H a n d b o o k

hydra

Firmware: 5.260000
GMT: Tue Jul 26 05:49:19 2016

Contents

About this documentation

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

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

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

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

Introduction to Venus-3

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

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

History

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

Command syntax

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

_ blank, (space) or (SP)

Parameter

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

Device Index

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

Command

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

The following letters are allowed for commands:

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

Command ending character while transmitting

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

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

Command ending character while receiving

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

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

Table of important ASCII signs for programming

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

Command execution

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

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

data input interfaces
scanner / stack
interpreter

Data converter

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

Scanner -> Interpreter -> Stack

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

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

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. 218) or ***Acceleration*** (p. 212) - 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. 575) register which will go high whenever move requests are blocked. Moreover, continued fast freewheeling move abortion (especially by *Ctrl+C* or ***nabort*** (p. 448)) 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. 579) 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 ***nmove*** (p. 435)
2:	0 1 ***ast*** (p. 579)

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. 365) or ***csave*** (p. 363) 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. 225)	Velocity setting, device-1
2:	5 2 ***snv***	Velocity setting, device-2
3:		
4:	2 1 ***setpitch*** (p. 400)	Pitch setting, device-1
5:	2 2 ***setpitch***	Pitch setting, device-2
6:	100 1 ***sna*** (p. 213)	Acceleration setting, device-1
7:		
8:	1 ***nsave*** (p. 365)	Save all parameters, device-1
9:	2 ***nsave***	Save all parameters, device-2
10:		
11:	1 ***ncal*** (p. 403)	Move to endswitches, device-1
12:	1 ***nrm*** (p. 425)	(find limits)

	Command	Description
13:		
14:	2 ***ncal***	Move to endswitches, device-2
15: 16:		
17:	15 1 ***nm*** (p. 435)	Positioning absolute, device-1
18:		
19:	1 ***nst*** (p. 579)	Ask for status, device-1
20:	1 ***gne*** (p. 372)	Ask for command decoding error (venus error)
21:	1 ***np*** (p. 589)	Ask for the actual position of device 1
22:		
23:	2.003 2 ***nm***	Positioning absolute, device 2
24:		
25:	2 ***nst***	Ask for status, device 2
26:	2 ***np***	Ask for actual position of device 2

Communication

Venus communication is available via

- the Ethernet interface
- the user RS232 port

Ethernet

With the Ethernet interface, port number is 400; default IP address is 192.168.1.200.

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.

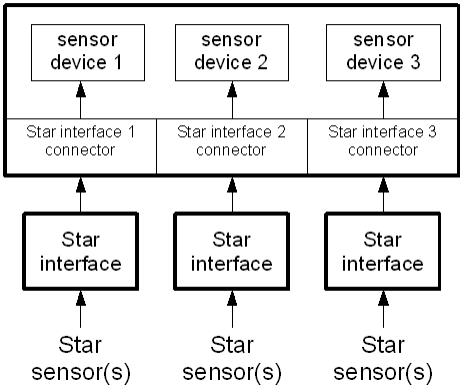
The **Sensor** device provides functionality that is especially tied to the position sensor interface port 3. The corresponding device index (3) has to be entered with each command line, so the minimum command line contains the command string and a device index. For instance, a command line querying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

3 S

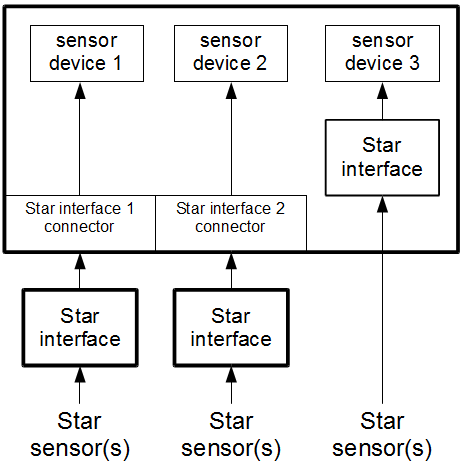
Position sensor routing

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.

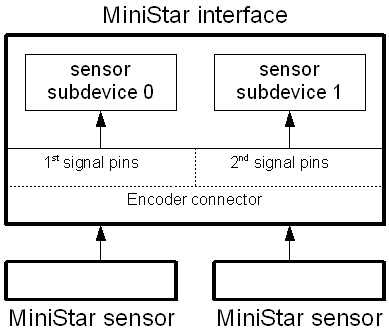
Hydra controller without Star interface

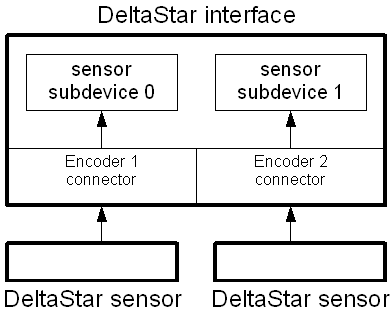


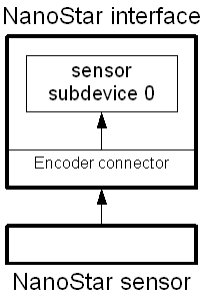
Hydra controller with built-in Star interface



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*** (p. 556) on how sensor routing is done. After routing, sensor position and (if applicable) trigger functions can be accessed using the axis device index.

Position coordinate, origin, and limits

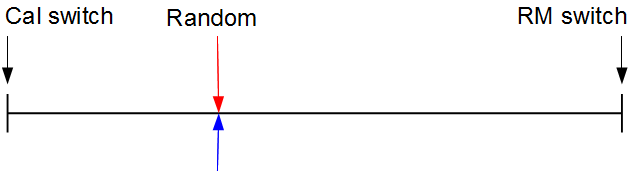
Absolute position encoders

With absolute encoders, the handling is fairly simple because the position scale holds a permanent origin. The operational position is valid from the start, and there is no need of referencing. To define the travel range related to the origin, use ***Initial limits*** (p. 410) .

Incremental position encoders

With incremental encoders, the position after powerup is approximately zero, so the (random) location of the slide at that moment also marks the initial position origin, and consequently the position coordinate needs referencing to a fixed point within the travel range before taking up operation. The initial situation is pictured below, where the *red* arrow marks the slide/sensor location, and the *blue* one denotes the operational origin.

-> Stage 1

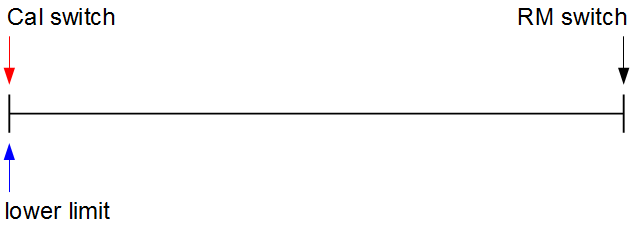


Standard procedure

This point usually is the limit switch at the lower end of the travel range, which will be called *calibration*, *cal* or *lower limit* switch throughout this document. After performing ***Calibration move*** (p. 402) , the operational origin will be located there, the shown position being 0 afterwards.

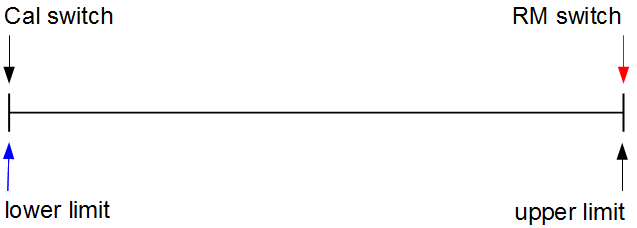
Before moving, make sure that Bit 0 is set in the ***Motion function*** (p. 415) register. The coloured arrows denote slide and origin locations as above.

-> Stage 2



Range measure move (p. 425) will move the slide to the limit switch at the opposite range end (*range measure*, *rm* or *upper limit* switch) and can be used to establish the travel range limits (***Hardware limits*** (p. 408)). Before moving, make sure that Bit 1 ist set in the ***Motion function*** register.

-> Stage 3



However, this step can be skipped if ***Initial limits*** (p. 410) are being used instead. Now the axis is basically ready for operation.

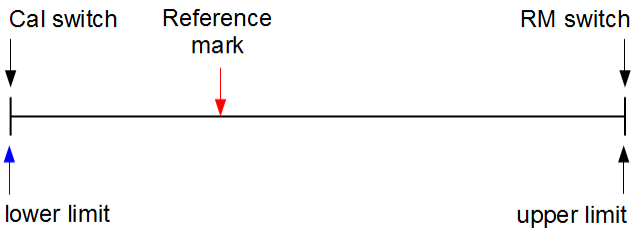
Optional followup procedure using single reference mark

To make the origin independent of the location of the switch (which might be too inaccurate or varying against the position scale with the application), a ***Reference move*** (p. 453) can be performed, provided the scale features one (or multiple) reference mark(s). To activate single reference mark detection before moving, make sure that

- ***Motion function*** (p. 415) register bit 2 is set, and bits 5 and 6 are cleared
- ***Reference configuration*** (p. 597) is set correctly

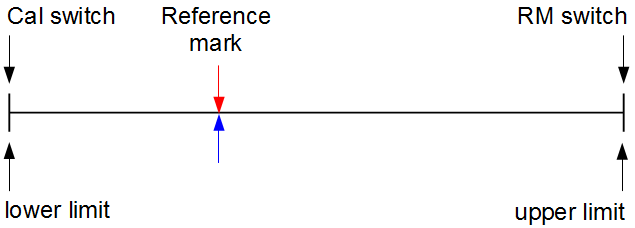
The slide will then be moved to the next reference mark in the given direction without any further action. The coloured arrows denote slide and origin locations as above.

-> Stage S4



The move will not implicitly shift the origin; if the origin is to be moved to the reference mark or any other location, this has to be done by setting the ***Position origin*** (p. 419) accordingly. ***Hardware limits*** (p. 408) will be matched implicitly.

-> Stage S5



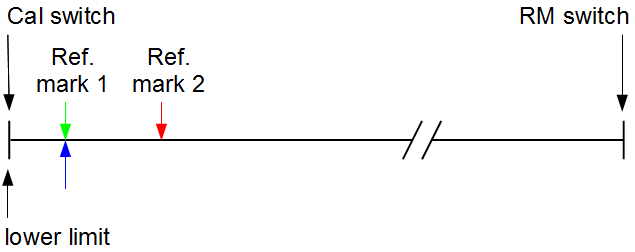
Optional followup procedure using multiple reference marks

If the position scale features multiple distance coded reference marks, ***Reference move*** (p. 453) can be used to make any of these reference marks a permanent origin (*reference origin*) of the scale. It would be self-suggesting (but not necessary) to choose the mark nearest to the cal switch, which we will do in the following. To activate reference origin establishment, make sure that

- bits 2, 5, and 6 of the ***Motion function*** (p. 415) register are set
- ***Scale basic increment*** (p. 522) , ***Reference window*** (p. 550) , and ***Reference configuration*** (p. 597) are set correctly

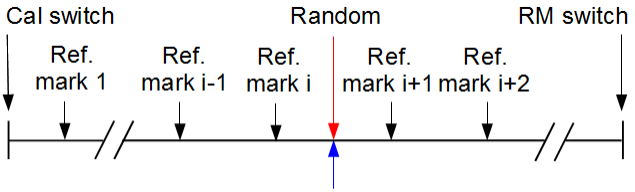
Executing ***Reference move*** will move the axis to the second reference position found, making the first one the origin. We start at *stage 2*, with the slide in the cal switch position. The coloured arrows denote slide and operational origin locations as above, the green one added being the reference origin.

-> Stage M3



Now clear bit 6 of the ***Motion function*** register; then make the origin permanent by ***Configuration storage*** (p. 362) . From now on, from any initial situation like this:

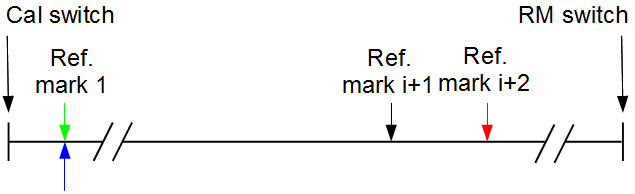
-> Stage M4



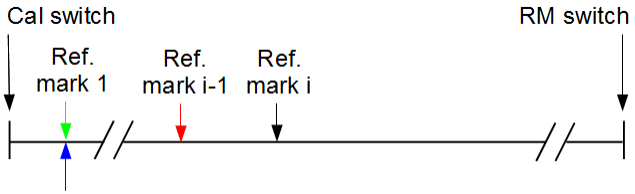
the position will be referenced to the stored origin as soon as a ***Reference move*** is performed in either direction.

-> Stage M5

After motion towards RM switch:



After motion towards Cal switch:

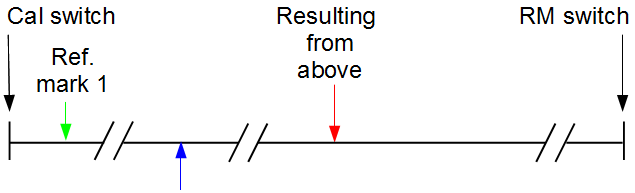


Note, however, that the procedure depends on finding two adjacent reference marks to complete; if a limit switch is found before, the axis will halt, leaving the referencing undone. It is therefore strongly advised to protect both ends of the travel range via limit switches.

Note further that, as opposed to the ***Cal move***, the referencing procedure does not include changing of ***Hardware limits*** (p. 408) .

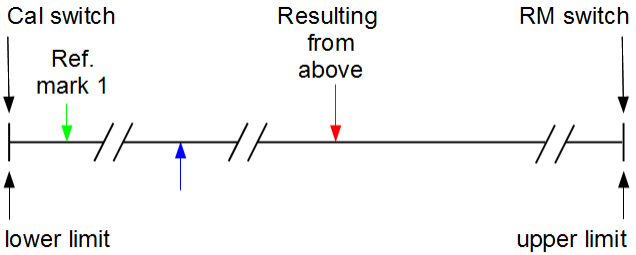
Now, if required, the operational origin can be shifted off the reference origin to any place within the motion range by setting the ***Position origin*** (p. 419) .

-> Stage M6



Now set the ***Hardware limits*** related to the operational origin.

-> Stage M7



General hints

Regardless of the technical environment, once the operational origin is determined, it can be shifted by the user to any location within the travel range at any time. This is done by setting the ***Position origin*** and does not affect positioning in any way. ***Hardware limits*** will implicitly be matched to the displacement and need not be touched. Note, however, that the operational origin - as opposed to the reference origin - is not storable.

If needed, use ***Reference offset*** (p. 429) to inquire the result of reference origin establishment. This can be useful to check if the distance coding direction matches the count direction of the position scale by repeating the origin finding procedure at different locations on the scale.

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 commutation, i.e. 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. 498), ***Pitch*** (p. 399), ***Motion direction*** (p. 412), and ***Scale period*** (p. 526)) 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. 399) and ***Motor pole pairs*** (p. 498). The angle offset will be referred to as the *commutation offset* throughout this chapter.

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 mm / 4 = 0.5 mm, and the angle offset following is 0.1 mm / 0.5 mm = 0.2.

Commutation request conditions

Normally, the *initial motor commutation* will automatically perform after controller powerup. However, utilizing ***Initial motor power state*** (p. 472), it may be configured to be put off until user request, keeping the motor dead in the mean time.

With the initial commutation done, the motor will be *recommuted* upon each ***Motor restart*** (p. 506) .

Commutation offset establishment

The commutation offset can be detected utilizing the ***Auto commutation*** (p. 469) 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 whenever a motor recommutation is requested. The resulting commutation offset is then automatically written to the ***Motor current shift*** (p. 480) parameter. Subsequently, the axis will commute with the result. Note that, depending on configuration, the enable state can vary during operation. With the auto commutation skipped, the commutation offset will be taken from a storable parameter instead. This may be

- the last auto commutation result from ***Motor current shift*** (p. 480)) *either*
- *or* an "optimized" angle which is manually defined by the user (see ***Motor parameters*** (p. 489), *optimum angle offset* entry)

The ***Auto commutation*** (p. 469) 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. Finally, there is a *mode* entry which defines the source of the commutation offset, thereby also establishing the ***Auto commutation*** (p. 469) enable conditions.

Auto commutation by default

With standalone Hydra controllers, the default *mode* setting is 1 (always on), i.e. initial commutation and later recommutation will both always implicitly execute ***Auto commutation*** (p. 469) .

Auto commutation with absolute position encoders

With *absolute position encoders*, the commutation 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 as follows:

Configuration example

Preconditions: An absolute position encoder is connected to the Hydra controller. The encoder is properly parameterized, working and assigned to axis 1.

Task: Define and store commutation offset at axis 1.

	Command	Description
1:	1 3 1 ***setamc*** (p. 471)	Make sure ***Auto commutation*** (p. 469) mode is 1 at axis 1. May be skipped if already commuted properly.
2:	1 ***init*** (p. 506)	Recommute axis 1, thereby establishing the commutation offset. May be skipped if already commuted properly.
3:	1 ***getMCShift*** (p. 481)	Read axis 1 commutation offset from ***Motor current shift*** (p. 480) for plausibility check. May be skipped if commuted properly.
4:	0 3 1 ***setamc***	Set ***Auto commutation*** mode at axis 1 to 0.
5:	1 ***nsave*** (p. 365)	Store axis 1 configuration.

Wait until parameter storage has finished.

	Command	Description
1:	***reset*** (p. 196)	Apply ***Controller reset***.

Result: Initial commutation and recommutation at axis 1 will always be done without execution of ***Auto commutation***. Commutation offset used will be taken from ***Motor current shift***.

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 (as with absolute encoders) will not do unless fixed initial conditions are provided by the user whenever a controller

powerup, ***Controller reset*** or ***Motor restart*** (p. 506) is being initiated. Due to this, ***Auto commutation*** (p. 469) should at least remain enabled with initial commutation, so setting *mode* to 0 is by no means recommended. However, once established, the commutation offset remains valid throughout operation. For this purpose, ***Auto commutation*** can be configured to be implicitly disabled after initial commutation.

Configuration example

Preconditions: An incremental position encoder is connected to the Hydra controller. The encoder is properly parameterized, working and assigned to axis 1. Initial commutation has been carried out successfully.

Task: Disable ***Auto commutation*** for all recommutations.

	Command	Description
1:	2 3 1 ***setamc*** (p. 471)	Set ***Auto commutation*** mode at axis 1 to 2.
2:	1 ***nsave*** (p. 365)	Store axis 1 configuration.

Wait until parameter storage has finished.

	Command	Description
1:	***reset*** (p. 196)	Apply ***Controller reset***.

Result: ***Auto commutation*** will now perform implicetly with initial commutation only. Recommutation will be done without renewed execution of ***Auto commutation***; commutation offset will then be taken from ***Motor current shift*** (p. 480) .

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 abolute position encoders, it is possible to enter the optimum angle offset manually as a Venus parameter (***Motor current shift*** (p. 480)), 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.

Configuration example

Preconditions: An absolute position encoder is connected to the Hydra controller. The encoder is properly parameterized, working and assigned to axis 2.

Task: Optimize commutation offset at axis 2, using ***Auto commutation*** (p. 469) results at intervallic positions, start position being s_0 and interval being *ds*.

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

Preparation: Make axis 2 ready for optimization procedure. Interval *ds* should be significantly smaller than ***Pitch*** (p. 399) .

	Command	Description
1:	1 3 2 ***setamc*** (p. 471)	Make sure ***Auto commutation*** mode is 1 at axis 2.
2:	s_0 2 ***nm*** (p. 435)	Move slide/rotor of axis 2 to start position s_0 .

Repeat following steps throughout significant position range:

	Command	Description
1:	*ds* 2 ***nr*** (p. 457)	Move slide/rotor of axis 2 by distance *ds*.
2:	2 ***init*** (p. 506)	Recommute axis 2, thereby establishing a new commutation offset.
3:	2 ***getMCShift*** (p. 481)	Read axis 2 commutation offset from ***Motor current shift*** and note for later evaluation.

Average out commutation results, forming optimized commutation offset *phi*. ***Motor current shift*** still contains the last ***Auto commutation*** result. Now disable ***Auto commutation*** and optimize and check the commutation offset as follows:

	Command	Description
1:	*phi* 2 ***setMCShift*** (p. 480)	Enter optimized commutation offset.
2:	0 3 2 ***setamc***	Set ***Auto commutation*** mode at axis 2 to 0.
3:	2 ***init***	Recommute axis 2 with new commutation offset to check for applicability.

If commutation fails, repeat whole procedure from the start. Possibly the ***Auto commutation*** parameters must be altered. Then store new commutation offset.

	Command	Description
1:	2 ***nsave*** (p. 365)	Store axis 2 configuration.

Wait until parameter storage has finished.

	Command	Description
1:	***reset*** (p. 196)	Apply ***Controller reset***.

Result: Initial commutation and recommutation at axis 2 will always be done without execution of ***Auto commutation***. Commutation offset (now optimized) will be taken from ***Motor current shift***.

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 homing has been done by application of an appropriate move, the auto commutation result can be replaced by the optimum angle offset. Hydra holds this as a Venus parameter (***Motor parameters*** (p. 489), *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. 402) or ***Reference move*** (p. 453), depending on the home location chosen. The calibration switch is to be used as the home location if no reference mark is available. Note, however, that with this configuration the optimization is sensitive to calibration switch displacement, after which the optimum commutation offset will have to be determined anew. The reference mark, in contrast, will usually be more precise, and still work after calibration switch displacement.

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. It is elementary, however, not to reset the controller before the whole procedure is finished. Otherwise, the procedure must start again from this point.

2. Wait for powerup sequence to finish.

3. Select motor optimization home location utilizing ***Motion function*** (p. 415) (must be done prior to step 5). Selection must not be changed afterwards. Keep optimization disabled for the time being.

4. Find home location by execution of move corresponding to the setting chosen above. It is *mandatory* to do this prior to step 5, because the reference point of the ***Motor current shift*** (p. 480) will change with homing!

5. Find optimum angle offset, applying ***Auto commutation*** (p. 469) several times at different slide/rotor locations. Average out the results read from ***Motor current shift***. Keep in mind results are correct only if step 4 has been carried out in advance.

6. Enter average value to ***Motor parameters***, *optimum angle offset* entry. Unless step 4 was skipped, the value entered is related to the home position, thus making a system constant. Otherwise, alignment of the ***Auto commutation*** result will be incorrect. Nevertheless, value entry without prior homing is useful in some cases, so value entry will *not* be blocked if homing was not done. Correct proceeding is due to the user!

7. Check result for applicability, utilizing ***Motor restart*** (p. 506) .

8. Arm motor optimization, again utilizing ***Motion function***. Take care not to alter the home location selection inadvertedly. Motor will recommute.

9. If the result is satisfactory, store parameters using ***Configuration storage (Axis)***.

10. Initiate ***Reset*** (p. 196) .

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

Note that it is basically possible to enable motor optimization in one go with home location selection during step 3, and skip step 7. However, this is not recommended, because in this case the motor will recommute immediately upon completion of step 6, without any chance of correcting a misentry, which might eventually lead to indeterminate action.

Optimization example

Preconditions: An incremental position encoder is connected to the Hydra controller. The encoder is properly parameterized, working and assigned to axis 2.

Task: Optimize commutation offset at axis 2, using ***Auto commutation*** results at intervallic positions, start position

being s_0 and interval being *ds*. Choose calibration limit switch as home location.

