. Initial action mask register - controller.....	193
23. Command chain 4.....	230	61. Initial limits.....	293
24. Command chain 5.....	231	62. Initial motor power state.....	341
25. Command chain 6.....	232	63. Internal action command (primary).....	194
26. Command chain 7.....	233	64. Internal action command (secondary).....	196
27. Command chain 8.....	234	65. Internal action command entry.....	198
28. Command chain entry.....	235	66. Internal action state register.....	203
29. Command chain execution.....	237	67. Internal event detect register.....	207
30. Command chain index entry.....	239	68. Internal event mode register.....	209
31. Configuration storage (Controller)	242	69. Internal event polarity register.....	212
32. Configuration storage (Axis)	245	70. Internal event state.....	215
33. Controller identification.....	108	71. Internal initial action mask register.....	217
34. Controller name.....	110	72. Interpreter error (Controller)	249
35. Controller position.....	423	73. Interpreter error (Axis)	251
36. Controller reference move.....	320		
37. Controller status.....	425		
38. Controller version.....	111		

74. Machine.....	279	119. Reference velocity.....	310
75. Machine error.....	116	120. Relative move.....	329
76. Manual device entry.....	264	121. Reset.....	121
77. Manual device parameters 0.....	267	122. Scale period.....	377
78. Manual device parameters 1.....	269	123. Sensor amplitudes.....	380
79. Manual device parameters 2.....	271	124. Sensor assignment.....	398
80. Manual device parameters 3.....	273	125. Sensor cache.....	383
81. Manual driver balance.....	220	126. Sensor status.....	405
82. Manual driver scan.....	222	127. Sensor temperature.....	385
83. Manual driver status.....	223	128. Serial communication.....	132
84. Manual motion control.....	275	129. Serial number.....	122
85. Motion direction.....	295	130. Servo control.....	407
86. Motion function.....	297	131. Software type.....	123
87. Motor brake.....	343	132. Stop deceleration.....	137
88. Motor current limit.....	346	133. Supply voltage.....	124
89. Motor current shift.....	348	134. Switch configuration.....	438
90. Motor dissipation.....	350	135. Switch status.....	441
91. Motor form.....	352	136. Target window.....	412
92. Motor optimization stage.....	355	137. Time of day.....	126
93. Motor parameters.....	357	138. Time on target.....	414
94. Motor phase current.....	362	139. Trigger capture buffer size.....	445
95. Motor phase number.....	364	140. Trigger capture index.....	447
96. Motor pole pairs.....	366	141. Trigger capture mode.....	448
97. Motor powerdown.....	373	142. Trigger capture polarity.....	450
98. Motor restart.....	374	143. Trigger capture position.....	452
99. Motor voltage gradient.....	368	144. Trigger capture position file.....	454
100. Motor voltage minimum.....	370	145. Trigger delay.....	456
101. Move abortion.....	322	146. Trigger event setup.....	458
102. Network.....	129	147. Trigger mode.....	460
103. Parameter stack (Controller)	118	148. Trigger output delay.....	467
104. Parameter stack (Axis)	253	149. Trigger output polarity.....	469
105. Parameterised absolute move....	323	150. Trigger output pulse width.....	471
106. Parameterised relative move....	324	151. Trigger pulse width.....	473
107. Pitch.....	281	152. Trigger status.....	475
108. Position display selection.....	431	153. User doubles.....	255
109. Position origin.....	300	154. User integers.....	257
110. Position origin configuration.....	303	155. User strings.....	259
111. Positioning control freeze.....	392	156. Vector acceleration.....	139
112. Positioning control mode.....	395	157. Vector move.....	331
113. Random move.....	325	158. Vector velocity.....	141
114. Range measure move.....	306	159. Velocity.....	143
115. Range measure velocity.....	308	160. Version.....	127
116. Reference configuration.....	434		
117. Reference move.....	327		
118. Reference status.....	436		

Command Reference

1. ast.....	421	46. getfreeze.....	392
2. clear.....	118	47. getinilimit.....	294
3. csave.....	243	48. getinimotorstate.....	342
4. errordecode.....	247	49. getinitactmask.....	189
5. est.....	420	50. getinitactmaskint.....	218
6. execcurrctmd.....	238	51. getmacpara.....	279
7. execnextcmd.....	237	52. getmanctrl.....	276
8. execprevcmd.....	238	53. getmanpara.....	265
9. filetrnpos.....	455	54. getmanst.....	224
10. freeze.....	393	55. getmaxcurrent.....	346
11. ga.....	140	56. getMCShift.....	349
12. gc.....	362	57. getmerror.....	117
13. ge.....	250	58. getmotindir.....	296
14. getaccel.....	140	59. getmotionfunc.....	298
15. getactcmd.....	163	60. getmotoptst.....	355
16. getactcmdint.....	199	61. getmotor.....	353
17. getactst.....	166	62. getmotorpara.....	360
18. getactstint.....	205	63. getnaccel.....	135
19. getadaptive.....	390	64. getncalswdist.....	287
20. getamc.....	339	65. getncalvel.....	289
21. getassignment.....	404	66. getnetpara.....	129
22. getaxc.....	115	67. getnlimit.....	291
23. getbaudrate.....	133	68. getnpos.....	300
24. getbrakefunc.....	344	69. getnrefvel.....	310
25. getcdfunc.....	317	70. getnrmvel.....	309
26. getcdwidth.....	318	71. getnvel.....	143
27. getchaincmd.....	236	72. getnversion.....	127
28. getcloop.....	396	73. getorgconfig.....	303
29. getclperiod.....	377	74. getphases.....	365
30. getclwindow.....	412	75. getpitch.....	281
31. getclwintime.....	415	76. getpolepairs.....	367
32. getcmdindex.....	240	77. getref.....	434
33. getcputemp.....	112	78. getrefst.....	436
34. getcurrent.....	337	79. getselpos.....	432
35. getdeviceclass.....	113	80. getsensorstatus.....	405
36. getdissipation.....	350	81. getserialno.....	122
37. getemsw.....	170	82. getsoftwaretype.....	123
38. getevtdet.....	173	83. getsp.....	410
39. getevtdetint.....	207	84. getstopdecel.....	138
40. getevtmode.....	175	85. getsw.....	439
41. getevtmodeint.....	210	86. getswst.....	441
42. getevtpol.....	181	87. gettemp.....	385
43. getevtpolint.....	213	88. gettime.....	126
44. getevtst.....	187	89. gettr.....	466
45. getevtstint.....	216	90. gettrdelay.....	456