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

Preparation: Perform steps 1 and 2. Then make axis 2 ready for optimization procedure as follows. Interval *ds* should be significantly smaller than ***Pitch*** (p. 399) .

	Command	Description
1:	7 2 ***setmotionfunc*** (p. 418)	Step 3. Choose axis 2 home location (calibration limit switch). Keep optimization disabled.
2:	2 ***ncal*** (p. 403)	Step 4. Find axis 2 home location.
3:	1 3 2 ***setamc*** (p. 471)	Make sure ***Auto commutation*** mode is 1 at axis 2.
4:	s_0 2 ***nm*** (p. 435)	Move slide/rotor of axis 2 to start position s_0 .

Step 5. Repeat following steps throughout significant position range:

	Command	Description
1:	*ds* 2 ***nr*** (p. 457)	Move slide/rotor of axis 2 by distance *ds*.
2:	2 ***init*** (p. 506)	Recommute axis 2, thereby establishing a new commutation offset.
3:	2 ***getMCShift*** (p. 481)	Read axis 2 commutation offset from ***Motor current shift*** and note for later evaluation.

Average out commutation results, forming optimized commutation offset *phi*. Now disable ***Auto commutation*** and optimize and check the commutation offset as follows:

	Command	Description
1:	*phi* 1 2 ***setmotorpara*** (p. 491)	Step 6. Enter optimized commutation offset to ***Motor parameters***, *optimum angle offset*.
2:	1 2 ***getmotorpara*** (p. 492)	Recheck value to make sure entry was correct.
3:	3 3 2 ***setamc***	Set ***Auto commutation*** mode at axis 2 to 3.
4:	2 ***init***	Step 7. Recommute axis 2 with new commutation offset to check for applicability. Axis will commute using ***Motor parameters***, *optimum angle offset*.

If commutation fails, repeat whole procedure from the start. Possibly the ***Auto commutation*** parameters must be altered. Otherwise, configure ***Calibration move*** to activate motor optimization implicetly. Then store new commutation offset and optimization configuration.

	Command	Description
1:	15 2 ***setmotionfunc***	Step 8. Reconfigure ***Calibration move*** to activate optimization implicetly.
2:	2 ***nsave*** (p. 365)	Step 9. Store axis 2 configuration.

Wait until parameter storage has finished.

	Command	Description
1:	***reset*** (p. 196)	Step 10. Apply ***Controller reset***.

Result: ***Auto commutation*** will now perform implicitly with initial commutation only. Afterwards, homing and optimization will be done in one go by application of ***Calibration move***. Then, each application of ***Motion restart*** will implicetly commute with ***Motor parameters***, *optimum angle offset*.

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".

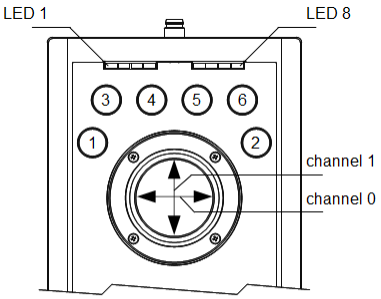
Each Hydra axis device holds 4 *manual devices* with a ***Manual device parameters*** set each. A *manual device* is a virtual slot any channel of any connected *manual driver* can virtually be plugged into in order to drive the respective axis. In doing so, multiple use is possible, so each driver channel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

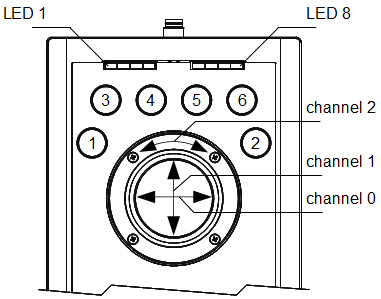
manual driver	driver index*	number of channels (m)	channel index
1st CAN joystick	0	2	0...1
		3	0...2
1st CAN handwheel	1	2	0...1
2nd CAN joystick	2	2	0...1
		3	0...2
2nd CAN handwheel	3	2	0...1

Manual driver properties

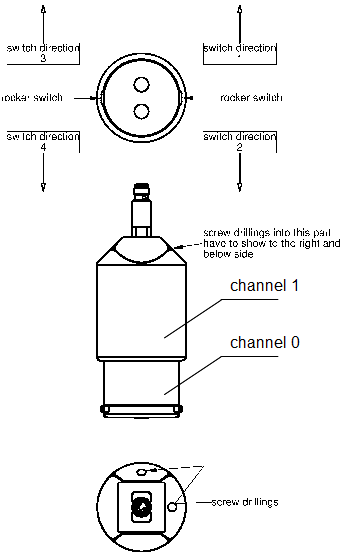
*2-channel and 3-channel CAN joysticks share the same driver indexes. The number of available channels is auto-detected by firmware.



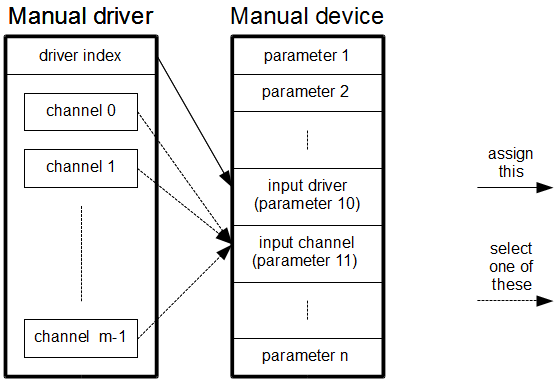
2-channel joystick



3-channel joystick



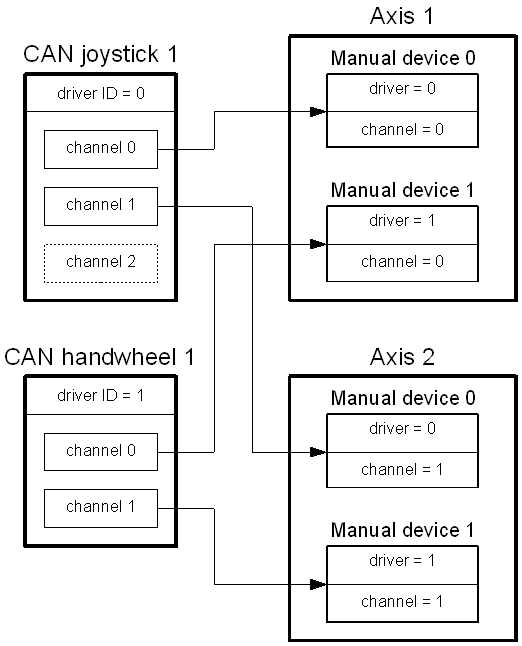
2-channel handwheel



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



Standard routing

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. 388) are 10 and 11, respectively.

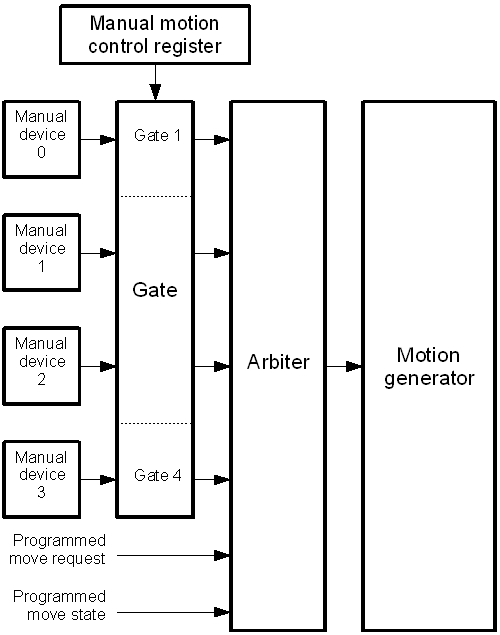
Note: Unlike in the table below, all commands have to be entered *in one line*.

	Command	Description
1:	0 10 1 1 ***setmanpara*** (p. 386)	Assign manual driver *joystick 1* to ***Manual device 0*** of Axis 1.
2:	0 10 1 2 ***setmanpara***	Assign manual driver *joystick 1* to ***Manual device 0*** of Axis 2.
3:	1 11 1 1 ***setmanpara***	Select channel 1 of manual driver assigned above (*joystick 1*) to feed ***Manual device 0*** of Axis 1.
4:	0 11 1 2 ***setmanpara***	Select channel 0 of manual driver assigned above (*joystick 1*) 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. 396) register); several *manual devices* may simultaneously be enabled at one axis. An arbiter inhibits conflicting accesses to axis motion.

Note that pushbutton function will *not* 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 either. Another facility of disabling motion data generation is the *mode* entry of the corresponding ***Manual device parameters***.



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 joystick parameterization example

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. 386)	Set full elongation velocity at ***Manual device 0*** of Axis 1 to 20 mm/s (*velocity* entry).
2:	200 2 1 1 ***setmanpara***	Set acceleration at ***Manual device 0*** of Axis 1 to 200 mm/s² (*acceleration* entry).
3:	2 6 1 1 ***setmanpara***	Set elongation-to-velocity transfer function at ***Manual device 0*** of Axis 1 to cubic progression (*input function* entry).
4:	2 7 1 1 ***setmanpara***	Set position data generation at ***Manual device 0*** of Axis 1 to be enabled throughout operation (*mode* entry).
5:	0.1 8 1 1 ***setmanpara***	Set position data generation threshold at ***Manual device 0*** of Axis 1 to 0.1 (*threshold* entry).
6:	0 9 1 1 ***setmanpara***	Set effective direction at ***Manual device 0*** of Axis 1 to standard direction (*direction* entry).
7:	0 10 1 1 ***setmanpara***	Assign manual driver *joystick 1* to ***Manual device 0*** of Axis 1 (*input driver* entry).
8:	1 11 1 1 ***setmanpara***	Select channel 1 of manual driver assigned above (*joystick 1*) to feed ***Manual device 0*** of Axis 1 (*input channel* entry).
9:	10 1 1 2 ***setmanpara***	Set full elongation velocity at ***Manual device 0*** of Axis 2 to 10 mm/s (*velocity* entry).
10:	500 2 1 2 ***setmanpara***	Set acceleration at ***Manual device 0*** of Axis 2 to 500 mm/s² (*acceleration* entry).
11:	1 6 1 2 ***setmanpara***	Set elongation-to-velocity transfer function at ***Manual device 0*** of Axis 2 to square progression (*input function* entry).

	Command	Description
12:	2 7 1 2 ***setmanpara***	Set position data generation at ***Manual device 0*** of Axis 2 to be enabled throughout operation (*mode* entry).
13:	0.1 8 1 2 ***setmanpara***	Set position data generation threshold at ***Manual device 0*** of Axis 2 to 0.1 (*threshold* entry).
14:	1 9 1 2 ***setmanpara***	Set effective direction at ***Manual device 0*** of Axis 2 to reverse direction (*direction* entry).
15:	0 10 1 2 ***setmanpara***	Assign manual driver *joystick 1* to ***Manual device 0*** of Axis 2 (*input driver* entry).
16:	0 11 1 2 ***setmanpara***	Select channel 0 of manual driver assigned above(*joystick 1*) to feed ***Manual device 0*** of Axis 2 (*input channel* entry).
17:	1 1 ***setmanctrl*** (p. 397)	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.

Solution 1:

	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.

Solution 2:

	Command	Description
1:	0 1 ***setmanctrl***	Disable ***Manual device 0*** of Axis 1.
2:	0 2 ***setmanctrl***	Disable ***Manual device 0*** of Axis 2.

Complete handwheel parameterization example

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 ***setmanpara*** (p. 386)	Set maximum velocity at ***Manual device 1*** of Axis 1 to 50 mm/s (*velocity* entry).
2:	500 2 2 1 ***setmanpara***	Set acceleration at ***Manual device 1*** of Axis 1 to 500 mm/s² (*acceleration* entry).
3:	323584 4 2 1 ***setmanpara***	Set wheel encoder resolution at ***Manual device 1*** of Axis 1 to 323584 inc./rev., which is appropriate for ITK CAN handwheel (*resolution* entry).

	Command	Description
4:	1 5 2 1 ***setmanpara***	Set wheel transformation at ***Manual device 1*** of Axis 1 to 1 mm/rev. (*ratio* entry).
5:	2 7 2 1 ***setmanpara***	Set position data generation at ***Manual device 1*** of Axis 1 to be enabled throughout operation (*mode* entry).
6:	1 10 2 1 ***setmanpara***	Assign manual driver *handwheel 1* to ***Manual device 1*** of Axis 1.
7:	0 11 2 1 ***setmanpara***	Select channel 0 of manual driver assigned above (*handwheel 1*) to feed ***Manual device 1*** of Axis 1.
8:	50 1 3 1 ***setmanpara***	Set maximum velocity at ***Manual device 2*** of Axis 1 to 50 mm/s (*velocity* entry).
9:	500 2 3 1 ***setmanpara***	Set acceleration at ***Manual device 2*** of Axis 1 to 500 mm/s² (*acceleration* entry).
10:	323584 4 3 1 ***setmanpara***	Set wheel encoder resolution at ***Manual device 2*** of Axis 1 to 323584 inc./ rev., which is appropriate for ITK CAN handwheel (*resolution* entry).
11:	10 5 3 1 ***setmanpara***	Set wheel transformation at ***Manual device 2*** of Axis 1 to 10 mm/rev. (*ratio* entry).
12:	2 7 3 1 ***setmanpara***	Set position data generation at ***Manual device 2*** of Axis 1 to be enabled throughout operation (*mode* entry).
13:	1 10 3 1 ***setmanpara***	Assign manual driver *handwheel 1* to ***Manual device 2*** of Axis 1.
14:	1 11 3 1 ***setmanpara***	Select channel 1 of manual driver assigned above (*handwheel 1*) to feed ***Manual device 2*** of Axis 1.
15:	7 1 ***setmanctrl*** (p. 397)	Enable ***Manual device 0*** , ***Manual device 1*** , and ***Manual device 2*** of Axis 1.

	Command	Description
16:	1 2 ***setmanctrl***	Enable ***Manual device 0*** of Axis 2.

Scale correction (extra option)

Correction availability

The scale correction is an extra option which needs to be enabled with a release code. It is available only with a valid release code file *and* valid correction files (one per axis) located on the flash file system. Note, however, that no release code is needed with with LMT specific firmware rev. 5.1100. Note further that there is no support for scale correction with rotational position encoders.

Correction description

The scale correction is designed to compensate static inaccuracies of the position scale. To establish the necessary correction data, the measurement scale is virtually divided into sections by a grid of equidistant nodes. The correction facility utilizes a table with one set of correction data per node, matching the respective scale position. The correction amount at interjacent positions is calculated from the node data by linear spline interpolation.

Steps to take in order to establish correction data:

1. Define scale position range.

2. Define location of first node position s_0 relative to scale origin. The origin location is the one that corresponds with scale position "0" in the intended application.

3. Define grid with node distance *d*, resulting in number of nodes *n*.

4. Record position aberration table over the range defined above (node number $i = 1...n$), with one entry per node, using a reference position measurement system. Aberration is

$ds(i) = s_m(i) - s_r(i)$

where

s_m is the node position of the uncorrected scale

s_r is the corresponding reference position

First node position is $s_m(1) = s_0$.

Second node position is $s_m(2) = s_0 + d$.

Last node position is $s_m(n) = s_0 + (n - 1) * d$.

5. Calculate correction coefficients a_0 and a_1 for each node from respective aberration values as follows:

$a_0(i) = ds(i)$

$a_1(i) = (ds(i+1) - ds(i)) / d$

and estimate a_1 for last node:

$a_1(n) = 2\ a_1(n-1) - a_1(n-2)$

Correction file

A correction file contains a correction table providing all information needed for scale correction with one axis.

Filename is:

- Dev_1_corr.csv for axis 1 (typically X)
- Dev_2_corr.csv for axis 2 (typically Y)

The file is divided into two sections:

- the correction table header (including general properties, e.g. table format)
- the correction table data from calculation above

Table header

The header section invariably consists of 6 lines described below. All data are given in ASCII with <CR><LF> as line delimiter.

Line No 1:

Format: String with 255 characters in maximum

Content: arbitrary

Usage: Free for customers need (e.g. description)

Example: Machine 3 axis 1 scale correction

Line No 2:

Format: integer<CR><LF>

Content: *1...3*

Usage: table alignment

Line No 3:

Format: integer<CR><LF>

Content: *1*

Usage: internal

Line No 4:

Format: double<CR><LF>

Content: s_0 (from above)

Usage: first node position, relative to defined scale origin [mm]

Example: -1.23456789

Line No 5:

Format: double<CR><LF>

Content: *d* (from above)

Usage: Distance between adjacent nodes [mm]

Example: 0.10

Line No 6:

Format: integer<CR><LF>

Content: *n* (from above)

Usage: Number of nodes

Example: 1201

Table data

The data section consists of the linear spline coefficients, each node represented by one line.

Note that the number of lines in the data section must equal the *n* parameter from the header section.

All data are given in ASCII with <CR><LF> as line delimiter.

Line No 7 to line No (6 + *n*)

Format: double<TAB>double<CR><LF>

Content: *a1(i) a0(i)* (from above, *i* = 1...*n*)

Usage: coefficients for linear correction spline

Example: -0.1234 5.6789

Table alignment

Before the correction facility is ready for use, the origin of the correction table taken from the correction file has to be aligned to a defined location within the motion range, the correction *home location*. This location will be one out of the following:

- the scale origin
- the calibration switch
- the scale reference mark

The home location selection is predefined by an entry in the correction file header. See ***Position correction file parameters*** (p. 515) for more information regarding the correction table properties.

Scale origin

With the scale origin selected as the home location, the correction facility is ready for use as soon as the system has powered up faultlessly. Note, however, that the scale origin selection is only applicable to and should preferably be used with systems featuring absolute position measurement.

Calibration switch

With the calibration switch selected as the home location, the correction facility is ready for use as soon as a ***Calibration move*** (p. 402) has completed successfully. Note, however, that the table will be realigned with each further ***Calibration move***, so the home location may shift slightly, depending on the reproducibility of the switching point and the ***Calibration velocity*** (p. 406) .

Scale reference mark

With the scale reference mark selected as the home location, the correction facility is ready for use as soon as a ***Reference move*** (p. 453) has completed successfully. Note, however, that the table will be realigned with each further ***Reference move***, so the home location may shift slightly, depending on the reference mark width and the ***Reference velocity*** (p. 431) .

Table loading

If a correction file is existent, the correction table is loaded automatically during powerup. However, with the correction not needed, the load procedure can be skipped utilizing the *autoload* entry of the ***Scale correction parameters***. Note that it is skipped by default.

Correction activation

The correction facility can be activated either automatically or manually. With automatical activation selected, the correction will become active once the table is aligned to its home location (see above). This can be achieved utilizing the *autoactivate* entry of the ***Scale correction parameters***. Otherwise, the correction facility will remain inactive until activated manually by ***Scale correction activation***.

Examples

Example 1

Preconditions: Correction files are available for both axes, release code is valid. The system is equipped with 2 translational axes with a travel range of 100 mm each, featuring incremental position scales with reference marks and calibration switches. Calibration and reference move parameters are set appropriately.

Task: Correction is to be activated automatically at both axes.

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

Step 1: Inquire selected home locations.

	Command	Description
1:	1 1 ***getposcorrfilepara*** (p. 516)	Inquire home location at Axis 1. Result be 1.
2:	1 2 ***getposcorrfilepara***	Inquire home location at Axis 2. Result be 2.

Inquiry result indicates home locations as follows:

- calibration switch at Axis 1
- scale reference mark at Axis 2

Step 2: Configure axes for automatic correction activation.

	Command	Description
1:	1 1 1 ***setposcorrpara*** (p. 519)	Enable *autoload* function at Axis 1.

	Command	Description
2:	1 1 2 ***setposcorrpara***	Enable *autoload* function at Axis 2.
3:	1 2 1 ***setposcorrpara***	Enable *autoactivate* function at Axis 1.
4:	1 2 2 ***setposcorrpara***	Enable *autoactivate* function at Axis 2.
5:	***save*** (p. 362)	Store all parameters.

Wait until storage has completed.

	Command	Description
1:	***reset*** (p. 196)	Reset controller.

Wait until controller has powered up again.

Step 3: Find respective home locations.

	Command	Description
1:	1 ***ncal*** (p. 403)	Execute ***Calibration move*** (p. 402) at Axis 1.
2:	2 ***ncal***	Execute ***Calibration move*** at Axis 2.

Wait until moves have completed. Now the correction will be active at Axis 1. Axis 2, with the correction still being inactive, is prepared to seek the (next) reference mark.

	Command	Description
1:	100 2 ***nrefmove*** (p. 456)	Execute ***Reference move*** at Axis 2.

After move completion, the correction will be active at Axis 2.

Example 2

Preconditions: Correction file is available for Axis 1, release code is valid. Axis 1 is a translational axis, equipped with an absolute position scale.

Task: Correction is to be activated automatically at Axis 1.

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

Step 1: Inquire selected home location.

	Command	Description
1:	1 1 ***getposcorrfilepara***	Inquire home location at Axis 1. Result be 0.

Inquiry result indicates the scale origin to be the home location.

Step 2: Configure Axis 1 for automatic correction activation.

	Command	Description
1:	1 1 1 ***setposcorrpara***	Enable *autoload* function at Axis 1.
2:	1 2 1 ***setposcorrpara***	Enable *autoactivate* function at Axis 1.
3:	***1 nsave***	Store Axis 1 parameters.

Wait until storage has completed.

	Command	Description
1:	***reset***	Reset controller.

Wait until controller has powered up again.

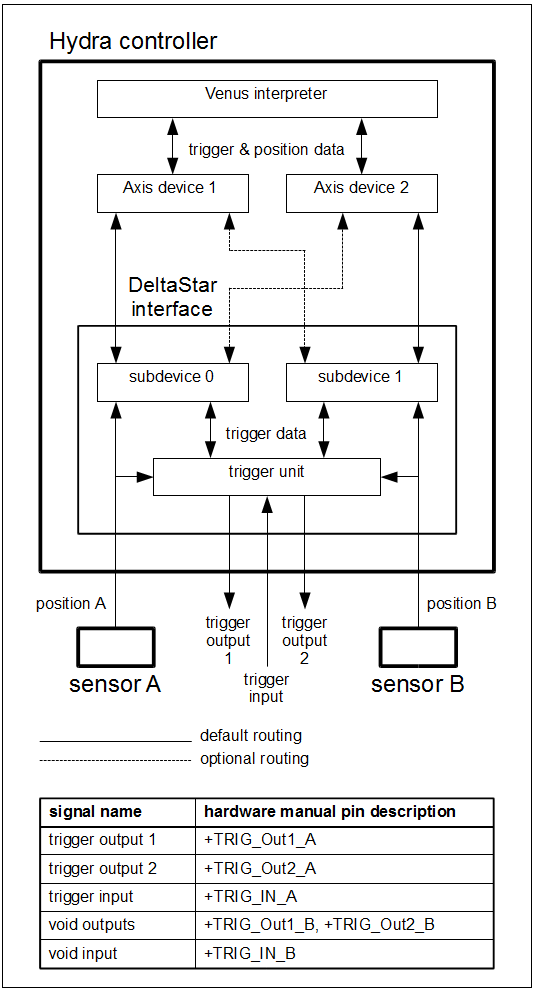
The correction will now be active at Axis 1.

Introduction to trigger function

To be able to use trigger function with any axis, the latter has to have a physically connected position sensor at its disposal. This requires correct position sensor (and, if not built-in, position interface) connection and assignment to the axis in question (s. also ***Sensor assignment*** (p. 556)). In any other case, trigger function will not be available.

Hydra distinguishes between the *DeltaStar* and the *Onboard* trigger facilities, so trigger function depends on the position sensor interfaces present at your Hydra controller. These two trigger facilities are mutually exclusive. The *DeltaStar* position sensor interface (*not* the DeltaStar Eco model) is equipped with a high resolution hardware trigger unit, providing a trigger signal output and position capture function. With any other position sensor interface (*including* the DeltaStar Eco model), Hydra provides an *Onboard* trigger, which is a firmware facility located on the Hydra controller, and applicable for simpler tasks requiring trigger signal generation. Though functional range is different, Venus commands and parameters are uniform. The trigger facility to be used is being auto-detected; manual selection is neither required nor offered.

DeltaStar trigger



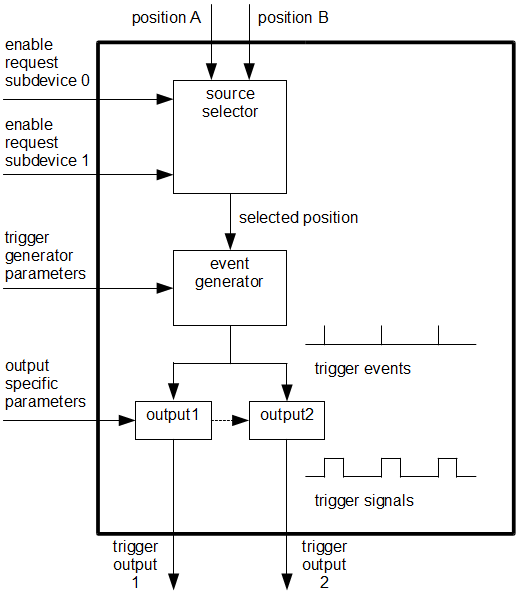
signal name	hardware manual pin description
trigger output 1	+TRIG_Out1_A
trigger output 2	+TRIG_Out2_A
trigger input	+TRIG_IN_A
void outputs	+TRIG_Out1_B, +TRIG_Out2_B
void input	+TRIG_IN_B

DeltaStar trigger overview

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.

Output trigger function



DeltaStar trigger output channel

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. 629) and ***Trigger event setup*** (p. 627) on how to enable the trigger function at a specified subdevice.

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 events can either be based on equal slide/rotor position distances or on positions individually set via a table. The *trigger generator parameters* ***Trigger mode***, ***Trigger delay compensation*** (p. 625), and

- ***Trigger event setup*** with the equidistant trigger function
- ***Trigger table point setting*** (p. 656) with the table trigger function

provide for definition of the trigger 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. 642), ***Trigger output pulse width*** (p. 645), and ***Trigger output delay*** (p. 640). 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 implicite 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 armed implicitly)
- third, if required, transmit trigger generator parameters for the target subdevice; trigger generator will be armed 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**

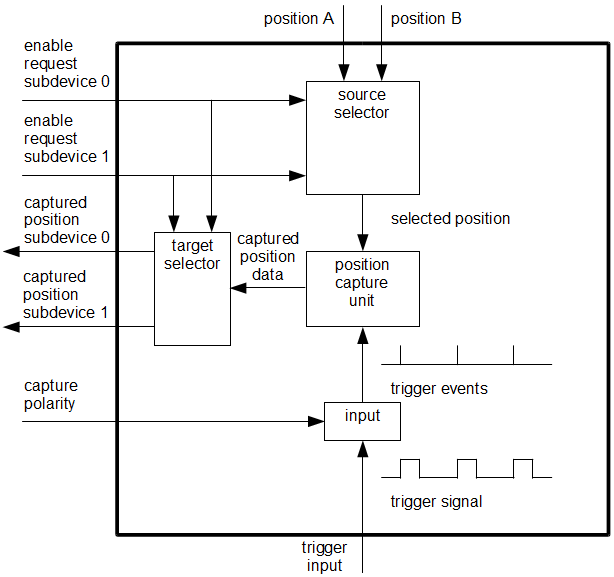
Note that whatever configuration options the Hydra firmware may offer, support is depending on the respective DeltaStar logic/firmware revision.

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*** (p. 556). 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.

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. 615) on how to enable the trigger function at a specified subdevice.



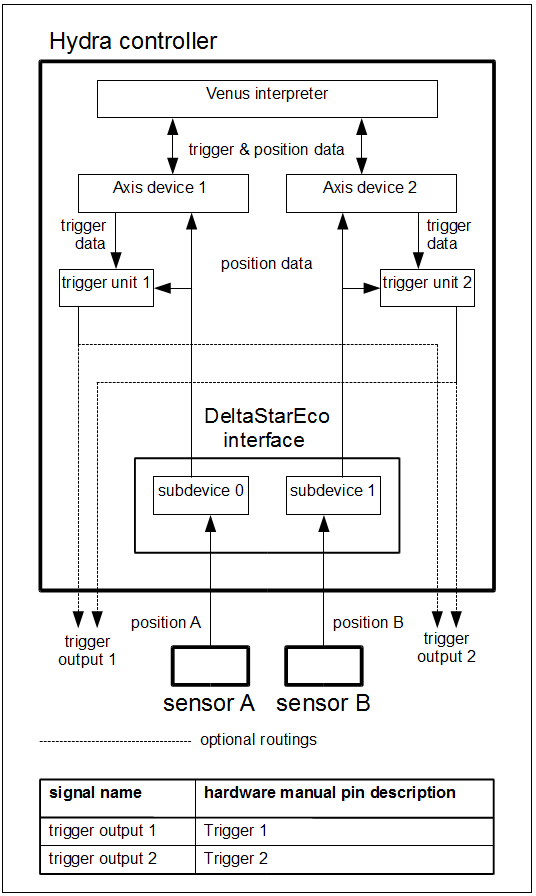
DeltaStar triggered position capture channel

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. 617)) 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. 619) .

Note that the minimum temporal gap between consecutive trigger events must be *250 µs* in order to maintain proper position capturing.

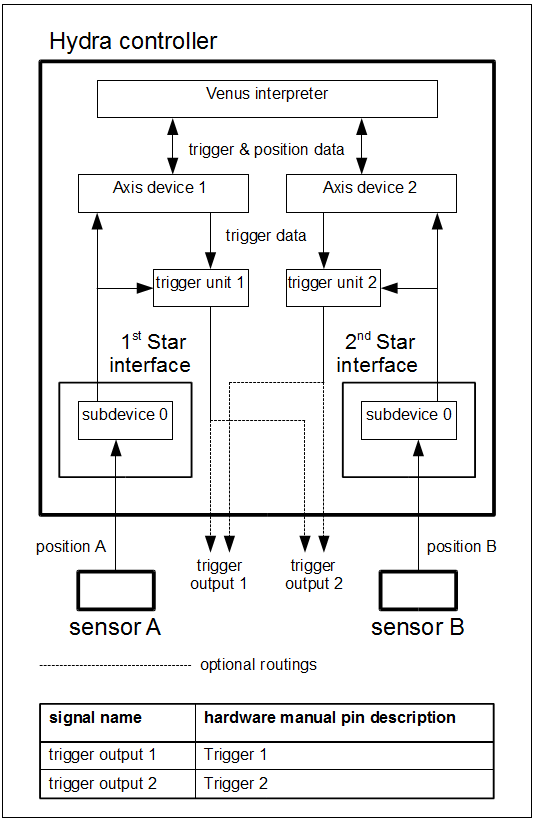
Note that the trigger unit is implicitly disarmed whenever a single sequence finishes, and always has to be rearmed for another sequence.

Onboard trigger



signal name	hardware manual pin description
trigger output 1	Trigger 1
trigger output 2	Trigger 2

Trigger overview (with DeltaStarEco)

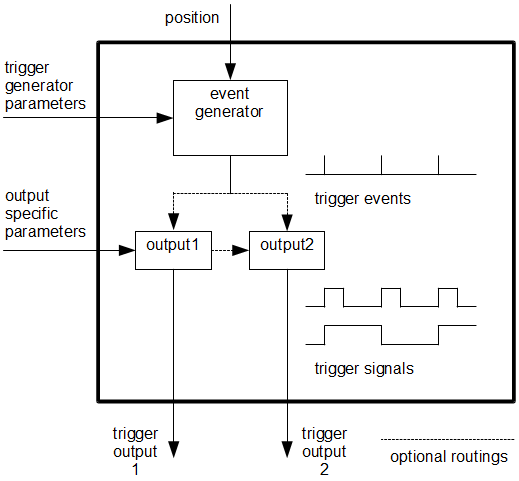


signal name	hardware manual pin description
trigger output 1	Trigger 1
trigger output 2	Trigger 2

Trigger overview (with any other Star interface)

Hydra offers two trigger signal outputs (**trigger output 1**, **trigger output 2**). Technical format and physical location of these depend on the specific Hydra model. The outputs can simultaneously be used for trigger function by both axes, and alternatively be used as general purpose TTL outputs. Hydra does not check or prevent conflicting access in any way; proper use is in full responsibility of the user.

Output trigger function



Onboard trigger output channel

The Hydra output trigger concept distinguishes generation of trigger events from the generation of the actual signal output as shown above. If required, the trigger output function can be enabled at both axes simultaneously.

The trigger position source is determined by the position sensor assignment to the target axis. See ***Trigger mode*** and ***Trigger event setup*** on how to enable the trigger function at a specified axis.

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 drives one out of the two dedicated **trigger outputs** which form trigger pulses out of each event, individually configurable regarding *output specific parameters* such as ***Trigger output polarity*** and ***Trigger output pulse width***. The overall result is an event-

synchronous trigger output signal. Output assignment is free, so both axes may alternately drive the same output.

Note that between consecutive trigger events, a gap of *500 µs min.* must be maintained. Whenever this condition is violated by excess of the maximum speed matching the trigger position interval, the trigger unit is being overrun. In this case, pulse generation will be freewheeling at the highest possible rate until catching up with the trigger events again. Therefore, the overall number of trigger pulses generated during the trigger sequence will always be correct as long as the move range covers the entire range of trigger positions. Edges may jitter by several 10 µs.

Note that the slide or rotor of the target axis has to be in stable standstill well outside the trigger area (i.e. below or above the first set trigger position, depending on the orientation of the sequence) prior to trigger activation. Otherwise, the trigger unit may generate faulty trigger pulses as soon as being activated.

Note that the trigger outputs are configured via the axes they are assigned to. Therefore, it is vital to perform output selection prior to output parameter configuration. The proper way to use the trigger output feature is to

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

Never try to alter trigger-related parameters while a trigger sequence is actually running. A currently running sequence has to be finalized in advance by disabling the trigger

channel via ***Trigger mode***; otherwise the trigger output behaviour will be indeterminate.

Available since firmware rev. 4.3000.

Note that the trigger unit is implicitly disarmed whenever a single sequence finishes, and always has to be rearmed for another sequence. Note further that once the trigger is configured, it is not necessary to retransmit the whole set of all trigger-related parameters prior to each trigger sequence. Once output selection is done and the output specific parameters are set, trigger sequences can be launched as follows:

- To repeat a trigger sequence generated from an invariable set of trigger positions in always the same processing order, it is sufficient to retransmit the ***Trigger mode*** each time you want to set up a new sequence.
- To change trigger positions or processing order with each new sequence, it is sufficient to transmit the new ***Trigger event setup*** to set up the sequence. The trigger channel will be armed implicitly.

Examples

Output trigger examples

Before trying one of the examples, please consider the documentation of the Venus parameters used therein.

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. 556)), so axis 1 is the position source. Current position at axis 1 is 0.

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

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

	Command	Description
1:	5 1 1 ***settroutpw*** (p. 645)	Set trigger pulse width at output 1 at axis 1 to 5 µs.
2:	8 1 1 ***settroutdelay*** (p. 640)	Set trigger delay at output 1 at axis 1 to 8 µs.
3:	0 1 1 ***settroutpol*** (p. 643)	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. 638)	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. 627)	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*** (p. 435)	Move axis 1 to 100 mm.

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

Example 2

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

Task: Put out a continuous 50 Hz signal at both trigger outputs, 3 µs pulsewidth at both outputs, active high at output 1, inverted and delayed by 7 µs at output 2.

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

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

Example 3

Preconditions: The Hydra controller is equipped with two NanoStar sensor interfaces. Current position at axis 2 is 0.

Task: Move axis 2 to 50 mm, trigger output 2 thereby generating 21 active low 1 ms wide equidistant trigger pulses, starting at 20 mm, stopping at 40 mm.

	Command	Description
1:	2 2 ***settr***	Prepare axis 2 to generate equidistant trigger pulses at output 2.
2:	20 40 21 2 ***settrpara***	Specify first and last trigger events to occur at axis 2 positions 20 mm and 40 mm, overall number of equidistant pulses being 21 (=> trigger interval = 1 mm), and arm trigger.
3:	1000 1 2 ***settroutpw***	Set trigger pulse width at output 2 (via axis 2) to 1 ms.
4:	1 1 2 ***settroutpol***	Specify trigger polarity at output 2 (via axis 2) to be active low.
5:	50 2 ***nm***	Move axis 2 to 50 mm.
6:	2 2 ***settr***	Rearm trigger.
7:	0 2 ***nm***	Return axis 2 to 0 mm.
8:	50 2 ***nm***	Move axis 2 to 50 mm.

Result: Pulse sequence will be put out twice as specified while axis is moving.

Example 4

Preconditions: The Hydra controller is equipped with two NanoStar sensor interfaces. Both axes are located at position 0.

Task: Consecutively move axes 1 and 2 to 10 mm each, trigger output 1 thereby generating 1 trigger pulse with each move: going high at 3 mm and return to low at 6 mm with the first, and going high at 5 mm and return to low at 7 mm with the second move.

	Command	Description
1:	3 1 ***settr***	Prepare axis 1 to generate flip-flop trigger pulse at output 1.
2:	3 6 2 1 ***settrpara***	Specify first and last trigger events to occur at axis 1 positions 3 mm and 6 mm, overall number of equidistant events being 2 (1 for each signal transition), and arm trigger.
3:	0 1 1 ***settroutpol***	Specify default trigger polarity at output 1 (via axis 1) to be low.
4:	10 1 ***nm***	Move axis 1 to 10 mm.

Result: Specified pulse will emerge at trigger output 1 while axis is moving. Wait until move has finished.

	Command	Description
1:	3 2 ***settr***	Prepare axis 2 to generate flip-flop trigger pulse at output 1.
2:	5 7 2 2 ***settrpara***	Specify first and last trigger events to occur at axis 2 positions 5 mm and 7 mm, overall number of equidistant events being 2 (1 for each signal transition), and arm trigger.
3:	10 2 ***nm***	Move axis 2 to 10 mm.

Result: Specified pulse will emerge at trigger output 1 while axis is moving.

Example 5

Preconditions: The Hydra controller is equipped with a DeltaStar sensor interface. The subdevice 0 is assigned

to axis device 1 (see ***Sensor assignment***), so axis 1 is the position source. Current position at axis 1 is 0.

Task: Move axis 1 to 80 mm, trigger output 1 thereby generating 4 active high 10 µs wide trigger pulses at positions 15 mm, 25 mm, 30 mm, and 75 mm, without any delay at output 1; respective direction signal (in consecutive order) high, low, high, and low at trigger output 2.

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

	Command	Description
1:	10 1 1 ***settroutpw***	Set trigger pulse width at output 1 at axis 1 to 10 µs.
2:	0 1 1 ***settroutdelay***	Set trigger delay at output 1 at axis 1 to 0 µs.
3:	0 1 1 ***settroutpol***	Specify trigger polarity at output 1 at axis 1 to be active high.
4:	4 1 ***settr***	Prepare axis 1 to generate table trigger pulses at output 1 and the direction signal at output 2.
5:	0 15 1 1 ***settrpoint*** (p. 656)	Specify 1st trigger event at axis 1 to occur at 15 mm, output 2 putting out high signal.
6:	1 25 0 1 ***settrpoint***	Specify 2nd trigger event at axis 1 to occur at 25 mm, output 2 putting out low signal.
7:	2 30 1 1 ***settrpoint***	Specify 3rd trigger event at axis 1 to occur at 30 mm, output 2 putting out high signal.
8:	3 75 0 1 ***settrpoint***	Specify 4th trigger event at axis 1 to occur at 75 mm, output 2 putting out low signal.
9:	1 ***sendtrtable*** (p. 658)	Transmit trigger table generated above to DeltaStar interface.

	Command	Description
10:	80 1 ***nm***	Move axis 1 to 80 mm.

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

Triggered position capture example

Example 6

Preconditions: The Hydra controller is equipped with a DeltaStar sensor interface. The subdevice 0 is assigned to axis device 1 (see ***Sensor assignment*** (p. 556)). 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:	1 1 ***settrinpol*** (p. 618)	Match trigger input to the active low trigger source by making it sensitive to falling edges.
2:	10 1 ***settrinsize*** (p. 611)	Prepare capture buffer to record up to 10 position values.
3:	1 1 ***settrin*** (p. 615)	Arm trigger capture function.
4:	100 1 ***nm*** (p. 435)	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 ***gettrinindex*** (p. 613)	Query number of captured position values. Should be 7 at least.

Query captured positions. We assume perfect action.

	Command	Description
1:	0 1 ***gettrinpos*** (p. 619)	Reply: 0.000000
2:	1 1 ***gettrinpos***	Reply: 6.250000
3:	2 1 ***gettrinpos***	Reply: 25.000000
4:	3 1 ***gettrinpos***	Reply: 50.000000
5:	4 1 ***gettrinpos***	Reply: 75.000000

	Command	Description
6:	5 1 ***gettrinpos***	Reply: 93.750000
7:	6 1 ***gettrinpos***	Reply: 100.000000

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

	Command	Description
1:	1 ***filetrinpos*** (p. 621)	Store text file of the record above.

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

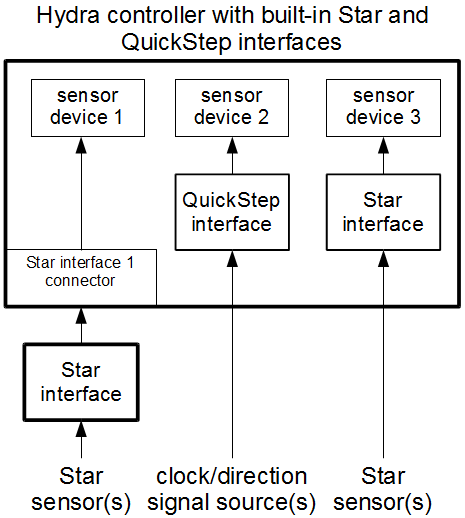
To store the record in a text file on the host PC, first open command line box and change to the target directory. Then type in (*one line*):

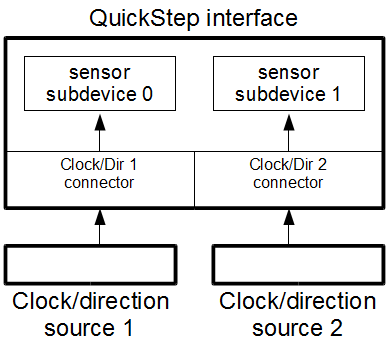
tftp -i 192.168.129.200 GET /ram/captureposition.txt record01.txt

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

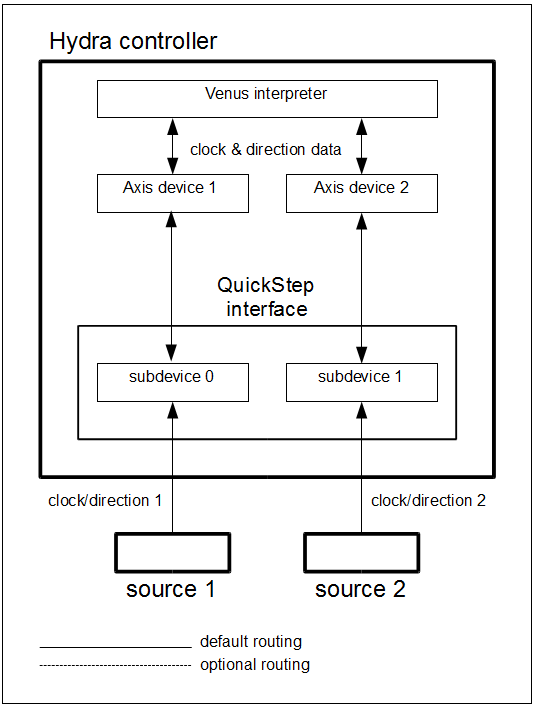
Clock and direction controlled motion function





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

The *sensor subdevices* are invariably assigned to the axes as shown above. The clock/direction function can be accessed under specification of the target axis device. See ***Clock and direction function*** (p. 441) and ***Clock and direction width*** (p. 444) on parameterization.

As opposed to programmed and manual moves, clock and direction operation will stop as soon as a motion limit is

touched. Then ***Motor restart*** (p. 506) must be executed in order to resume operation. Limit switches are ineffective with clock and direction operation so ***Position origin*** (p. 419) and - if existent - ***Motion limits*** must always be determined in advance.

As opposed to programmed and manual moves, clock and direction operation will not work with jerk-optimized motion. Activation of the clock and direction function will automatically change ***Acceleration function*** (p. 437) to standard mode.

Not available with LMT specific firmware rev. 5.1100.

Example

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

Task: Clock/direction source 2 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*** (p. 442)	Disable clock/direction control function at axis 2.
2:	1000 2 ***setcdwidth*** (p. 444)	Set motion distance per incoming clock pulse at axis 2 to 1 µm.
3:	2 2 ***setcdfunc***	Enable clock/direction control function at axis 2. Position alignment is done implicetely prior to activation.