91. gettrin.....	449	136. nrefmove.....	328
92. gettrinindex.....	447	137. nrm.....	306
93. gettrinpol.....	450	138. nrmove.....	329
94. gettrinpos.....	452	139. nsave.....	245
95. gettrinsize.....	446	140. nst.....	420
96. gettroutdelay.....	468	141. nstatus.....	420
97. gettroutpol.....	469	142. nversion.....	127
98. gettroutpw.....	472	143. p.....	423
99. gettrpara.....	459	144. pm.....	323
100. gettrst.....	475	145. pr.....	324
101. gettrwidth.....	473	146. r.....	334
102. getumotgrad.....	369	147. refmove.....	320
103. getumotmin.....	370	148. reset.....	121
104. getusupply.....	124	149. S.....	380
105. getvardbl.....	256	150. sa.....	139
106. getvarint.....	258	151. save.....	242
107. getvarstring.....	260	152. savecache.....	383
108. getvel.....	142	153. SC.....	381
109. getversion.....	111	154. scanman.....	222
110. gi.....	337	155. sensorreconnect.....	406
111. gme.....	117	156. setaccel.....	139
112. gna.....	135	157. setactcmd.....	160
113. gne.....	252	158. setactcmdint.....	200
114. gnv.....	143	159. setactst.....	167
115. gsd.....	138	160. setactstint.....	204
116. gsp.....	119	161. setadaptive.....	391
117. gv.....	142	162. setamc.....	340
118. identify.....	110	163. setassignment.....	399
119. init.....	374	164. setbaudrate.....	132
120. m.....	333	165. setbrakefunc.....	345
121. manbal.....	221	166. setcdfunc.....	316
122. merrordcode.....	248	167. setcdwidth.....	319
123. motoroff.....	373	168. setchaincmd.....	236
124. nabort.....	322	169. setcloop.....	396
125. ncal.....	286	170. setclperiod.....	378
126. ncalibrate.....	286	171. setclwindow.....	413
127. nclear.....	253	172. setclwintime.....	414
128. ngsp.....	254	173. setcmdindex.....	240
129. nidentify.....	108	174. setdoutst.....	169
130. nm.....	314	175. setemsw.....	171
131. nmove.....	314	176. setevtmode.....	175
132. np.....	429	177. setevtmodeint.....	210
133. nr.....	329	178. setevtpol.....	181
134. nrandmove.....	325	179. setevtpolint.....	212
135. nrangeasure.....	306	180. setinilimit.....	293

181. setinimotorstate.....	341	226. setvel.....	141
182. setinitactmask.....	189	227. sna.....	136
183. setinitactmaskint.....	218	228. snv.....	144
184. setmacpara.....	280	229. ssd.....	137
185. setmanctrl.....	276	230. st.....	427
186. setmanpara.....	265	231. status.....	427
187. setmaxcurrent.....	347	232. sv.....	141
188. setMCShift.....	348	233. v2m.....	334
189. setmotiondir.....	295	234. v2r.....	335
190. setmotionfunc.....	298	235. version.....	111
191. setmotor.....	353		
192. setmotorpara.....	359		
193. setnaccel.....	136		
194. setncalswdist.....	288		
195. setncalvel.....	290		
196. setnetpara.....	130		
197. setnlimit.....	292		
198. setnpos.....	301		
199. setnrefvel.....	311		
200. setnrmvel.....	308		
201. setnvel.....	144		
202. setorgconfig.....	304		
203. setphases.....	364		
204. setpitch.....	282		
205. setpolepairs.....	366		
206. setref.....	435		
207. setselpos.....	431		
208. setsp.....	410		
209. setstopdecel.....	137		
210. setsw.....	439		
211. settr.....	465		
212. settdelay.....	456		
213. settrin.....	448		
214. settrinpole.....	451		
215. settrinsize.....	445		
216. setttroutdelay.....	467		
217. setttroutpol.....	470		
218. setttroutpw.....	471		
219. setttrpara.....	458		
220. setttrwidth.....	474		
221. setumotgrad.....	368		
222. setumotmin.....	371		
223. setvardbl.....	256		
224. setvarint.....	257		
225. setvarstring.....	260		