Result: Clock/direction source 2 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 seperate 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

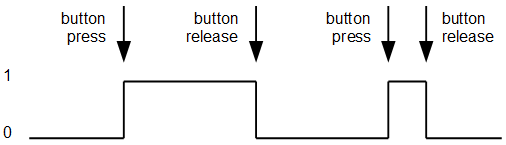
- a digital input pin *or*
- a pushbutton or key at an external CAN device *or*
- a timer *or*
- an internal condition

In this context,

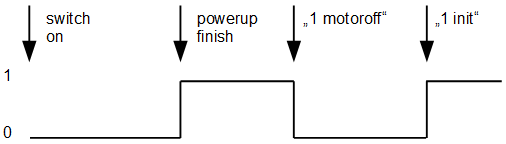
- a pushbutton can be pictured as a digital input pin that goes high whenever the button is pressed and vice versa
- a timer can be pictured as a digital input pin that goes high whenever it has expired (i.e. false whenever running, also if halted before expiration)
- an internal condition can be pictured as a digital input that goes high whenever the respective condition is met

Examples:

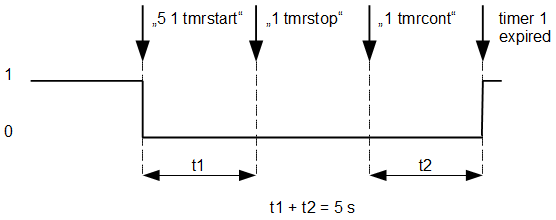
1. pushbutton event state:



2. „axis ready“ condition event state:



3. timer 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
0	custom functions (undocumented)
1	controller
2	CAN joystick 1
3	CAN handwheel 1
4..5	reserved
6	timer group

group	event sources	event index	digital outputs	output index
controller	6 x digital input pin	1...6	6 x digital output pin*	1...6
CAN joystick 1	6 x pushbutton	1...6	8 x LED	1...8
CAN handwheel 1	4 x pushbutton	1...4	1 x LED	1
timer	8 x timer	1...8	-	-

* consisting of 3 user-accessible plus 3 internal outputs

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:

<table>
<tr><th>event source index</th><th>condition</th></tr>
<tr><td>1</td><td>axis powered up
true once the axis has passed the powerup sequence</td></tr>
<tr><td>2</td><td>axis ready
true whenever the Axis status (p. 575) register bit 8 value is 1</td></tr>
<tr><td>3</td><td>axis resting on target
truth table:
<table>
<tr><th>Axis status bit 0</th><th>Axis status bit 5</th><th>Time on target (p. 572)</th><th>condition value</th></tr>
<tr><td>1</td><td>X</td><td>X</td><td>0</td></tr>
<tr><td>0</td><td>0</td><td>> 0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>> 0</td><td>1</td></tr>
<tr><td>0</td><td>X</td><td>= 0</td><td>1</td></tr>
</table></td></tr>
<tr><td>4</td><td>reserved</td></tr>
<tr><td>5</td><td>processing Calibration move (p. 402)</td></tr>
<tr><td>6</td><td>processing Range measure move (p. 425)</td></tr>
<tr><td>7</td><td>reference mark found</td></tr>
</table>

Action command function

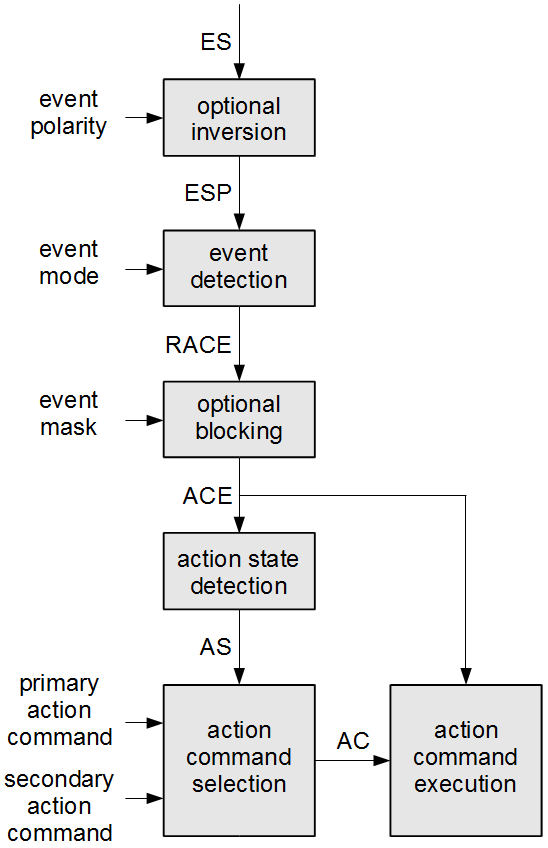
The action command feature allows for automatic execution of user definable Venus command strings (*action commands*, *AC*) upon the occurrence of certain events (*action command events*, *ACE*). For this purpose, each I/O group features two sets of executable Venus command strings, labeled "primary" 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.

Before continuing, note that with LMT microscope stages, the action command facility will possibly be factory preconfigured. In this case, the respective settings should not be altered without factory consultation in order to avoid parameter corruption and indeterminate behaviour.

There are several user options:

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

The following block chart overviews action command operation.



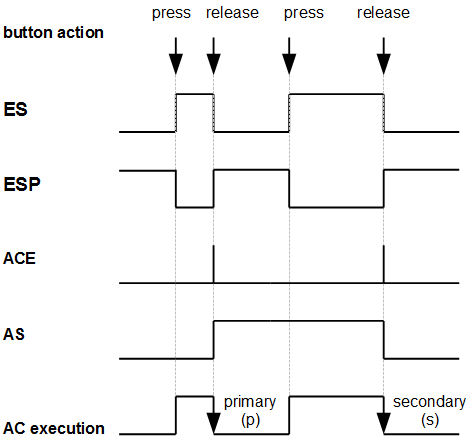
Action command processing flow chart

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 (bidirectional operation), every *ESP* change will make a *RACE (raw action command event)*; otherwise (unidirectional operation), negative (1 to 0) level

transitions will be discarded. Lastly, a blocking unit will suppress *RACE* if *mask* is 0, the result being *ACE*.

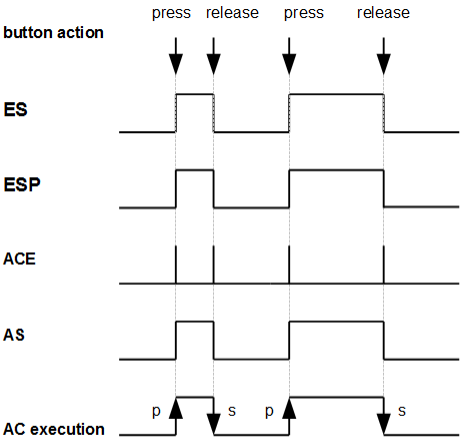
In order to select the command to be executed (primary vs. secondary), the *action state (AS)* is formed out of *ACE*. It changes state whenever an *ACE* occurs. That means with each *ACE*, the primary (*AS* = 0) and secondary (*AS* = 1) command strings for the corresponding *event source* will be executed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

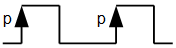
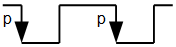
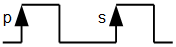
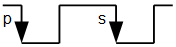
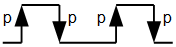
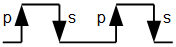
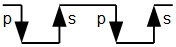
A similar example, but mode being 1 (bidirectional function, command events follow each pushbutton press *and* release) and polarity being 0 (i.e. press => primary, release => secondary command execution):



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

Note that event occurrence to command execution delay will jitter by approximately 25 ms. The time interval between consecutive events must correspond to this (i.e. event rate should be kept well below 40 events per second).

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

*caption:

p ↑ primary action command executed on positive level transition / pushbutton press / timer expiration

p ↓ primary action command executed on negative level transition / pushbutton release / timer start

s ↑ secondary action command executed on positive level transition / pushbutton press / timer expiration

s ↓ secondary action command executed on negative level transition / pushbutton release / timer start

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. 287) and ***Internal event state*** (p. 317) for state query of a single event source. See ***Digital output state*** (p. 258) 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. 284) holds one polarity bit for each joystick pushbutton, bit 0 representing pushbutton 1.

I/O registers per group and entries to access them:

parameter	controller device I/O	axis device I/O	default value
event mode	***Event mode entry*** (p. 273)	***Internal event mode register*** (p. 311)	0
event polarity	***Event polarity entry*** (p. 280)	***Internal event polarity register*** (p. 314)	0
event mask	***Event mask entry*** (p. 266)	*unavailable until further notice*	-1
initial action mask*	***Initial action mask entry*** (p. 289)	***Internal initial action mask register*** (p. 319)	0
event detect	***Event detect register*** (p. 264)	***Internal event detect register*** (p. 309)	-
action state	***Action state register*** (p. 252)	***Internal action state register*** (p. 305)	-

*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. 244) and ***Internal action command entry*** (p. 300) for detailed description.

Action command examples

The Venus parameterization will now be explained by means of some practical applications, also showing I/O access, always presuming that the respective *Event mask register* bit settings are being left at their defaults (1), keeping the corresponding event sources unblocked.

Example 1

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. 247)	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.

	Command	Description
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.
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. 282)	Set event sensitivity to active high at all controller inputs.
2:	54 1 ***setevtmode*** (p. 275)	Set event mode to bidirectional 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 cartesic 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) cartesic 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***	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 ***setevtpol*** (p. 282)	Set event sensitivity to active high at all joystick pushbuttons.
2:	0 2 ***setevtmode*** (p. 275)	Set event mode to unidirectional 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

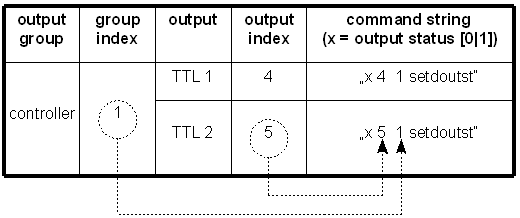
Task:

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:



Then we assign the 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	2	„1 4 1 setdoutst" (set TTL1)	„0 4 1 setdoutst" (clear TTL1)
axis 2	2	axis ready	2	„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 setactcmdint*
2. Secondary: *s* 1 *i g setactcmdint*

Note:

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

	Command	Description
1:	"1 4 1 setdoutst" 0 2 1 ***setactcmdint*** (p. 302)	Assign "set TTL 1" to "axis 1 ready" event.
2:	"0 4 1 setdoutst" 1 2 1 ***setactcmdint***	Assign "clear TTL 1" to "axis 1 not ready" event.
3:	"1 5 1 setdoutst" 0 2 2 ***setactcmdint***	Assign "set TTL 2" to "axis 2 ready" event.
4:	"0 5 1 setdoutst" 1 2 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	2	2	1	1
axis 2	2	axis ready	2	2	1	1

polarity register value = 2

mode register value = 2

	Command	Description
1:	2 1 ***setevtpolint*** (p. 314)	Set event sensitivity to active low at "axis 1 ready" condition.

	Command	Description
2:	2 2 ***setevtpolint***	Set event sensitivity to active low at "axis 2 ready" condition.
3:	2 1 ***setevtmodeint*** (p. 312)	Set event mode to bidirectional function at "axis 1 ready" condition.
4:	2 2 ***setevtmodeint***	Set event mode to bidirectional function at "axis 2 ready" condition.

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

Example 4

Task:

Have axis 2 alternate between two positions periodically. Positions be 10 mm and 20 mm, respectively, and moves to be initiated every 5 seconds.

Preparation:

We utilize timers 1 and 2 for alternating move initiation (multivibrator fashion). At first, we determine the command strings for moving, which is obviously "10 2 nm" and "20 2 nm", and for mutual timer starting (5 seconds running time), which is "5 1 tmrstart" and "5 2 tmrstart", respectively. Then we assign the command strings to the timers 1 and 2. To have the controller launch a move whenever the corresponding timer starts, and have the timers start each other mutually when expiring, we assign as follows:

Action command assignment:

input group	group index (g)	timer	timer index (i)	primary command string (p)	secondary command string (s)
timer	6	timer 1	1	„10 2 nm“	„5 2 tmrstart“
		timer 2	2	„20 2 nm“	„5 1 tmrstart“

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:	"10 2 nm" 0 1 6 ***setactcmd*** (p. 247)	Assign move to 10 mm to primary action of timer 1.
2:	"5 2 tmrstart" 0 2 6 ***setactcmd***	Assign timer 2 start to secondary action of timer 1.
3:	"20 2 nm" 0 2 6 ***setactcmd***	Assign move to 20 mm to primary action of timer 2.
4:	"5 1 tmrstart" 1 2 6 ***setactcmd***	Assign timer 1 start to secondary action of timer 2.

At last, we set the *polarity* and *mode* register values. To have the timer initiate the primary action with every start and the secondary action with every expiration,

both *polarity* and *mode* are to be set to 1 at both event sources.

Mode and polarity register calculation:

input group	group index	timer	timer index	bit value	polarity	mode
timer	6	timer 1	1	1	1	1
		timer 2	2	2	1	1
		timer 3	3	4	any (0)	any (0)
		timer 4	4	8	any (0)	any (0)
		timer 5	5	16	any (0)	any (0)
		timer 6	6	32	any (0)	any (0)
		timer 7	7	64	any (0)	any (0)
		timer 8	8	128	any (0)	any (0)

polarity register value = 2 + 1 = 3 (bits 0 and 1 set)
mode register value = 2 + 1 = 3 (bits 0 and 1 set)

	Command	Description
1:	3 6 ***setevtpol*** (p. 282)	Set event sensitivity to active low at timers 1 and 2.
2:	3 6 ***setevtmode*** (p. 275)	Set event mode to bidirectional function at timers 1 and 2.

The overall configuration of the timers would match AC execution pattern *4b*.

Launching the move sequence now requires the initial start of timer 1 ("5 1 tmrstart"), e.g. by manual command entrance. For start by pushbutton, digital input, or internal event, the initial command could itself be defined as an action command string as well. The preceeding examples have shown how to achieve that.

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 was 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

Note that the mask will merely allow or prevent initial command execution. It will *never* prevent the *ACE* from changing the *AS* from 0 to 1 if the respective event source (*ESP*, involving polarity configuration) is active from the start, regardless of the setting. If this occurs with a setting where both primary and secondary action commands are defined, secondary action command will implicitly be up for next execution (see *Next action* chart below). This is necessary to keep *ES* and *AS* consistent for applications where *event mode* setting is 1 (AC execution patterns *4a* or *4b*); however, it might be confusing in applications where

- *event mode* setting is 0 (AC execution patterns *2a* or *2b*) and
- *initial event mask* setting is 0 and
- the corresponding event source (*ESP*) is active by default

In this case, it may be advisable to swap action commands (primary vs. secondary).

pattern*	initial logical state	next command execution
1a	any	primary
1b	any	primary
2a	0	primary
	1	secondary
2b	0	secondary
	1	primary
3	any	primary
4a	0	primary
	1	secondary
4b	0	secondary
	1	primary

*see *Action command configuration* table above

Next action

Initial event example

Example 5

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. 294) 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.

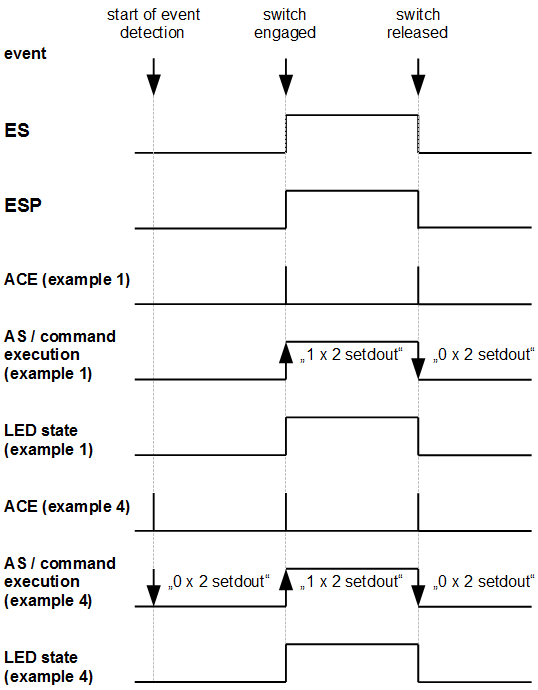
Note:

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

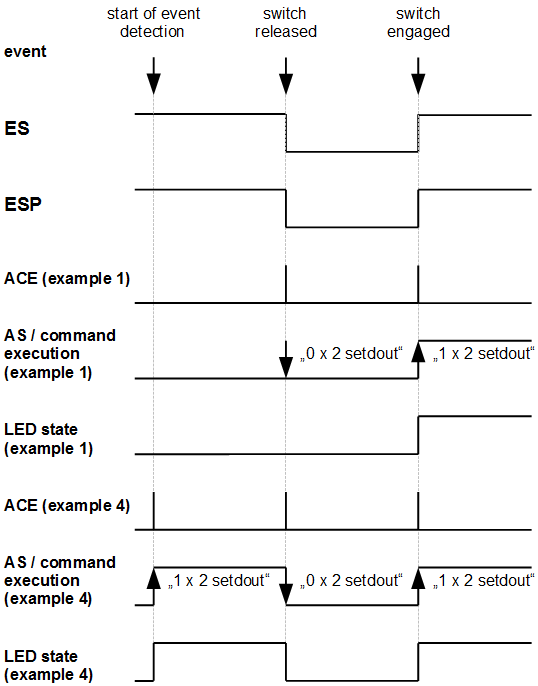
	Command	Description
1:	54 1 ***setinitactmask*** (p. 291)	Set bits 1, 2, 4 and 5 in ***Initial action mask register - controller***.
2:	***save*** (p. 362)	Store configuration.
3:	***reset*** (p. 196)	Reset controller.

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)

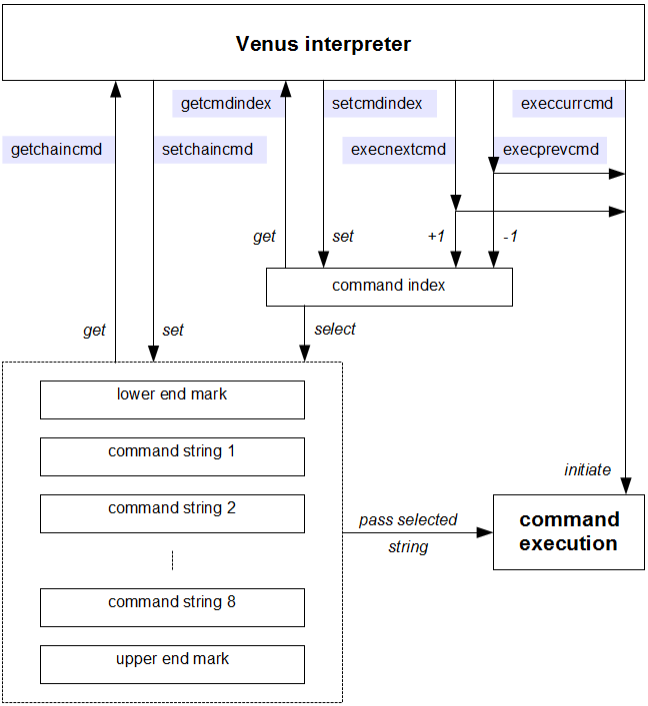


Example 1/4 signal chart (mode = 1, polarity = 0, initial event state = 1)

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. 354) and ***setchaincmd*** (p. 354).
- 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. 355), ***execprevcmd*** (p. 357), or ***execcurrcmd*** (p. 356), 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. 360) and ***setcmdindex*** (p. 361).
- 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.

Before continuing, note that with LMT microscope stages, the command chain facility will possibly be factory preconfigured. In this case, the respective settings should not be altered without factory consultation in order to avoid parameter corruption and indeterminate behaviour.

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).

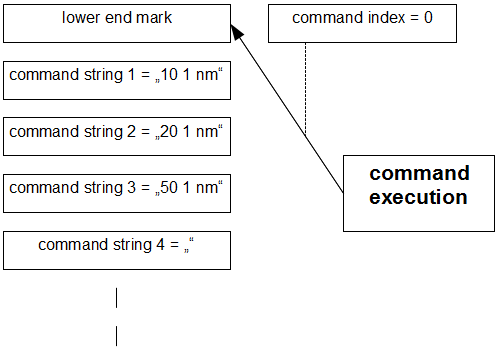
Task: Axis 1 is to be moved to definite stop stations at position coordinates 10 mm, 20 mm, and 50mm, utilizing the command chain feature. Execution of ***execnextcmd*** (p. 355) is to target next stop in positive direction. Execution of ***execprevcmd*** (p. 357) is to target next stop in negative direction.

Preparation: We utilize the ***nm*** (p. 435) 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. 354)	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.

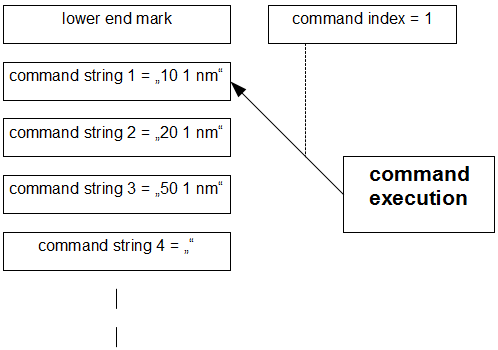


Command chain example step 1

Task: Move axis 1 to first stop.

	Command	Description
1:	3 ***execnextcmd***	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.

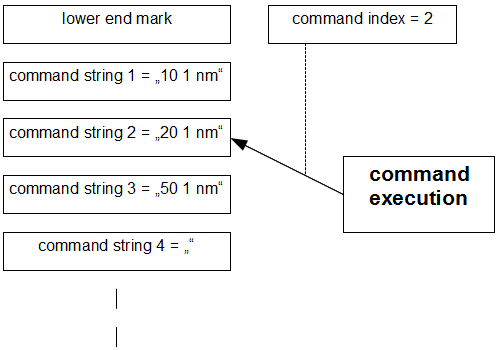


Command chain example step 2

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

	Command	Description
1:	3 ***execnextcmd***	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.

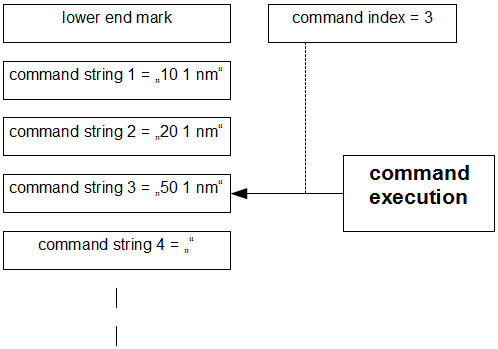


Command chain example step 3

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

	Command	Description
1:	3 ***execnextcmd***	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.

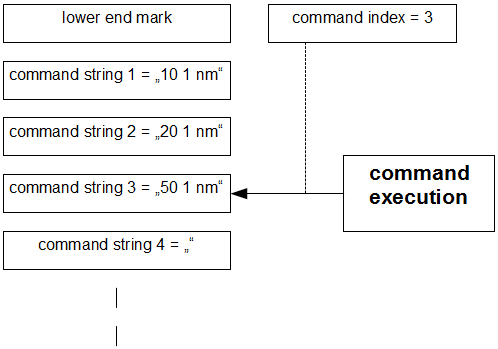


Command chain example step 4

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

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

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

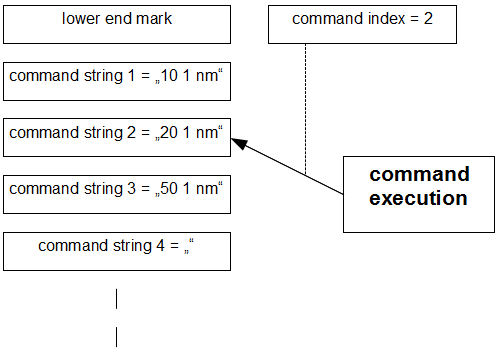


Command chain example step 5

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

	Command	Description
1:	3 ***execprevcmd***	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.

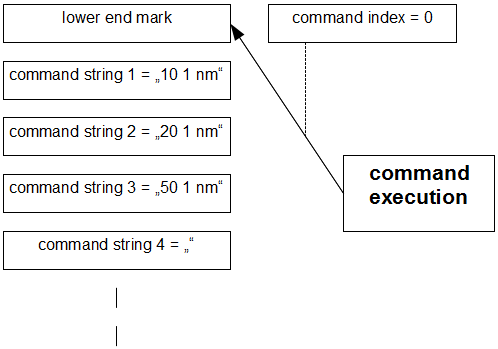


Command chain example step 6

Task: Reset command chain 3 to start anew.

	Command	Description
1:	0 3 ***setcmdindex*** (p. 361)	Set command index to 0.

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

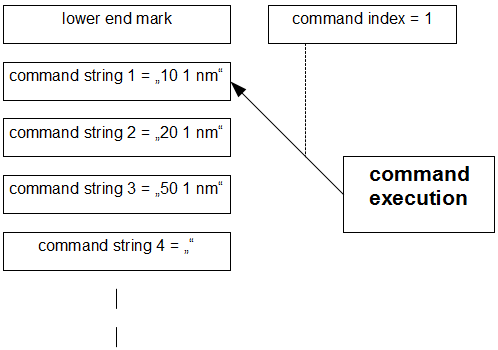


Command chain example step 1

Task: Move axis 1 to first stop.

	Command	Description
1:	3 ***execnextcmd***	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 ***setactcmd*** (p. 247)	Write "3 execnextcmd" to primary action command string of CAN joystick 1 pushbutton 1.
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).

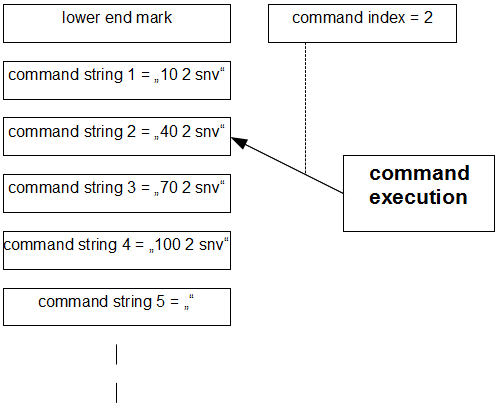
Task: After powerup, ***Velocity*** (p. 224) at axis 2 is to be 40 mm/s. It is to be raised at steps of 30 mm/s up to 100 mm/s max. whenever handwheel pushbutton 1 is being pushed. It is to be lowered at steps of 30 mm/s down to 10 mm/s min. whenever handwheel pushbutton 2 is being pushed.

Preparation: We utilize ***snv*** (p. 225) 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. 354)	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.
5:	"100 2 snv" 4 5 ***setchaincmd***	Assign "100 2 snv" to command string 4 of command chain 5.
6:	2 5 ***setcmdindex*** (p. 361)	Set command index of command chain 5 to 2.
7:	"5 execnextcmd" 0 1 3 ***setactcmd*** (p. 247)	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*** (p. 362)	Save all Venus parameters.
10:	***reset*** (p. 196)	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

Rotatory axes

For rotatory systems, there are a few special functions to ease operation. These are in detail:

- ***Position cycle*** (p. 591)
- ***Axis alignment*** (p. 439)
- ***getnrev*** (p. 588)
- ***getrev*** (p. 582)

These extra functions are not available with LMT specific firmware rev. 5.1100.

An example to show how to use the functions:

Example

Preconditions: Axis 1 be a rotatory axis with ***Pitch*** (p. 399) set to 2.0 mm, making one motor axis turn. The axis provide a normally open reference switch to relate to, wired to the cal limit switch input, and a specific position to return to repeatedly (called "load position"). The distance between the reference switch and the load position be 0.15 axis turns. Revolution counter (s. ***getnrev*** (p. 588)) be 0.

Task: Move axis to load position, then repeatedly perform a procedure where axis 1 is to execute 13.25 axis turns (positive direction), then return to the load position by the shortest way possible.

Preparation: ***Position cycle*** (p. 591) is to equal ***Pitch*** to have the position revolve with each motor axis turn. Distance to be covered by the rotational move is 13.25 * ***Pitch*** = 26.5 mm. Distance between the reference switch and the load position is 0.15 * ***Pitch*** = 0.3 mm.

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

	Command	Description
1:	2 1 ***setposcycle*** (p. 592)	Set ***Position cycle*** at axis 1 to 2 mm.
2:	4 0 1 ***setsw*** (p. 605)	Configure cal/reference switch input to be generally disarmed, armed only while a ***Calibration move*** (p. 402) is performed.

2. Calibration.

	Command	Description
1:	1 ***ncal*** (p. 403)	Search axis 1 reference switch.
2:	0.3 1 ***nm*** (p. 435)	Move axis 1 to load position.
3:	0 1 ***setnpos*** (p. 420)	Shift axis 1 origin to load position.
4:	1 ***np*** (p. 589)	Inquire axis 1 position.
5:	1 ***getnrev***	Inquire axis 1 revolution counter.

Result: Axis 1 is located at load position. Inquiry will show position: 0, rev. counter: 0.

3. Normal operation, performed repeatedly.

	Command	Description
1:	26.5 1 ***nr*** (p. 457)	Have axis 1 rotate by 13.25 axis turns (positive direction).

Result: Axis 1 is located at position 0.5 mm.

	Command	Description
1:	1 ***np***	Inquire axis 1 position.
2:	1 ***getnrev***	Inquire axis 1 revolution counter.

Result: Inquiry will show position: 0.5, rev. counter: 13.

	Command	Description
1:	0 0 1 ***align*** (p. 439)	Have axis 1 return to load position by the shortest way.

Result: Axis 1 is located at load position.

	Command	Description
1:	1 ***np***	Inquire axis 1 position.
2:	1 ***getnrev***	Inquire axis 1 revolution counter.

Result: Inquiry will show position: 0.0, rev. counter: 13.

	Command	Description
1:	0 1 ***setnpos***	Shift origin by covered distance (13 rev.) in order to clear revolution counter.
2:	1 ***np***	Inquire axis 1 position.
3:	1 ***getnrev***	Inquire axis 1 revolution counter.

Result: Inquiry will show position: 0.0, rev. counter: 0.
Sequence can be repeated ad lib.

Firmware history

Short feature history since rev. 4.0000

Rev. 4.0000

- Jerk-optimized motion (extra option)
- Scale correction (extra option)
- ***FRT parameters*** (p. 541)
- Limit stop function (prevents nominal position from exceeding limits with manual moves)
- ***getdoutst*** (p. 260)

Rev. 4.1300

- Position teach-in:
  - ***Position latching*** (p. 181)
  - ***Position storage*** (p. 188)
  - ***Position restorage*** (p. 184)
- ***Stop move*** (p. 459)
- Emergency-off state now irreversible if caused by position sensor detection failure in closed loop mode (machine error code 101)
- Motor optimization reworked completely (s. chapter ***Motor commutation handling***)

Rev. 4.1400

- ***MAC address*** (p. 168)

Rev. 4.2000

- ***Servo control*** (p. 565), entry index 13. Factory setting is 0; user setting is mandatory for motor operation with linear, AC and DC motors

Rev. 4.3000

- Onboard trigger facility

Rev. 4.4000

- Onboard trigger flip-flop option
- Reference mark distance decoding facility
- ***Servo control*** (p. 565) , entry index 14

Rev. 4.5000

- Reference mark detection reworked and distance decoding issues fixed

Rev. 5.0000

- **Important notice regarding controllers with new flash devices: First supporting firmware - "downdates" of controllers with factory firmware from 5.0000 on to firmware revisions lower than 5.0000 will cause irreparable boot failure**

Rev. 5.0300

- definable ***Reference window*** (p. 550) for distance decoding

Rev. 5.1000

- DeltaStar table trigger facility
- Rotational axis facility
- Extended position range with all acceleration functions (approx. +/- 4.6 * 10^{12} mm)
- Extended ***Switch configuration*** (p. 604) options

Devices

Controller [Device 0]

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

Controller

ControllerIO

Dynamics

InputOutput

Interpreter

Motion

Status and position

Axis [Device 1..2]

Controller

Dynamics

InputOutput

Interpreter

Manual operation

Mechanic

Mechanic setup

Motion

Motor

Safety functions

Sensoric

Servo

Status and position

Switches and reference mark

Trigger

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

Dynamics

InputOutput

Interpreter

Manual operation

Mechanic

Mechanic setup

Motion

Motor

Safety functions

Sensoric

Servo

Status and position

Switches and reference mark

Trigger

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
LOWQUALTITY=4
NOTMAPPED=8
NOTCONNECTED=16
NOMT=32
POSOUTOFRANGE=64
ILLSUBDEVICE=128
any combination of the above values are possible

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

Syntax

Reg.Ex:
-?[01234567890]+

Examples :
1001
512
-3165

sensortypes

Description

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

Properties

Type: string

---

nidentify

Returns the ***Controller identification***.

Syntax

{device} ***nidentify***

Reply

[identifystring]

Examples

Example

	Command	Description
1:	3 ***nidentify***	Return ***Controller identification*** of device 3.

Controller name

read-only

Hydra name: "hydra"; supplemented by name of a special firmware build if indicated.

Commands

Properties

Type: string

---

identify

Returns the ***Controller name***.

Syntax

identify

Reply

[identifystring]

Examples

Example

	Command	Description
1:	***identify***	Returns ***Controller name***.

Controller version

read-only

CPU firmware revision.

Commands

Properties

Type: double

---

version (getversion)

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

Syntax

version

Reply

[version]

Examples

Example

	Command	Description
1:	***version***	Returns ***Controller version***.

CPU temperature

read-only

CPU temperature.

Commands

Properties

Type: double
Unit: °C

---

getcputemp

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

Syntax

getcputemp

Reply

[temperature]

Examples

Example

	Command	Description
1:	***getcputemp***	Returns ***CPU temperature***.

Device class

Type of a device specified by device number.

Commands

Properties

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

---

getdeviceclass

Returns the ***Device class***.

Syntax

[devicenr] ***getdeviceclass***

Reply

[class code]

Examples

Example

	Command	Description
1:	2 ***getdeviceclass***	Return ***Device class*** of devicce 2.

Device count

read-only

Number of devices minus the controller device.

Commands

Properties

Type: int

---

getaxc

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

Syntax

getaxc

Reply

[axiscount]

Examples

Example

	Command	Description
1:	***getaxc***	Returns ***Device count***.

MAC address

read-only

Hydra controller specific media access control address.

Valid since ***Controller version*** (p. 163) 4.1400.

Commands

Properties

Type: string

---

getmacadr

Returns the ***MAC address***.

Syntax

getmacadr

Reply

[address]

Examples

Example

	Command	Description
1:	***getmacadr***	Inquire ***MAC address***.

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	I²t overflow
30	CAN controller error
31	CAN initialization error
32	CAN device version mismatch
33	CAN joystick offset range violation
40	Trigger input frequency violation
100	EEPROM checksum error
101	No sensor available [1)]
103	sensor position invalid
104	EEPROM write error
110	Emergency while commuting [2)]
111	Master invalid [3)]
112	Safety invalid [3)]
113	Motor form invalid [4)]

1) Usually occurs due to one of the following:

- missing sensor connection
- electrical error, e.g. cable break
- electrical or mechanical sensor misalignment
- scale flaw or misalignment

When closed loop mode is selected, will result in permanent emergency-off state - only to be reversed by ***Reset*** (p. 196). If so, also indicated by ***Axis status*** (p. 575), bit 15.

2) Usually occurs when Emergency-off switch is active from the start of or being engaged during a motor commutation procedure. Will result in permanent emergency-off state - only to be reversed by ***Reset***. If so, also indicated by ***Axis status***, bit 15.

3) Effective with customized systems only. Will result in permanent emergency-off state - only to be reversed by ***Reset***. If so, also indicated by ***Axis status***, bit 15.

4) No piezo motor support. Will result in permanent emergency-off state - only to be reversed by ***Reset***. If so, also indicated by ***Axis status***, bit 15.

Note that although ***gme*** (p. 173) is associated with axis devices, there is only *one* overall machine error stack. Individual axis access is not provided; device index is involved for compatibility purpose only. This means command execution will always return the most recent error code from the stack, regardless of whether the command target device is identical to the error-affected device or not. To identify both error code and device, ***gme*** should be used with device index 0. The value returned then combines error code and device index according to the formula:

value = 1000 * i + c

where *i* would be the device index and *c* the error code.

For instance, if a following error has occurred at axis 2,

2 ***gme*** will pop the stack and return 13.

1 ***gme***, too, will pop the stack and return 13.

0 ***gme***, in contrast, will pop the stack and return 2013.

Commands

Properties

Name	Type
[devicenr]	int
[machine error]	int

---

gme (getmerror)

Returns the most recent ***Machine error*** (p. 170) code and removes it from the machine error stack. Returns 0 if the stack is empty. The command ***merrordecode*** (p. 368) returns the error description of the code in a string.

Syntax

[devicenr] ***gme***

Reply

[machine error]

Examples

	Command	Description
1:	0 ***gme***	Returns the machine error code and affected device according to the given formula, 0 if no error messages pending

Number format


storable

Return format specification of floating point Venus parameter values.

Commands

Properties

Name	Type	Description
[format base]	int	<table><tr><th>value</th><th>format</th></tr><tr><td>0</td><td>standard format: 6 decimal places after point, no leading blanks, no leading "+"</td></tr><tr><td>1</td><td>custom format according to *precision*, *width*, and *sign* parameters</td></tr></table>
[precision]	int	number of decimal places after point (valid if *format base* parameter set to 1)
[width]	int	number of decimal places before point (valid if *format base* parameter set to 1); matched to number of significant decimals if 0
[sign]	int	leading sign (valid if *format base* parameter set to 1)<table><tr><th>value</th><th>image</th></tr><tr><td>0</td><td>no leading "+" with positive values</td></tr><tr><td>1</td><td>leading "+" with positive values</td></tr></table>

getformat

Returns the current setting of the ***Number format***.

Syntax

getformat

Reply

[sign] [width] [precision] [format base]

Examples

Example

	Command	Description
1:	***getformat***	Returns ***Number format***.

---

setformat

Sets the ***Number format***.

Syntax

[sign] [width] [precision] [format base] ***setformat***

Examples

Example 1

Task: Set ***Number format*** to

- 6 decimal places before point
- 9 decimal places after point
- leading "+" with positive values

	Command	Description
1:	1 6 9 1 ***setformat***	Set format as specified.

Parameter stack (Controller)

read-only

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

Commands

Properties

Type: int

---

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:	***clear***	Clear ***Parameter stack***.

---

gsp

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

Syntax

gsp

Reply

[stackpointer]

Examples

Example

Preconditions: Parameter stack is empty.

	Command	Description
1:	***gsp*** (p. 179)	Returns 0.
2:	0 2	Push 2 parameter values on stack without entering a command string.
3:	***gsp***	Returns 2.
4:	***clear*** (p. 178)	Clears stack.
5:	***gsp***	Returns 0 again.

Position latching

Facility that latches a given all-axes coordinate in controller RAM. After latching, the coordinate is volatile and pending for ***Position storage*** (p. 188). Position can be selected as one out of the following:

- momentary nominal coordinate (s. ***latchscalepos*** (p. 182))
- momentary scale coordinate (s. ***latchnompos*** (p. 182))
- manually entered coordinate (s. ***latchextpos*** (p. 183))

Note that the physical taught-in positions always relate to the initial (powerup) coordinate; use with incremental position encoders is therefore not recommendable.

Note that the physical taught-in positions will shift according to the ***Position origin*** (p. 419) applied.

Valid since ***Controller version*** (p. 163) 4.1300.

Commands

Properties

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

latchnompos

Latches momentary nominal coordinate.

Syntax

latchnompos

Examples

Example

	Command	Description
1:	***latchnompos***	Latches momentary nominal coordinate.

latchscalepos

Latches momentary scale coordinate.

Syntax

latchscalepos

Examples

Example

	Command	Description
1:	***latchscalepos***	Latches momentary scale coordinate.

latchextpos

Latches given coordinate.

Syntax

[position X] [position Y] ***latchextpos***

Examples

Example

	Command	Description
1:	10.0 20.0 ***latchextpos***	Latches coordinate (10 mm ; 20 mm).

Position restorage (Controller)

Indexed restorage of coordinate stored at the given index in the controller EEPROM (s. ***Position storage*** (p. 188)).

Note that the physical taught-in positions always relate to the initial (powerup) coordinate; use with incremental position encoders is therefore not recommendable.

Note that the physical taught-in positions will shift according to the ***Position origin*** (p. 419) applied.

Valid since ***Controller version*** (p. 163) 4.1300.

Commands

Properties

Name	Type	Description
[index]	int	range: 0...99
[result]	int	0 upon success -1 if *index* out of bounds -2 if no position stored at *index*

---

restorepos

Induces vectorial move to coordinate stored at the given index.

Void if - at the given index - no coordinate has been stored so far or if the coordinate has been cleared beforehand by application of ***clearlp*** (p. 190).

Syntax

[index] ***restorepos***

Reply

[result]

Examples

Title

	Command	Description
1:	3 ***restorepos***	Restores coordinate stored at index 3.

All axes will move to coordinate stored at index 3. Action will be virtually vectorial.

Position restorage (Axis)

Indexed restorage of coordinate stored at the given index in the controller EEPROM (s. ***Position storage***).

Note that the physical taught-in positions always relate to the initial (powerup) coordinate; use with incremental position encoders is therefore not recommendable.

Note that the physical taught-in positions will shift according to the ***Position origin*** applied.

Valid since ***Controller version*** 4.1300.

Commands

Properties

Name	Type	Description
[index]	int	range: 0...99
[result]	int	0 upon success -1 if *index* out of bounds -2 if no position stored at *index*

---

nrestorepos

Induces move to coordinate stored at the given index.

Void if - at the given index - no coordinate has been stored so far or if the coordinate has been cleared beforehand by application of ***clearlp***.

Syntax

[index] {device} ***nrestorepos***

Reply

[result]

Examples

Example

	Command	Description
1:	2 1 ***nrestorepos***	Restores axis 1 position stored at index 2.

Axis 1 will move to respective portion of coordinate stored at index 2.

Position storage

Immediate indexed storage of coordinate buffered by last application of ***Position latching*** (p. 181) , allowing for permanent deposition of up to 100 different coordinates in the controller EEPROM. The position can later be restored by use of ***Position restorage (Axis)*** or ***Position restorage (Controller)***.

Note that the physical taught-in positions always relate to the initial (powerup) coordinate; use with incremental position encoders is therefore not recommendable.

Note that the physical taught-in positions will shift according to the ***Position origin*** (p. 419) applied.

Note that the coordinates will not be stored in one of the parameter files, but in a separate text file. As opposed to the Venus parameters, this file will not be touched by ***Configuration storage*** (p. 362) .

Valid since ***Controller version*** (p. 163) 4.1300.

Commands

Properties

Name	Type	Description
[index]	int	range: 0...99
[result]	int	-1 if *index* out of bounds; otherwise 0

storelpnv

Returns the current setting of the ***Position storage***.

Syntax

[index] ***storelpnv***

Reply

[result]

storelp

Stores currently latched coordinate in controller EEPROM at given *index* if the latter is within bounds, thus overwriting former memory content.

Syntax

[index] ***storelp***

Reply

[result]

Examples

Example

	Command	Description
1:	3 ***storelp***	Stores currently latched coordinate at index 3.

clearlp

Clears coordinate stored in controller EEPROM at given *index* if the latter is within bounds.

Syntax

[index] ***clearlp***

Reply

[result]

Examples

Example

	Command	Description
1:	2 ***clearlp***	Clears coordinate stored at index 2.

ProductID

not storable

Commands

Properties

Type: string

---

getproductid

Returns the current setting of the ***ProductID***.

Syntax

getproductid

Reply

[product]

Release code

read-only

Release code for extra options read from code file. Empty if file is missing.

Valid since ***Controller version*** (p. 163) 4.0000.

Commands

Properties

Type: string

---

getcode

Returns ***Release code***.

Syntax

getcode

Reply

[code]

Examples

Example

	Command	Description
1:	***getcode***	Returns ***Release code***.

Released options

read-only

Bit-coded register showing which extra options are available (corresponding bit set) and which are not (corresponding bit cleared).

Valid since ***Controller version*** (p. 163) 4.0000.

bit	description
0	position correction
1	jerk-reduced motion

Commands

Properties

Type: int

---

getoptions

Returns ***Released options***.

Syntax

getoptions

Reply

[options]

Examples

Example

	Command	Description
1:	***getoptions***	Returns ***Released options***.

Reset

Controller hardware reset.

Commands

---

reset

Initiates ***Reset***.

Volatile data will be lost. Execute ***save*** or ***csave*** (p. 363) before execution if needed.

Syntax

reset

Examples

Example

	Command	Description
1:	***reset***	Initiate ***Reset***.

Serial number

read-only

Serial number of controller hardware.

Commands

Properties

Type: string

---

getserialno

Returns ***Serial number***.

Syntax

getserialno

Reply

[number]

Examples

Example

	Command	Description
1:	***getserialno***	Returns ***Serial number***.

Software type

read-only

Commands

Properties

Type: string

---

getsoftwaretype

Returns the ***Software type***.

Syntax

getsoftwaretype

Reply

[type]

Stored position

read-only

Coordinate recently stored at given index by ***Position storage*** (p. 188) .

Commands

Properties

Name	Type
[index]	int
[position X]	double
[position Y]	double
[result]	int

---

getlp

Returns the actual ***Stored position***.

Syntax

[index] ***getlp***

Reply

[position X] [position Y] [result]

Examples

Example

	Command	Description
1:	1 ***getlp***	Returns the actual ***Stored position*** at index 1.

Supply voltage

read-only

Supply voltage and type of the motor power stage.

Commands

Properties

Name	Type
[value]	double
[type]	string

---

getusupply

Returns the ***Supply voltage***.

Syntax

getusupply

Reply

[value] [type]

Examples

Example

	Command	Description
1:	***getusupply***	Returns ***Supply voltage***.

Time of day

read-only

Greenwich mean time.

Commands

Properties

Type: string

---

gettime

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

Syntax

gettime

Reply

[time]

Examples

Example

	Command	Description
1:	***gettime***	Returns ***Time of day***.

Version


read-only

Same as ***Controller version*** (p. 163) . Needs a specified device index, but returns the same result. Compatibility purpose only.

Commands

Properties

Type: double

---

nversion (getnversion)

Returns the actual ***Version***.

Syntax

{device} ***nversion***

Reply

[versionnumber]

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

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 ***getnetpara***	Returns controller IP address.

---

setnetpara

Configures the ***Network***.

Note that a change of the ***Network*** setting will not take effect immediately. The requested configuration will only be effective on next ***Reset*** (p. 196) .

Syntax

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

Examples

Example

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

	Command	Description
1:	"192.168.129.200" 0 ***setnetpara***	Set IP address.
2:	***save***	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 RS232-2/USB option built in, port 2 must always be switched off.

Commands

Properties

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

---

setbaudrate

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

Syntax

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

Examples

Example

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

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

---

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


storable

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).

The setting must comply with the following restriction:

Acceleration <= 1/60 s * ***Jerk*** (p. 216)

Commands

Properties

Type: double
Unit: mm/s²
Default: 60.0
Range: 0.001 ... 500000.0

---

gna (getnaccel)

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

Syntax

{device} ***gna***

Reply

[acceleration]

Examples

Example

	Command	Description
1:	2 ***gna***	Returns ***Acceleration*** at axis 2.

---

sna (setnaccel)

Sets the ***Acceleration***.

Syntax

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

Examples

Example

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

Dynamic state

read-only

Compact diagnosis information on dynamic controller data. Regard the nonuniform measurement units.

Not supported with LMT specific firmware rev. 5.1100.

Commands

Properties

Name	Type	Unit	Description
[nominal position]	double	mm	s. ***Device position*** (p. 588), with ***Position display selection*** (p. 594) set to 0
[scale position]	double	mm	s. ***Device position***, with ***Position display selection*** set to 1
[deviation]	long long	nm	scale position aberration from nominal position
[executive position]	long long	nm	position equivalent of set motor angle
[nominal acceleration]	long long	nm/T² *	acceleration derived from nominal position * T = 0.25 ms
[nominal velocity]	long long	nm/T *	velocity derived from nominal position * T = 0.25 ms
[measured velocity]	double	mm/16T *	velocity derived from scale position * T = 0.25 ms
[pwm supply]	int	-	equivalent of Z-stage PWM duty cycle
[relative load angle]	double	**	angle equivalent of motor load ** normalized to 1.0 / phasewidth
[cycle number]	long long	-	machine cycle based time equivalent

gds

Returns the actual ***Dynamic state***.

Syntax

{device} ***gds***

Reply

[nominal position] [scale position] [deviation] [executive position] [nominal acceleration] [nominal velocity] [measured velocity] [pwm supply] [relative load angle] [cycle number]

Examples

Example

	Command	Description
1:	2 ***gds***	Returns ***Dynamic state*** at axis 2.

Jerk


storable

Jerk limitation for interrupted programmed moves.

Valid since ***Controller version*** (p. 163) 4.0000.

Effective with jerk-optimized motion only.

The setting must comply with the following restriction:

Jerk >= 60 s^{-1} * ***Acceleration*** (p. 212)

Commands

Properties

Type: double
Unit: mm/s^3

---

gnj (getnjerk)

Returns the current setting of the ***Jerk***.

Syntax

{device} ***gnj***

Reply

[jerk]

---

snj (setnjerk)

Sets the ***Jerk***.

Syntax

[jerk] {device} ***snj***

Stop deceleration


storable

Deceleration for immediate halt used upon

- touch of either end switch during any move
- ***Move abortion*** (p. 448)
- *Ctrl*+*C* shortcut

Replaced by ***Acceleration*** (p. 212) setting for braking slope calculation whenever the latter is currently higher. This does not affect the parameter setting, though.

Commands

Properties

Type: double
Unit: mm/s²
Default: 60.0
Range: 0.001 ... 500000.0

---

ssd (setstopdecel)

Sets the ***Stop deceleration***.

Syntax

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

Examples

Example

	Command	Description
1:	2000 1 ***ssd***	Set ***Stop deceleration*** 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 ***gsd***	Return ***Stop deceleration*** at axis 2.

Vector acceleration


storable

Vectorial acceleration used with ***Vector move*** (p. 460) .

Commands

Properties

Type: double
Unit: mm/s²
Default: 1.0
Range: 0.001 ... 500000.0

---

ga (getaccel)

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

Syntax

ga

Reply

[acceleration]

---

sa (setaccel)

Sets the ***Vector acceleration***.

Syntax

[acceleration] ***sa***

Examples

Example

	Command	Description
1:	1000 ***sa***	Set ***Vector acceleration*** to 1 m/s².

Vector velocity


storable

Vectorial velocity used with ***Vector move*** (p. 460) .

Commands

Properties

Type: double
Unit: mm/s
Default: 1.0
Range: 0.00001 ... 10000.0

---

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]

Examples

Example

	Command	Description
1:	***gv***	Returns ***Vector velocity***

Velocity


storable

Velocity used by all programmed moves but

- ***Calibration move*** (p. 402)
- ***Range measure move*** (p. 425)
- ***Reference move*** (p. 453)
- ***Vector reference move***

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

Commands

Properties

Type: double
Unit: mm/s
Default: 20.0
Range: 0.00001 ... 10000.0

---

gnv (getnvel)

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

Syntax

{device} ***gnv***

Reply

[velocity]

Examples

Example

	Command	Description
1:	1 ***gnv***	Returns ***Velocity*** at axis 1.

---

snv (setnvel)

Sets the ***Velocity***.

Syntax

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

Examples

Example

	Command	Description
1:	50 1 ***snv***	Set ***Velocity*** 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 command 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. 276)

s. ***Event polarity register - CAN handwheel 1* (p. 283)

s. ***Action command - CAN handwheel 1 (secondary) (p. 230)

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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 handwheel 1 (secondary)


storable

Set of user definable command strings with one string assigned to each input. On certain events at any of 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. 228) . 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. 276)

s. ***Event polarity register - CAN handwheel 1* (p. 283)

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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)


storable

Set of user definable command 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. 277)

s. ***Event polarity register - CAN joystick 1* (p. 284)

s. ***Action command - CAN joystick 1 (secondary) (p. 234)

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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
5	[pushbutton 5]	string
6	[pushbutton 6]	string

Action command - CAN joystick 1 (secondary)

storable

Set of user definable command strings with one string assigned to each input. On certain events at any of 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. 232) . 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. 277)

s. ***Event polarity register - CAN joystick 1* (p. 284)

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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
5	[pushbutton 5]	string
6	[pushbutton 6]	string

Action command - controller (primary)

storable

Set of user definable command 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. 278)

s. ***Event polarity register - controller* (p. 285)

s. ***Action command - controller (secondary) (p. 238) .

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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	[input 1]	string
2	[input 2]	string
3	[input 3]	string
4	[input 4]	string
5	[input 5]	string
6	[input 6]	string

Action command - controller (secondary)


storable

Set of user definable command strings with one string assigned to each input. On certain events at any of the digital controller inputs, the respective command string is executed alternately with the respective command string from ***Action command - controller (primary)*** (p. 236) . 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. 278)

s. ***Event polarity register - controller* (p. 285)

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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	[input 1]	string
2	[input 2]	string
3	[input 3]	string
4	[input 4]	string
5	[input 5]	string
6	[input 6]	string

Action command - timer (primary)


storable

Set of user definable command strings executed on certain timed events, with one string assigned to each timer. Execution conditions are:

internal event mode *	secondary internal action command defined ***	internal event polarity **	execution upon
0	yes	0	every other expiration
0	yes	1	every other start
0	no	0	every expiration
1	yes		
0	no	1	every start
1	yes		
1	no	any	any change of expiration state

*s. ***Event mode register - timer*** (p. 279)

s. ***Event polarity register - timer* (p. 286)

s. ***Action command - timer(secondary).

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

See also ***Timer entry*** (p. 337) for information on how to operate a timer.

Properties

i	Name	Type
1	[timer 1]	string
2	[timer 2]	string
3	[timer 3]	string
4	[timer 4]	string
5	[timer 5]	string
6	[timer 6]	string
7	[timer 7]	string
8	[timer 8]	string
9	[timer 9]	string
10	[timer 10]	string
11	[timer 11]	string
12	[timer 12]	string
13	[timer 13]	string
14	[timer 14]	string
15	[timer 15]	string
16	[timer 16]	string

Action command - timer (secondary)

storable

Set of user definable command strings with one string assigned to each timer. On certain events from any timer, the respective command string is executed alternately with the respective command string from ***Action command - timer (primary)*** (p. 240) . Execution conditions are:

internal event mode *	internal event polarity **	execution upon
0	0	every other expiration
0	1	every other start
1	0	every start
1	1	every expiration

*s. ***Event mode register - timer*** (p. 279)

s. ***Event polarity register - timer* (p. 286)

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

See also ***Timer entry*** (p. 337) for information on how to operate a timer.

Properties

i	Name	Type
1	[timer 1]	string
2	[timer 2]	string
3	[timer 3]	string
4	[timer 4]	string
5	[timer 5]	string
6	[timer 6]	string
7	[timer 7]	string
8	[timer 8]	string
9	[timer 9]	string
10	[timer 10]	string
11	[timer 11]	string
12	[timer 12]	string
13	[timer 13]	string
14	[timer 14]	string
15	[timer 15]	string
16	[timer 16]	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 *event 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*.

group index	event index *	command index	target command string set
0	-	-	*reserved for application specific functions*
1	1...6	0	***Action command - controller (primary)*** (p. 236)
1	1...6	1	***Action command - controller (secondary)*** (p. 238)
2	1...6	0	***Action command - CAN joystick 1 (primary)*** (p. 232)
2	1...6	1	***Action command - CAN joystick 1 (secondary)*** (p. 234)
3	1...4	0	***Action command - CAN handwheel 1 (primary)*** (p. 228)
3	1...4	1	***Action command - CAN handwheel 1 (secondary)*** (p. 230)
4	-	-	*reserved*
5	-	-	*reserved*
6	1...8	0	***Action command - timer (primary)*** (p. 240)
6	1...8	1	***Action command - timer (secondary)*** (p. 242)

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

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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 digital I/O processing" for general information on how digital inputs, CAN device pushbuttons and timers are processed.

Commands

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

---

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 ***getactcmd***	Returns secondary ***Action command*** of controller input 4.

---

setactcmd

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

If the *value* itself contains another string (like a filename), it can be entered with outer (') and inner (") quotation marks as follows:

'"innerstring"outerstring'

For instance: '"This is a string!" 0 setvarstring'

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 ***setactcmd***	Set primary action for pushbutton 3.
2:	"0 2 setcloop" 0 4 2 ***setactcmd***	Set primary action for pushbutton 4.
3:	"1 1 setcloop" 1 3 2 ***setactcmd***	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.

Action state register


not storable

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
4	-	-	*reserved*
5	-	-	*reserved*
6	timer	0...7	timers 1...8

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

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Commands

Properties

Name	Type
[group index]	int
[value]	int

---

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.

---

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. 252) .

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

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

DAC voltage

not storable

Voltage set at specified digital/analog converter (DAC) output.

Commands

Properties

Name	Type	Description
[index]	int	<table><tr><th>value</th><th>pin selection*</th></tr><tr><td>1</td><td>DAC_OUT1</td></tr><tr><td>2</td><td>DAC_OUT2</td></tr><tr><td>3</td><td>DAC_OUT3</td></tr><tr><td>4</td><td>DAC_OUT4</td></tr></table> * according to hardware manual
[voltage]	double	range: 0.0 V ... 3.3 V

---

setdac

Sets the ***DAC voltage*** at the specified output. Note that the value set will be limited to the voltage range of 0.0 V ... 3.3 V.

Syntax

[voltage] [index] ***setdac***

Examples

Example

	Command	Description
1:	1.7 1 ***setdac***	Sets ***DAC voltage*** at DAC_OUT2 to 1.7 V.

---

getdac

Returns the current setting of the ***DAC voltage*** at the specified output. Note that the value returned is matched to the DAC resolution, so it might aberrate slightly from the value preceedingly applied by ***setdac*** (p. 256).

Syntax

[index] ***getdac***

Reply

[voltage]

Examples

Example

	Command	Description
1:	3 ***getdac***	Returns ***DAC voltage*** at DAC_OUT4.

Digital output state

not storable

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**
TTL output	0 => low level 1 => high level 2 => periodically alternating level
Open drain output	0 => drain isolated from ground 1 => drain connected to ground 2 => periodically alternating state
LED	0 => off 1 => on

Outputs supported at the time:

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

* hardware I/O, invariably configured as output

** special purpose, not user accessible

Commands

Properties

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

---

getdoutst

Returns the actual ***Digital output state***. Return value will be -1 if access is invalid (i.e. if it addresses external hardware which is not connected).

Syntax

[output index] [group index] ***getdoutst***

Reply

[value]

---

setdoutst

Sets ***Digital output state*** as specified.

Syntax

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

Examples

Example

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

Emergency switch


storable

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

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

Commands

Properties

Type: int

value	enable state
0	disabled
1	enabled

---

getemsw

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

Syntax

getemsw

Reply

[mask]

Examples

Example

	Command	Description
1:	***getemsw***	Returns ***Emergency switch*** 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 ***setemsw***	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
4	-	-	*reserved*
5	-	-	*reserved*
6	timer	0...7	timers 1...8

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

Commands

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 ***getevtdet***	Returns content of CAN joystick 1 ***Event detect register***. 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 mask entry

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

group index	target register
0	*reserved for application specific functions*
1	***Event mask register - controller*** (p. 271)
2	***Event mask register - CAN joystick 1*** (p. 270)
3	***Event mask register - CAN handwheel 1*** (p. 269)
4	*reserved*
5	*reserved*
6	***Event mask register - timer*** (p. 272)

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

Event blockade will affect the processing of incoming events with some corresponding registers. When active, it will

- never inhibit changes of the respective ***Event state*** (p. 287)
- always inhibit changes of the ***Event detect register*** (p. 264) bit in question
- inhibit changes of the ***Action state*** bit in question *only* if corresponding ***Event mode register*** bit setting is 0

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

Commands

Properties

Name	Type
[group index]	int
[value]	int

---

getevtmask

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

Syntax

[group index] ***getevtmask***

Reply

[value]

Examples

Example

Task: Query content of ***Event mask register - timer*** (p. 272) .

Preparation: Timer I/O group index is 6.

	Command	Description
1:	6 ***getevtmask***	Returns content of ***Event mask register - timer***.

---

setevtmask

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

Syntax

[value] [group index] ***setevtmask***

Examples

Example

Task: Block events generated by timers 2 and 5.

Preparation: Bits 1 (timer 2) and 4 (timer 5) are to be cleared. Register value is $-1 - 2^1 - 2^4 = -19$. Timer I/O group index is 6.

	Command	Description
1:	-19 6 ***setevtmask***	Sets content of ***Event mask register - timer*** to -19.

Result: Events generated by timers 2 and 5 will be blocked.

Event mask register - CAN handwheel 1

not storable

Register that contains the mask settings of all pushbuttons at CAN handwheel 1. It can be used to temporarily block occurring trigger events for an ***Action command*** . The events of a pushbutton will be blocked whenever the corresponding register bit is 0.

bit setting	*Action command* will be
0	blocked
1	passed

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

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Properties

Type: int

Event mask register - CAN joystick 1

not storable

Register that contains the mask settings of all pushbuttons at CAN joystick 1. It can be used to temporarily block occurring trigger events for an ***Action command*** . The events of a pushbutton will be blocked whenever the corresponding register bit is 0.

bit setting	*Action command* will be
0	blocked
1	passed

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

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Properties

Type: int

Event mask register - controller


not storable

Register that contains the mask settings of all digital controller inputs. It can be used to temporarily block occurring trigger events for an ***Action command*** . The events at an input will be blocked whenever the corresponding register bit is 0.

bit setting	***Action command*** **will be**
0	blocked
1	passed

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

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Properties

Type: int

Event mask register - timer


not storable

Register that contains the mask settings of all timers. It can be used to temporarily block occurring trigger events for an ***Action command*** . The events of a pushbutton will be blocked whenever the corresponding register bit is 0.

bit setting	***Action command*** will be
0	blocked
1	passed

representative bits	event sources
0...7	timers 1...8

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

See also ***Timer entry*** (p. 337) for information on how to operate a timer.

Properties

Type: int

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
0	*reserved for application specific functions*
1	***Event mode register - controller*** (p. 278)
2	***Event mode register - CAN joystick 1*** (p. 277)
3	***Event mode register - CAN handwheel 1*** (p. 276)
4	*reserved*
5	*reserved*
6	***Event mode register - timer*** (p. 279)

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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.

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

Commands

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. 277) .

Preparation: CAN joystick 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 bidirectional.

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

	Command	Description
1:	12 2 ***setevtmode***	Sets content of ***Event mode register - CAN joystick 1*** 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 *or* every release, depending on polarity setting
1	every push *and* every release

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

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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 *or* every release, depending on polarity setting
1	every push *and* every release

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

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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 *or* every falling edge, depending on polarity setting
1	every rising *and* every falling edge

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

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Properties

Type: int

Event mode register - timer


storable

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

bit setting	***Action command*** **triggered by**
0	every expiration *or* every start, depending on polarity setting
1	every expiration *and* every start

representative bits	event sources
0...7	timers 1...8

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

See also ***Timer entry*** (p. 337) for information on how to operate a timer.

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
0	*reserved for application specific functions*
1	***Event polarity register - controller*** (p. 285)
2	***Event polarity register - CAN joystick 1*** (p. 284)
3	***Event polarity register - CAN handwheel 1*** (p. 283)
4	*reserved*
5	*reserved*
6	***Event polarity register - timer*** (p. 286)

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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.

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

Commands

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. 283) .

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

	Command	Description
1:	3 ***getevtpol***	Returns content of ***Event polarity register - CAN handwheel 1***.

---

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 ***setevtpol***	Sets content of ***Event polarity register - controller*** (p. 285) 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

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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


storable

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

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Properties

Type: int

Event polarity register - timer


storable

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

representative bits	event sources
0...7	timers 1...8

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

See also ***Timer entry*** (p. 337) for information on how to operate a timer.

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
timer	0 => running or halted 1 => expired

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

Commands

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
0	*reserved for application specific functions*
1	***Initial action mask register - controller*** (p. 294)
2	***Initial action mask register - CAN joystick 1*** (p. 293)
3	***Initial action mask register - CAN handwheel 1*** (p. 292)
4	*reserved*
5	*reserved*
6	***Initial action mask register - timer*** (p. 295)

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

Due to its nature, the setting will only show effect after ***Configuration storage*** (p. 362) and ***Reset*** (p. 196) .

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

Commands

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. 294) .

Preparation: Controller I/O group index is 1.

	Command	Description
1:	1 ***getinitactmask***	Returns content of ***Initial action mask register - controller***.

---

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 ***setinitactmask***	Sets content of ***Initial action mask register - CAN handwheel 1*** (p. 292) 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

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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 joystick 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

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

storable

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

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Properties

Type: int

Initial action mask register - timer


storable

Register that contains the initial action mask settings of all digital timers. Each condition is represented by one bit indicating which timers 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...7	timers 1...8

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

See also ***Timer entry*** (p. 337) for information on how to operate a timer.

Properties

Type: int

Internal action command (primary)


storable

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. 311)

s. ***Internal event polarity register* (p. 314)

s. ***Internal action command (secondary) (p. 298) .

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 powered, false if dropped out
3	[condition 3]	string	true if axis is resting on target position; false if performing move or position not in target window

i	Name	Type	Description
4	[condition 4]	string	reserved
5	[condition 5]	string	true if ***Calibration move*** (p. 402) processing
6	[condition 6]	string	true if ***Range measure move*** (p. 425) processing
7	[condition 7]	string	true if reference mark found
8	[condition 8]	string	true if sensor position valid
9	[condition 9]	string	true if machine error pending for inquiry
10	[condition 10]	string	true if trigger sequence running

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. 296) . 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. 311)

s. ***Internal event polarity register* (p. 314)

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 powered, false if dropped out
3	[condition 3]	string	true if axis is resting on target position; false if performing move or position not in target window
4	[condition 4]	string	reserved
5	[condition 5]	string	true if ***Calibration move*** (p. 402) processing

i	Name	Type	Description
6	[condition 6]	string	true if ***Range measure move*** (p. 425) processing
7	[condition 7]	string	true if reference mark found
8	[condition 8]	string	true if sensor position valid
9	[condition 9]	string	true if machine error pending for inquiry
10	[condition 10]	string	true if trigger sequence running

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. 296)
1...2	1	***Internal action command (secondary)*** (p. 298)

* ***[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

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; it has to be entered with the command line nonetheless. Any value (e.g. " ") will be accepted. If it is omitted, though, the Venus interpreter will report an ***Interpreter error*** (code 1002), and execution will be denied. This peculiarity is due to 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 ***getactcmdint***	Returns secondary ***Internal action command*** of "axis 1 ready" condition.

---

setactcmdint

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

If the *value* itself contains another string (like a filename), it can be entered with outer (') and inner (") quotation marks as follows:

'"innerstring"outerstring'

For instance: '"This is a string!" 0 setvarstring'

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 bidirectional function, using ***setevtmodeint*** (p. 312).

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 ***setevtmodeint***	Set modes for "axis 1 powered up" and "axis 1 ready" to bidirectional function.
2:	3 2 ***setevtmodeint***	Set modes for "axis 2 powered up" and "axis 2 ready" to bidirectional function.

Internal action state register


not storable

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	condition
0	axis powered up
1	axis ready
2	axis resting on target
3	*reserved*
4	*processing* ***Calibration move*** (p. 402)
5	*processing* ***Range measure move*** (p. 425)
6	*reference mark found*

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

Commands

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 ***setactstint***	Sets content of internal ***Internal action state register*** 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. 305) .

Preparation: Axis index is 2.

	Command	Description
1:	2 ***getactstint***	Returns content of axis 2 ***Internal action state register***.

Internal event detect register


not storable

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
3	*reserved*
4	*processing* ***Calibration move*** (p. 402)
5	*processing* ***Range measure move*** (p. 425)
6	*reference mark found*

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

Commands

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 ***getevtdetint***	Returns content of axis 1 ***Internal event detect register***. 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


storable

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 *or* every true-to-false transition, depending on polarity setting
1	every true-to-false *and* every false-to-true transition

bit	condition
0	axis powered up
1	axis ready
2	axis resting on target
3	*reserved*
4	*processing* ***Calibration move*** (p. 402)
5	*processing* ***Range measure move*** (p. 425)
6	*reference mark found*

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

Commands

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. 311) at axis 2 .

Preparation: Axis index is 2.

	Command	Description
1:	2 ***getevtmodeint***	Returns content of ***Internal event mode register*** 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 bidirectional.

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 ***setevtmodeint***	Sets content of ***Internal event mode register*** 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


storable

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
3	*reserved*
4	*processing* ***Calibration move*** (p. 402)
5	*processing* ***Range measure move*** (p. 425)
6	*reference mark found*

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

Commands

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 ***setevtpolint***	Sets content of ***Event polarity register - controller*** 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. 314) at axis 1.

Preparation: Axis index is 1.

	Command	Description
1:	1 ***getevtpolint***	Returns content of ***Internal event polarity register*** at axis 1.

Internal event state


not storable

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

bit	condition
0	axis powered up
1	axis ready
2	axis resting on target
3	*reserved*
4	*processing* ***Calibration move*** (p. 402)
5	*processing* ***Range measure move*** (p. 425)
6	*reference mark found*

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

Commands

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 ***getevtstint***	Returns ***Internal event state*** 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


storable

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
3	*reserved*
4	*processing* ***Calibration move*** (p. 402)
5	*processing* ***Range measure move*** (p. 425)
6	*reference mark found*

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

Commands

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 ***setinitactmaskint***	Sets content of ***Internal initial action mask register*** 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 ***getinitactmaskint***	Returns content of ***Internal initial action mask register*** at axis 1.

Manual driver amplitude

read-only

Balanced raw output amplitude generated by specified manual driver channel. Useful with joystick type drivers only, where the raw value is specified as the momentary to full joystick elongation ratio, ranging from -1.0 through 1.0. With handwheel type drivers, the value returned will be 0.0.

Commands

Properties

Name	Type	Description
[driver]	int	**value** / **driver**: 0 joystick 1; 1 handwheel 1
[channel]	int	**driver type** / **range**: 2-channel joystick 0..1; 3-channel joystick 0..2; handwheel 0..1
[value]	double	**driver** / **range**: joystick -1.0 .. +1.0; handwheel 0.0

---

getmanamp

Returns the actual ***Manual driver amplitude***.

Syntax

[channel] [driver] ***getmanamp***

Reply

[value]

Examples

Example

	Command	Description
1:	1 0 ***getmanamp***	Returns ***Manual driver amplitude*** at channel 1 of 1st CAN joystick.

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.....325

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

Initiates ***Manual driver scan***.

Syntax

scanman

Examples

	Command	Description
1:	***scanman***	Initiate ***Manual driver scan***.

Manual driver status

read-only

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.....328

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.

Timer 1

read-only

Status data of timer 1. See ***tmr*** (p. 342) and ***tmrst*** (p. 338) for inquiry.

Properties

Name	Type	Unit	Description
[residual time]	int	s	Residual running time until expiration, not including halt periods
[running state]	int	-	1 if running, 0 if halted

Timer 2

read-only

Status data of timer 2. See ***tmr*** (p. 342) and ***tmrst*** (p. 338) for inquiry.

Properties

Name	Type	Unit	Description
[residual time]	int	s	Residual running time until expiration, not including halt periods
[running state]	int	-	1 if running, 0 if halted

Timer 3

read-only

Status data of timer 3. See ***tmr*** (p. 342) and ***tmrst*** (p. 338) for inquiry.

Properties

Name	Type	Unit	Description
[residual time]	int	s	Residual running time until expiration, not including halt periods
[running state]	int	-	1 if running, 0 if halted

Timer 4

read-only

Status data of timer 4. See ***tmr*** (p. 342) and ***tmrst*** (p. 338) for inquiry.

Properties

Name	Type	Unit	Description
[residual time]	int	s	Residual running time until expiration, not including halt periods
[running state]	int	-	1 if running, 0 if halted

Timer 5

read-only

Status data of timer 5. See ***tmr*** (p. 342) and ***tmrst*** (p. 338) for inquiry.

Properties

Name	Type	Unit	Description
[residual time]	int	s	Residual running time until expiration, not including halt periods
[running state]	int	-	1 if running, 0 if halted

Timer 6

read-only

Status data of timer 6. See ***tmr*** (p. 342) and ***tmrst*** (p. 338) for inquiry.

Properties

Name	Type	Unit	Description
[residual time]	int	s	Residual running time until expiration, not including halt periods
[running state]	int	-	1 if running, 0 if halted

Timer 7

read-only

Status data of timer 7. See ***tmr*** (p. 342) and ***tmrst*** (p. 338) for inquiry.

Properties

Name	Type	Unit	Description
[residual time]	int	s	Residual running time until expiration, not including halt periods
[running state]	int	-	1 if running, 0 if halted

Timer 8

read-only

Status data of timer 8. See ***tmr*** (p. 342) and ***tmrst*** (p. 338) for inquiry.

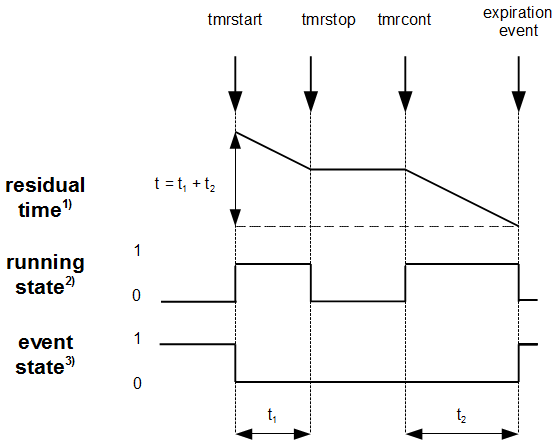
Properties

Name	Type	Unit	Description
[residual time]	int	s	Residual running time until expiration, not including halt periods
[running state]	int	-	1 if running, 0 if halted

Timer entry

Entry to the *timer* facility which allows to have the controller generate and evaluate timed events to trigger *action commands*. Operation is depicted below, where *t* is the total running time entered with ***tmrstart*** (p. 341). The *residual time* and *running state* can be inquired via ***tmr*** (p. 342) and ***tmrst*** (p. 338).

Timer operation scheme



1) see ***tmr*** for inquiry

2) see ***tmrst*** for inquiry

3) see ***Event state*** (p. 287) for description

See introductory chapter "Introduction to digital I/O processing" for general information on the action command feature and on how timers are processed.

Commands

Properties

Name	Type	Unit	Description
[timer index]	int	-	selection index of target timer 1 : ***Timer 1*** (p. 329) 2 : ***Timer 2*** (p. 330) 3 : ***Timer 3*** (p. 331) 4 : ***Timer 4*** (p. 332) 5 : ***Timer 5*** (p. 333) 6 : ***Timer 6*** (p. 334) 7 : ***Timer 7*** (p. 335) 8 : ***Timer 8*** (p. 336)
[value]	double	s*	effective running time *25 ms stepwidth; 50 ms min.

---

tmrst

Returns current *running state* of specified timer.

value	description
0	idle or halted
1	started and running

Syntax

[timer index] ***tmrst***

Reply

[value]

Examples

Example

	Command	Description
1:	3 ***tmrst***	Returns *running state* of ***Timer 3*** (p. 331) .

---

tmrreset

Resets specified timer and sets corresponding ***Event state*** (p. 287) .

Note that the effective reset event my be delayed by up to approx. 25 ms.

Syntax

[value] [timer index] ***tmrreset***

The *value* entry is an ineffective dummy here; it has to be entered with the command line nonetheless. Any value (e.g. 0) will be accepted. If it is omitted, though, the Venus interpreter will report an ***Interpreter error*** (p. 369) (code 1002), and execution will be denied. This peculiarity is due to specific parameter stack handling.

Examples

Example

	Command	Description
1:	4 ***tmrreset***	Resets ***Timer 4*** (p. 332) .

---

tmrstop

Halts specified timer without affecting corresponding ***Event state***.

Note that the effective halt event my be delayed by up to approx. 25 ms.

Syntax

[value] [timer index] ***tmrstop***

The *value* entry is an ineffective dummy here; it has to be entered with the command line nonetheless. Any value (e.g. 0) will be accepted. If it is omitted, though, the Venus interpreter will report an ***Interpreter error*** (code 1002), and execution will be denied. This peculiarity is due to specific parameter stack handling.

Examples

Example

	Command	Description
1:	2 ***tmrstop***	Stops ***Timer 2*** (p. 330) .

---

tmrcont

Releases specified timer from halted state without affecting corresponding ***Event state***.

Note that the effective release event my be delayed by up to approx. 25 ms.

Syntax

[value] [timer index] ***tmrcont***

The *value* entry is an ineffective dummy here; it has to be entered with the command line nonetheless. Any value (e.g. 0) will be accepted. If it is omitted, though, the Venus interpreter will report an ***Interpreter error*** (code 1002), and execution will be denied. This peculiarity is due to specific parameter stack handling.

Examples

Example

	Command	Description
1:	6 ***tmrcont***	Releases ***Timer 6*** (p. 334) from halted state.

---

tmrstart

Starts specified timer with running time specified by *value*, and clears corresponding ***Event state***.

Note that the effective start event my be delayed by up to approx. 25 ms.

Syntax

[value] [timer index] ***tmrstart***

Examples

Example

	Command	Description
1:	10 1 ***tmrstart***	Starts ***Timer 1*** (p. 329) with 10 s running time.

---

tmr

Returns *residual time* of specified timer.

Syntax

[timer index] ***tmr***

Reply

[value]

Examples

Example

	Command	Description
1:	5 ***tmr***	Returns *residual time* of ***Timer 5*** (p. 333) .

Interpreter

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

Command chain 1


storable

Table containing *command* strings executed in accordance with the *command index* which provides command selection. See ***Command chain execution*** (p. 355) on how to use the chain and access the index. See ***Command chain entry*** (p. 353) on how to access the individual command strings.

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Properties

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


storable

Table containing *command* strings executed in accordance with the *command index* which provides command selection. See ***Command chain execution*** (p. 355) on how to use the chain and access the index. See ***Command chain entry*** (p. 353) on how to access the individual command strings.

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Properties

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


storable

Table containing *command* strings executed in accordance with the *command index* which provides command selection. See ***Command chain execution*** (p. 355) on how to use the chain and access the index. See ***Command chain entry*** (p. 353) on how to access the individual command strings.

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Properties

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


storable

Table containing *command* strings executed in accordance with the *command index* which provides command selection. See ***Command chain execution*** (p. 355) on how to use the chain and access the index. See ***Command chain entry*** (p. 353) on how to access the individual command strings.

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Properties

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


storable

Table containing *command* strings executed in accordance with the *command index* which provides command selection. See ***Command chain execution*** (p. 355) on how to use the chain and access the index. See ***Command chain entry*** (p. 353) on how to access the individual command strings.

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Properties

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


storable

Table containing *command* strings executed in accordance with the *command index* which provides command selection. See ***Command chain execution*** (p. 355) on how to use the chain and access the index. See ***Command chain entry*** (p. 353) on how to access the individual command strings.

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Properties

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


storable

Table containing *command* strings executed in accordance with the *command index* which provides command selection. See ***Command chain execution*** (p. 355) on how to use the chain and access the index. See ***Command chain entry*** (p. 353) on how to access the individual command strings.

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Properties

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


storable

Table containing *command* strings executed in accordance with the *command index* which provides command selection. See ***Command chain execution*** (p. 355) on how to use the chain and access the index. See ***Command chain entry*** (p. 353) on how to access the individual command strings.

Note: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

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

Properties

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: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

Note that the *command index* is stored along with the chain commands when a ***Configuration storage*** (p. 362) 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

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: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

Note that the *command index* is stored along with the chain commands when a ***Configuration storage*** (p. 362) is done.

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

Commands

Properties

Type: int
range: 1...8

---

execnextcmd

Increases *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***

---

inccmdindex

Increases *command index* of specified ***Command chain*** by one without command execution.

For parameterized access to the *command index*, see ***Command chain index entry*** (p. 358) .

Syntax

[chain index] ***inccmdindex***

---

execprevcmd

Decreases *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***

---

deccmdindex

Decreases *command index* of specified ***Command chain*** by one without command execution.

For parameterized access to the *command index*, see ***Command chain index entry***.

Syntax

[chain index] ***deccmdindex***

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: With LMT specific firmware rev. 5.1100, this is reserved for factory configuration and to be left untouched by the user.

Note that the *command index* is stored along with the chain commands when a ***Configuration storage*** (p. 362) is done.

See ***Command chain execution*** (p. 355) for *command index* incrementation / decrementation.

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

chain index	target chain
1	***Command chain 1*** (p. 345)
2	***Command chain 2*** (p. 346)
3	***Command chain 3*** (p. 347)
4	***Command chain 4*** (p. 348)
5	***Command chain 5*** (p. 349)
6	***Command chain 6*** (p. 350)
7	***Command chain 7*** (p. 351)
8	***Command chain 8*** (p. 352)

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

Commands

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. 355) or ***execprevcmd*** (p. 357) 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. 532) .

Commands

---

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. 579) 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.

Since firmware rev. 4.2000:

If one or more files get corrupted during the storage procedure, the invalid data will be lost. The controller will, however, restore the most recent valid parameter set during the next powerup and generate an ***Interpreter error***. Subsequently, application of motor power at the

corresponding axes (all axes if Device 0 parameter file is affected) will be blocked for safety reasons, and remain so until the next parameter storage is being executed.

For storage of data from ***Sensor cache*** (p. 532), additional execution of ***savecache*** (p. 532) 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.

Since firmware rev. 4.2000:

If the file gets corrupted during the storage procedure, the invalid data will be lost. The controller will, however, restore the most recent valid parameter set during the next powerup and generate an ***Interpreter error***. Subsequently, application of motor power will be blocked at all axes for safety reasons, and remain so until the next parameter storage and following controller reset are being executed.

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

Syntax

csave

Examples

Example

	Command	Description
1:	***csave***	Initiate ***Configuration storage*** of controller device.

Configuration storage (Axis)

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

Commands

---

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.

Since firmware rev. 4.2000:

If the file gets corrupted during the storage procedure, the invalid data will be lost. The controller will, however, restore the most recent valid parameter set during the next powerup and generate an ***Interpreter error***. Subsequently, application of motor power will be blocked for safety

reasons, and remain so until the next parameter storage and following controller reset are being executed.

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

Syntax

{device} ***nsave***

Examples

Example

	Command	Description
1:	1 ***nsave***	Initiate ***Configuration storage*** of axis 1.

Error decoder

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

Commands

Properties

Name	Type
[errorcode]	int
[errorstring]	string

---

errordecode

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

Syntax

[errorcode] ***errordecode***

Reply

[errorstring]

Examples

Example

	Command	Description
1:	2000 ***errordecode***	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 ***merrordecode***	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
102	undefined symbol
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
3100	most recent valid parameter set restored due to file corruption

Commands

Properties

Type: int

ge

Returns the most recent ***Interpreter error*** code and removes it from the interpreter error stack. Returns 0 if the stack is empty. The command ***errordecode*** (p. 367) returns the error description of the code in a string.

Syntax

ge

Reply

[errorregister]

Examples

Example

	Command	Description
1:	***ge***	Returns ***Interpreter error***.

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
102	undefined symbol
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
3100	most recent valid parameter set restored due to file corruption

Commands

Properties

Type: int

gne

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

Syntax

{device} ***gne***

Reply

[errorregister]

Examples

Example

	Command	Description
1:	1 ***gne***	Return ***Interpreter error*** 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

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 ***nclear***	Clears axis ***Parameter stack***.

---

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 ***ngsp***	Returns number of parameters on ***Parameter stack***.

User doubles


storable

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

Commands

Properties

i	Name	Type	Description
0	[vardbl_0]	double	Reserved for factory configuration with with LMT specific firmware rev. 5.1100. In this case, do not alter. User specific with any other firmware rev.
1	[vardbl_1]	double	Reserved for factory configuration with with LMT specific firmware rev. 5.1100. In this case, do not alter. User specific with any other firmware rev.
2	[vardbl_2]	double	Reserved for factory configuration with with LMT specific firmware rev. 5.1100. In this case, do not alter. User specific with any other firmware rev.
3	[vardbl_3]	double	Reserved for factory configuration with with LMT specific firmware rev. 5.1100. In this case, do not alter. User specific with any other firmware rev.
4	[vardbl_4]	double	User specific.
5	[vardbl_5]	double	User specific.
6	[vardbl_6]	double	User specific.
7	[vardbl_7]	double	User specific.
8	[vardbl_8]	double	User specific.
9	[vardbl_9]	double	User specific.

getvardbl

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

Syntax

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

Reply

[*Value*]

Examples

Example

	Command	Description
1:	5 ***getvardbl***	Returns entry at index 5 of ***User doubles***.

setvardbl

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

Syntax

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

Examples

Example

	Command	Description
1:	123.456 6 ***setvardbl***	Writes 123.456 to entry 6 of ***User doubles***.

User integers


storable

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

Commands

Properties

i	Name	Type
0	[varint_0]	int
1	[varint_1]	int
2	[varint_2]	int
3	[varint_3]	int
4	[varint_4]	int
5	[varint_5]	int
6	[varint_6]	int
7	[varint_7]	int
8	[varint_8]	int
9	[varint_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

Properties

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

getvarstring

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

Syntax

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

Reply

[*Value*]

Examples

Example

	Command	Description
1:	3 ***getvarstring***	Returns entry at index 3 of ***User strings***.

---

setvarstring

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

Syntax

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

Examples

Example

	Command	Description
1:	"mystring" 4 ***setvarstring***	Writes "mystring" to entry 4 of ***User strings***.

Manual operation

Before operating the controller manually, please take at 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

Properties

Name	Type	Description
[control index]	int	selection index of target manual device 1 : ***Manual device parameters 0*** (p. 388) 2 : ***Manual device parameters 1*** (p. 390) 3 : ***Manual device parameters 2*** (p. 392) 4 : ***Manual device parameters 3*** (p. 394)
[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 ***setmanpara***	Sets the *velocity* at the 1st manual device of axis 1 to 100 mm/s.
2:	2 6 1 2 ***setmanpara***	Sets the *input function* at the 1st manual device of axis 2 to cubic progression.

getmanpara

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

Syntax

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

The *parameter value* entry is an ineffective dummy here; it has to be entered with the command line nonetheless. Any value (e.g. 0) will be accepted. If it is omitted, though, the Venus interpreter will report an ***Interpreter error*** (code

1002), and execution will be denied. This peculiarity is due to specific parameter stack handling.

Reply

[parameter value]

Examples

Example

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

Manual device parameters 0

storable

Properties of manual device 0.

The value of the *jerk* and *acceleration* entries must comply with the following restriction:

jerk >= 60 s^{-1} * *acceleration*

A programmed move unconditionally overrules manual motion.

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

Properties

i	Name	Type	Unit	Description
1	[velocity]	double	mm/s	manual move velocity limit Default: 40.0 Range: 0.00001 ... 10000.0
2	[acceleration]	double	mm/s²	manual move acceleration limit Default: 100.0 Range: 0.001 ... 500000.0
3	[alt. velocity]	double	mm/s	manual move velocity valid when button pressed - *void since* ***Controller version*** (p. 163) *2.200*
4	[resolution]	int	1/rev	*handwheel* - encoder steps per revolution invariably *323584* with ITK 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, 2: cubic, etc.)

i	Name	Type	Unit	Description
7	[mode]	int	-	0: disabled 1: enabled until next programmed move; disabled afterwards 2: enabled until next programmed move and afterwards *Note: continuity purpose only - set value should normally be 2; disabling should be done utilizing* ***Manual motion control*** (p. 396)
8	[threshold]	double	-	*joystick* - 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 1: handwheel 1
11	[input channel]	int	-	joystick - 0: horizontal, 1: vertical, 2: rotational handwheel - 0: front rotor, 1: rear rotor
18	[jerk]	double	mm/s³	manual move jerk limit; effective with jerk-optimized operation mode only - *valid since* ***Controller version*** *4.0000*

Manual device parameters 1

storable

Properties of manual device 1.

The value of the *jerk* and *acceleration* entries must comply with the following restriction:

jerk >= 60 s^{-1} * *acceleration*

A programmed move unconditionally overrules manual motion.

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

Properties

i	Name	Type	Unit	Description
1	[velocity]	double	mm/s	manual move velocity limit Default: 40.0 Range: 0.00001 ... 10000.0
2	[acceleration]	double	mm/s²	manual move acceleration limit Default: 100.0 Range: 0.001 ... 500000.0
3	[alt. velocity]	double	mm/s	manual move velocity valid when button pressed - *void since* ***Controller version*** (p. 163) *2.200*
4	[resolution]	int	1/rev	*handwheel* - encoder steps per revolution invariably *323584* with ITK 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, 2: cubic, etc.)

i	Name	Type	Unit	Description
7	[mode]	int	-	0: disabled 1: enabled until next programmed move; disabled afterwards 2: enabled until next programmed move and afterwards *Note: continuity purpose only - set value should normally be 2; disabling should be done utilizing* ***Manual motion control*** (p. 396)
8	[threshold]	double	-	*joystick* - 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 1: handwheel 1
11	[input channel]	int	-	joystick - 0: horizontal, 1: vertical, 2: rotational handwheel - 0: front rotor, 1: rear rotor
18	[jerk]	double	mm/s³	manual move jerk limit; effective with jerk-optimized operation mode only - *valid since* ***Controller version*** *4.0000*

Manual device parameters 2

storable

Properties of manual device 2.

The value of the *jerk* and *acceleration* entries must comply with the following restriction:

jerk >= 60 s^{-1} * *acceleration*

A programmed move unconditionally overrules manual motion.

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

Properties

i	Name	Type	Unit	Description
1	[velocity]	double	mm/s	manual move velocity limit Default: 40.0 Range: 0.00001 ... 10000.0
2	[acceleration]	double	mm/s²	manual move acceleration limit Default: 100.0 Range: 0.001 ... 500000.0
3	[alt. velocity]	double	mm/s	manual move velocity valid when button pressed - *void since* ***Controller version*** (p. 163) *2.200*
4	[resolution]	int	1/rev	*handwheel* - encoder steps per revolution invariably *323584* with ITK 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, 2: cubic, etc.)

i	Name	Type	Unit	Description
7	[mode]	int	-	0: disabled 1: enabled until next programmed move; disabled afterwards 2: enabled until next programmed move and afterwards *Note: continuity purpose only - set value should normally be 2; disabling should be done utilizing* ***Manual motion control*** (p. 396)
8	[threshold]	double	-	*joystick* - 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 1: handwheel 1
11	[input channel]	int	-	joystick - 0: horizontal, 1: vertical, 2: rotational handwheel - 0: front rotor, 1: rear rotor
18	[jerk]	double	mm/s^3	manual move jerk limit; effective with jerk-optimized operation mode only - *valid since* ***Controller version*** *4.0000*

Manual device parameters 3

storable

Properties of manual device 3.

The value of the *jerk* and *acceleration* entries must comply with the following restriction:

jerk >= 60 s^{-1} * *acceleration*

A programmed move unconditionally overrules manual motion.

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

Properties

i	Name	Type	Unit	Description
1	[velocity]	double	mm/s	manual move velocity limit Default: 40.0 Range: 0.00001 ... 10000.0
2	[acceleration]	double	mm/s²	manual move acceleration limit Default: 100.0 Range: 0.001 ... 500000.0
3	[alt. velocity]	double	mm/s	manual move velocity valid when button pressed - *void since* ***Controller version*** (p. 163) *2.200*
4	[resolution]	int	1/rev	*handwheel* - encoder steps per revolution invariably *323584* with ITK 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, 2: cubic, etc.)

i	Name	Type	Unit	Description
7	[mode]	int	-	0: disabled 1: enabled until next programmed move; disabled afterwards 2: enabled until next programmed move and afterwards *Note: continuity purpose only - set value should normally be 2; disabling should be done utilizing* ***Manual motion control*** (p. 396)
8	[threshold]	double	-	*joystick* - 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 1: handwheel 1
11	[input channel]	int	-	joystick - 0: horizontal, 1: vertical, 2: rotational handwheel - 0: front rotor, 1: rear rotor
18	[jerk]	double	mm/s³	manual move jerk limit; effective with jerk-optimized operation mode only - *valid since* ***Controller version*** *4.0000*

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 the manual drivers will always be effective, regardless of their respective enable settings. Enable state has no impact on LED function either.

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

Commands

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 ***setmanctrl***	Enable ***Manual device 0*** and ***Manual device 1*** 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 ***getmanctrl***	Returns ***Manual motion control*** at axis 1.

Mechanic

Pitch

storable

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

To operate the device in closed loop mode, the parameter setting has to provide that delivered move distances equal actually covered move distances. See also ***Motor pole pairs*** (p. 498)

Commands

Properties

Type: double
Unit: mm

---

getpitch

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

Syntax

{device} ***getpitch***

Reply

[pitch]

Examples

Example

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

Causes device 1 motor axis to turn twice.

Mechanic setup

Calibration move

Composite move for the purpose of finding the lower motion range limit and defining the position origin. The function is in detail also dependent on the ***Switch configuration*** (p. 604) chosen. 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. 404) 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. 588))
- *if bit 2 of* ***Switch configuration*** *is set to 0*: setting of ***Hardware limits*** (p. 408)
  - *lowerlimit* will be set to 0
  - *upperlimit* will be set to *upperlimit* of ***Initial limits*** (p. 410)
- scale correction activation (if configured accordingly)
- motor optimization (if configured accordingly)

See ***Motion function*** (p. 415) on move configuration. See also introductory chapter "Motor commutation handling" on motor optimization.

Commands

ncal (ncalibrate)

Initiates a ***Calibration move***.

Syntax

{device} ***ncal***

Examples

Example

	Command	Description
1:	1 ***ncal***	Initiates ***Calibration move*** at axis 1.

Calibration switch distance


storable

Distance from "Cal" end switch at which ***Calibration move*** (p. 402) terminates.

Commands

Properties

Type: double
Unit: mm

---

setncalswdist

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

Syntax

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

Examples

Example

	Command	Description
1:	1 1 ***setncalswdist***	Set axis 1 ***Calibration switch distance*** to 1 mm.

---

getncalswdist

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

Syntax

{device} ***getncalswdist***

Reply

[dist from switch]

Examples

Example

	Command	Description
1:	2 ***getncalswdist***	Returns ***Calibration switch distance*** at axis 2.

Calibration velocity


storable

Velocity at which a ***Calibration move*** (p. 402) will run. Set separately for motion towards switch and backward motion, respectively.

Commands

Properties

i	Name	Type	Unit	Description
1	[vel into switch]	double	mm/s	Default: 6.0 Range: 0.00001 ... 10000.0
2	[vel out of switch]	double	mm/s	Default: 1.0 Range: 0.00001 ... 10000.0

---

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 ***getncalvel***	Returns ***Calibration velocity*** at axis 1.

---

setncalvel

Sets the ***Calibration velocity***.

Syntax

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

Examples

Example

	Command	Description
1:	3 1 2 ***setncalvel***	Set axis 2 ***Calibration velocity***, *vel into switch* entry, to 3 mm/s.

Hardware limits


not storable

Limits of available motion range, applying to all moves except ***Calibration move*** (p. 402) and ***Range measure move*** (p. 425). 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. 410).

Commands

Properties

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

---

getnlimit

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

Syntax

{device} ***getnlimit***

Reply

[lowerlimit] [upperlimit]

Examples

Example

	Command	Description
1:	2 ***getnlimit***	Return ***Hardware limits*** at axis 2.

---

setnlimit

Sets the ***Hardware limits***.

Never alter ***Hardware limits*** while axis is moving; doing so may potentially result in indeterminate action.

Syntax

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

Examples

Example

	Command	Description
1:	-50 200 1 ***setnlimit***	Set axis 1 ***Hardware limits*** to -50 (*lower limit*) and 200 (*upper limit*).

Initial limits


storable

Defines storable values for the initialization of the ***Hardware limits*** (p. 408) . 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. 402) .

Regard that the initial limits will not be matched upon change of the ***Position origin*** (p. 419) .

Setting both initial limits to 0 will reset them to their defaults (+/- 200 m).

Commands

Properties

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

---

getinilimit

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

Syntax

{device} ***getinilimit***

Reply

[lowerlimit] [upperlimit]

Examples

Example

	Command	Description
1:	1 ***getinilimit***	Returns ***Initial limits*** at axis 1.

---

setinilimit

Sets the ***Initial limits***.

Syntax

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

Examples

Example

	Command	Description
1:	-70 70 1 ***setinilimit***	Set axis 1 ***Initial limits*** to -70 (*lower limit*) and 70 (*upper limit*).

Motion direction


storable

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

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

Regard that the setting has to be consistent with the count direction of the respective position sensor. If uncertain about that, put the device into open loop mode before saving the parameter setting; then perform ***Reset*** (p. 196) and verify consistency after controller powerup.

Note that with linear and AC motors, direction must not be inverted without matching and storage of the motor commutation offset.

Commands

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. 402), ***Range measure move*** (p. 425) and ***Reference move*** (p. 453) by a bit-coded value. If the respective bit is low, each corresponding move request will be discarded. Also configures motor optimization and reference mark distance decoding.

<table>
<tr><th>Bit number</th><th>description</th></tr>
<tr><td>0</td><td>Calibration move activation*</td></tr>
<tr><td>1</td><td>Range measure move activation*</td></tr>
<tr><td>2</td><td>Reference move activation*</td></tr>
<tr><td>3</td><td>Arming of motor optimization*</td></tr>
<tr><td>4</td><td>Motor optimization selection
<table>
<tr><th>bit value</th><th>home location</th><th>initiating move</th></tr>
<tr><td>0</td><td>cal limit switch</td><td>Calibration move</td></tr>
<tr><td>1</td><td>reference mark</td><td>Reference move</td></tr>
</table></td></tr>
<tr><td>5</td><td>Arming of reference mark distance decoding*
valid since firmware rev. 4.4000</td></tr>
<tr><td>6</td><td>Arming of Reference offset detection*
valid since firmware rev. 4.4000</td></tr>
</table>

* 0 = off; 1 = on

With LMT specific firmware rev. 5.1100, the register content will invariably be zero, implicitly disabling all calibration, position referencing and motor optimization options. These are not needed with LMT microscope stages.

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. 487) for information on what step to take next.

Note that reference mark distance decoding will only yield results with position scales featuring multiple (at least 2) such marks; otherwise, ***Reference move*** will arrive at the respective motion limit without having detected a reference mark.

Commands

Properties

Type: int

---

getmotionfunc

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

Syntax

{device} ***getmotionfunc***

Reply

[mask]

Examples

Example

	Command	Description
1:	1 ***getmotionfunc***	Returns ***Motion function*** at axis 1.

---

setmotionfunc

Sets the ***Motion function***.

Syntax

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

Examples

Example

Task: Disable ***Range measure move*** (p. 425), but leave ***Calibration move*** and ***Reference move*** 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.

Note that taught-in positions (s. ***Position latching*** (p. 181) , ***Position storage*** (p. 188) , ***Position restorage*** (p. 184)) will physically shift according to the ***Position origin*** (p. 419) applied.

Commands

Properties

Type: double

---

getnpos

Returns current ***Position origin***.

Syntax

{device} ***getnpos***

Reply

[position offset]

Examples

Example 1

A sequence of commands (targeting device 1), showing the ***setnpos*** (p. 420)***/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. 422)) is set to default (0).

	Command	Description
1:	1 ***getnpos***	Returns origin offset which is 0 by default.
2:	20 1 ***nm*** (p. 435)	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 offset] {device} ***setnpos***

Examples

Example 1

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

Example 2

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

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

Example 3

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

Position origin configuration


storable

Configures the handling of the ***Position origin*** (p. 419) . The only parameter to be configured at the time is the effective direction, i.e. the sign of the origin.

Commands

Properties

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

---

getorgconfig

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

Syntax

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

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 ***getorgconfig***	Return *sign* entry of ***Position origin configuration*** 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*** (p. 589)	Returns -10.000000.
6:	2 ***np***	Returns 10.000000.
7:	1 ***getnpos*** (p. 419)	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. 408)) to actual position

Upon execution of *Ctrl*+*C* or ***nabort*** (p. 448), 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. 224) , ***Stop deceleration*** (p. 218) and ***Acceleration*** (p. 212)).

See ***Motion function*** (p. 415) on move configuration.

Commands

---

nrm (nrangemeasure)

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

Syntax

{device} ***nrm***

Examples

Example

	Command	Description
1:	2 ***nrm***	Initiates ***Range measure move*** at axis 2.

Range measure velocity


storable

Velocity at which a ***Range measure move*** (p. 425) will run. Set separately for motion towards switch and motion out of switch, respectively.

Commands

Properties

i	Name	Type	Unit	Description
1	[vel into switch]	double	mm/s	Default: 6.0 Range: 0.00001 ... 10000.0
2	[vel out of switch]	double	mm/s	Default: 1.0 Range: 0.00001 ... 10000.0

---

setnrmvel

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

Syntax

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

Examples

Example

	Command	Description
1:	5 1 ***setnrmvel***	Set axis 1 ***Range measure velocity*** 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

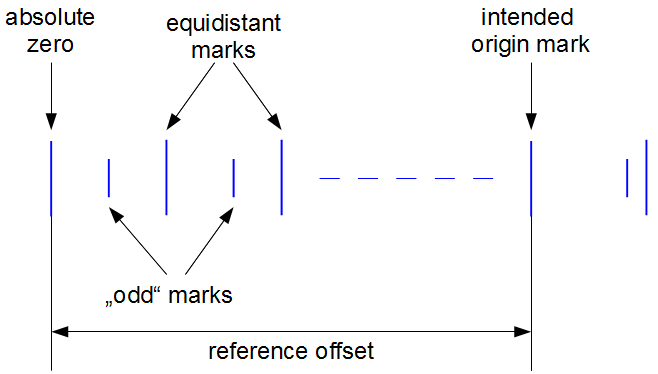
Example

	Command	Description
1:	2 ***getnrmvel***	Returns ***Range measure velocity*** at axis 2.

Reference offset

storable

Offset of the reference mark used as the origin with the position scale assigned, related to the absolute zero of the distance coding. The offset is established utilizing ***Reference move*** (p. 453) .



Valid with scales featuring multiple distance coded reference marks only.

Available since firmware rev. 4.4000.

Commands

Properties

Type: double

getrefoffset

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

Syntax

{device} ***getrefoffset***

Reply

[position offset]

Examples

Example

	Command	Description
1:	1 ***getrefoffset***	Returns the ***Reference offset*** at axis 1.

Reference velocity


storable

Defines motion velocity for reference position mark detection. The setting takes effect upon execution of ***refmove*** (p. 446) and ***nrefmove*** (p. 456).

Commands

Properties

i	Name	Type	Unit	Description
1	[forward velocity]	double	mm/s	Default: 10.0 Range: 0.00001 ... 10000.0
2	[backward velocity]	double	mm/s	Default: 10.0 Range: 0.00001 ... 10000.0

---

getnrefvel

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

Syntax

{device} ***getnrefvel***

Reply

[forward velocity] [backward velocity]

Examples

Example

	Command	Description
1:	1 ***getnrefvel***	Returns ***Reference velocity*** at axis 1.

---

setnrefvel

Sets the ***Reference velocity***.

Syntax

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

Examples

Example

	Command	Description
1:	20 1 2 ***setnrefvel***	Set axis 2 *forward* ***Reference velocity*** to 20 mm/s.

Motion

Before use of clock and direction interface, please take at 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. 448), the slide/rotor will halt immediately.

Commands

Properties

Type: double

---

nm (nmove)

Initiates an ***Absolute move***.

Syntax

[target position] {device} ***nm***

Examples

Example

	Command	Description
1:	10 1 ***nm***	Initiate a move at axis 1 to coordinate 10 mm.

Acceleration function

storable

Selection of move trajectory shape.

Valid since ***Controller version*** (p. 163) 4.0000.

value	description
0	standard shape
1	reserved
2	jerk-optimized shape

Commands

Properties

Type: int

---

getnaccelfunc

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

Syntax

{device} ***getnaccelfunc***

Reply

[mode]

---

setnaccelfunc

Sets the ***Acceleration function***.

Blocked by default - available with a valid release code file only. Available by default with LMT specific firmware rev. 5.1100. See ***Released options*** (p. 194) on how to inquire available options.

Actual mode activation is delayed until next standstill of axis.

Entry of any invalid value (other than 0 or 2) will activate the standard mode (0).

Syntax

[mode] {device} ***setnaccelfunc***

Axis alignment

For rotational axes (***Position cycle*** (p. 591) set to a valid value) only. Move that aligns motor axis to the specified *target position* in the given *direction* within the current ***Position cycle*** based revolution.

Void with LMT specific firmware rev. 5.1100.

Available since firmware rev. 5.1000.

Commands

Properties

Name	Type	Unit	Description
[direction]	int	-	<table><tr><th>value</th><th>description</th></tr><tr><td>-1</td><td>negative</td></tr><tr><td>0</td><td>auto (shortest way)</td></tr><tr><td>+1</td><td>positive</td></tr></table>
[target position]	double	mm	***Position cycle*** > value >= 0

---

align

Performs ***Axis alignment***.

Syntax

[target position] [direction] {device} ***align***

Examples

Example

	Command	Description
1:	3.2 -1 1 ***align***	Aligns axis 1 to position 3.2 mm in negative direction.

Clock and direction function

storable

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. As well as with programmed and manual moves, the target position will be transformed into a trajectory according to the shape chosen (s. ***Acceleration function*** (p. 437) . The trajectory is subject to all parameterized velocity, acceleration, and, where applicable, jerk limitations. For changing the position to clock ratio, see ***Clock and direction width*** (p. 444)).

Note that the clock/direction facility

- will not respond to limit switches; travel range limits must be established prior to activation
- will be disabled automatically whenever a travel range limit is touched during motion; reenabling can be done by application of ***Motor restart*** (p. 506)

Commands

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***.

Note that a new setting will only take effect with next execution of ***Motor restart***.

Syntax

[enabled] {device} ***setcdfunc***

Examples

Example

	Command	Description
1:	2 1 ***setcdfunc***	Enable the ***Clock and direction function*** 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 ***getcdfunc***	Returns ***Clock and direction function*** enable state at axis 1.

Clock and direction width


storable

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. 441) is enabled.

Commands

Properties

Type: int
Unit: nm

---

setcdwidth

Sets the ***Clock and direction width***.

Never alter ***Clock and direction width*** while ***Clock and direction function*** (p. 441) is active.

Syntax

[value] {device} ***setcdwidth***

Examples

Example

	Command	Description
1:	0 2 ***setcdfunc*** (p. 442)	Request ***Clock and direction function*** to be deactivated at axis 2.
2:	2 ***init***	Validate deactivation.
3:	1000 2 ***setcdwidth***	Set ***Clock and direction width*** at axis 2 to 1 µm / pulse.

---

getcdwidth

Returns the current setting of the ***Clock and direction width***.

Syntax

{device} ***getcdwidth***

Reply

[value]

Examples

Example

	Command	Description
1:	2 ***getcdwidth***	Return ***Clock and direction width*** at axis 2.

Controller reference move

Extended version of ***Reference move*** (p. 453); 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. 431) and ***Reference configuration*** (p. 597).

Upon execution of *Ctrl+C* or ***nabort*** (p. 448), the slide(s)/ rotor(s) will halt immediately.

Commands

Properties

Type: double
Unit: mm

---

refmove

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

Syntax

[distance] ***refmove***

Examples

Example

	Command	Description
1:	15 1 ***refmove***	Initiate ***Controller reference move*** 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. 218) or ***Acceleration*** (p. 212) - whatever setting is higher.

Commands

---

nabort

Executes ***Move abortion***.

Syntax

{device} ***nabort***

Examples

Example

	Command	Description
1:	1 ***nabort***	Initiate ***Move abortion*** at axis 1.

Parameterised absolute move


not storable

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

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

Commands

Properties

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

---

pm

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

Syntax

[targetposition] [velocity] [acceleration] [deceleration] {device}
pm

Parameterised relative move


not storable

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

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

Commands

Properties

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

---

pr

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

Syntax

[distance] [velocity] [acceleration] [deceleration] {device} ***pr***

Random move

A sequence of moves targeting virtually random positions at virtually random velocities. The distance and velocity of each invidual move are calculated in a way it will not take more than about 8 seconds. Move velocity will be ***Velocity*** (p. 224) at most, and move acceleration/deceleration will be ***Acceleration*** (p. 212) .

For move parameter calculation, valid ***Motion limits*** and a valid ***Position origin*** (p. 419) are needed. Carry out all necessary action to establish both before execution.

Upon execution of *Ctrl+C* or ***nabort*** (p. 448), the slide/rotor will halt immediately.

Commands

---

nrandmove

Executes ***Random move***.

Syntax

{device} ***nrandmove***

Examples

Example

	Command	Description
1:	2 ***nrandmove***	Initiates ***Random move*** at axis 2.

Reference move

Composite relative move covering the given distance. While moving, reference position mark detection is performed according to ***Reference configuration*** (p. 597). The procedure depends on the mode selection (***Motion function*** (p. 415), bits 5 and 6).

1. Partial operations with multiple reference detection and distance decoding, in the order of execution:

- motion until a first reference mark is found (center detection)
- if a first reference mark was found, motion continuance until a second reference mark is found (edge detection)
- if second reference mark was found,
  - braking and backwards motion to found position
  - establishment of absolute position, related to the origin reference mark
  - establishment of ***Reference offset*** (p. 429) if configured accordingly
- update of ***Reference status*** (p. 599) according to move result

2. Partial operations with single reference detection, in the order of execution:

- motion until a reference mark is found (center detection)
- if reference mark was found,
  - braking and backwards motion to found position
  - scale correction activation (if configured accordingly)
  - motor optimization (if configured accordingly)
- setting of ***Reference status*** according to move result

If - at any time during the process, and regardless of the selected mode - the complete move distance is covered or a limit switch is touched, the axis will halt, leaving the reference mark detection unfinished. See ***Reference status*** for information upon the move result.

It is strongly recommended to still protect the travel range limits with appropriate limit switches or sensors when utilizing the distance decoding. Any damage resulting from the use of this feature without limit protection is in user responsibility.

Note that, with the multiple reference detection enabled, ***Reference move*** (p. 453) will not enable position correction or perform motor optimization.

Decreasing ***Reference velocity*** (p. 431) setting helps increase the reliability of reference detection; it should be set well below

v_{Max} = 2000 * ***Scale period*** (p. 526) / [s]

With the device moving too fast, reference detection may fail though the mark is crossed while moving.

Upon execution of *Ctrl+C* or ***nabort*** (p. 448), the slide/rotor will halt immediately.

See ***Motion function*** on move configuration. With linear or AC motors, see also introductory chapter "Motor commutation handling" on motor optimization.

Note that the distance decoding feature is only available since firmware rev. 4.4000.

Commands

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. 224) and ***Acceleration*** (p. 212) .

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. 448), the slide/rotor will halt immediately.

Commands

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 ***nr***	Initiates ***Relative move*** at axis 1 which targets position coordinate -20 mm from start position.

Stop move

Immediate stop of currently running move. Braking slope is set according to ***Acceleration*** (p. 212) .

Valid since ***Controller version*** (p. 163) 4.1300.

Commands

---

nstop

Induces ***Stop move***.

Syntax

{device} ***nstop***

Examples

Example

	Command	Description
1:	1 ***nstop***	Stops axis 1 from moving.

Vector move

Vectorial move, the trajectory course of which is specified by

- the cartesic x and y position components of the target location or move distance
- the ***Vector velocity*** (p. 222)
- the ***Vector acceleration*** (p. 220)

The controller will move the object along a straight line approximately.

Upon execution of *Ctrl+C* or ***nabort*** (p. 448), the slide(s)/ rotor(s) will halt immediately.

The coordinate is subjected to ***Coordinate transformation*** when active.

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*** (p. 222)	Set the vectorial velocity at 10 mm/s.
2:	100 ***sa*** (p. 221)	Set the vectorial acceleration at 100 mm/s².
3:	4.330127 2.5 ***r*** (p. 462)	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

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. 463) and ***v2r*** (p. 464)

---

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 ***r***	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. 464), 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. 462), 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).

---

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).

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. 494)).

Commands

gi (getcurrent) 467

Properties

Type: double
Unit: A

---

gi (getcurrent)

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

Syntax

{device} ***gi***

Reply

[absolutcurrent]

Examples

Example

	Command	Description
1:	2 ***gi***	Returns ***Absolute motor current*** at axis 2.

Auto commutation


storable

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

1500 µs 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. 484) set to 0, 4, or 5).

Commands

Properties

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

<table>
<tr><th>i</th><th>Name</th><th>Type</th><th>Unit</th><th>Description</th></tr>
<tr><td>3</td><td>[mode]</td><td>int</td><td>-</td><td>Defines source of commutation offset for initial commutation (during powerup) and later recommutation upon Motor restart (p. 506) .
Valid since Controller version (p. 163) 4.1100.
Commutation offset source other than Auto commutation (p. 469) will result in skipping of the auto commutation procedure.
<table>
<tr><th>Value</th><th>initial commutation</th><th>recommutation</th></tr>
<tr><td>0 *</td><td>Motor current shift (p. 480)</td><td>Motor current shift</td></tr>
<tr><td>1</td><td>Auto commutation</td><td>Auto commutation</td></tr>
<tr><td>2</td><td>Auto commutation</td><td>Motor current shift</td></tr>
<tr><td>3</td><td>Auto commutation</td><td>Motor parameters (p. 489) , optimized angle offset entry **</td></tr>
</table>
* applicable to systems with stepper motor or absolute position encoder only
** if valid only, else Motor current shift</td></tr>
</table>

---

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


storable

Power state of motor after powerup sequence; has no impact on the execution of ***Auto commutation*** (p. 469) during powerup. If motor is configured to be dead after powerup, power can be recovered by execution of ***Motor restart*** (p. 506) .

Commands

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 ***setinimotorstate***	Set ***Initial motor power state*** at axis 1 to 0.
2:	***save***	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 ***getinimotorstate***	Return ***Initial motor power state*** at axis 2.

Motor brake

storable

Mechanical motor brake. It is possible to dedicate one of the digital controller outputs to a special function which is designed to control a mechanical motor brake, especially for the use with vertical axes to be held in position while the motor is without power. During powerup, the brake output will be kept at low level until the motor is powered, then go high to open the brake. In case of emergency power off, the output will switch to low level again and kept at this state until the motor is repowered by ***Motor restart*** (p. 506) .

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 other special output functions or the ***Digital output state*** (p. 258) 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.

Note that the setting will only be effective after ***Configuration storage*** (p. 362) and ***Reset*** (p. 196) .

Commands

Properties

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

---

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.

---

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.

Motor current limit


storable

Defines the motor current limit used for safety purposes. As soon as the actual ***Absolute motor current*** (p. 467) 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. 506) .

Ineffective whenever set to 0 (i.e. overcurrent protection will be disabled).

Commands

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 ***getmaxcurrent***	Returns ***Motor current limit*** at axis 2.

---

setmaxcurrent

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

Syntax

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

Examples

Example

	Command	Description
1:	10 1 ***setmaxcurrent***	Set ***Motor current limit*** 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. 469) . Automatically related to *home position* after *homing* procedure.

Not to be touched with stepper motors.

Can be corrected manually to optimize motor commutation with axes that combine a linear or AC motor with an absolute scale. See also introductory chapter "Motor commutation handling".

Commands

Properties

Type: double

---

setMCShift

Sets the ***Motor current shift*** value.

Regard that the value will be overwritten by every application of ***Auto commutation*** (p. 469) .

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. 489) on how to parameterize the average dissipation check.

Commands

Properties

Type: double

---

getdissipation

Returns the actual ***Motor dissipation***.

Syntax

{device} ***getdissipation***

Reply

[dissipation]

Examples

Example

	Command	Description
1:	1 ***getdissipation***	Returns ***Motor dissipation*** 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

*not yet supported

Will invariably select linear motor (2) with LMT specific firmware rev. 5.1100.

A mismatched setting will prevent proper motor operation.

Commands

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 ***setmotor***	Sets the ***Motor form*** at device 1 to *linear motor*.
2:	***save***	Stores the complete parameters set.

After next ***Reset***, device 1 will be configured to drive a linear motor.

Motor optimization stage

read-only

Bit coded information on the motor optimization status. This has a rather unique status as it cannot directly be modified by the user, but is stored partially upon ***Configuration storage*** (p. 362) .

bit index	description	
0	**bit value**	**description**
	0	homing pending
	1	home position found
1	**bit value**	**description**
	0	optimized offset specification pending
	1	optimized offset specified
2	**bit value**	**description**
	0	optimization inactive
	1	optimization active

Bits 0 and 2 will always be cleared after powerup, whereas bit 1 is subject to storage for recognition after powerup.

Note that bits 1 and 2 will be cleared upon

- change of ***Motion direction*** (p. 412)
- change of home selection (s. ***Motion function*** (p. 415))

Commands

Properties

Type: int

getmotoptst

Returns the actual ***Motor optimization stage***.

Syntax

{device} ***getmotoptst***

Reply

[state]

Examples

Example

	Command	Description
1:	2 ***getmotoptst***	Returns ***Motor optimization stage*** 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. 469) runs with variation of the slide/rotor location and environmental conditions. It is designed to be entered manually and relates to the chosen home location (either calibration limit switch or reference mark location), independent of the initial motor/slide location.

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 [A^2].

Note that the *optimized angle offset* must be handled with care because entering a new value will immediately recommute the motor. A mismatched value will inhibit proper motor operation and may cause damage.

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. 415) for information on how to select the motor optimization home position and arm/disarm the motor optimization facility. Inquire ***Motor optimization stage*** (p. 487) for information if motor optimization facility is ready for *optimized angle offset* entry.

See ***Motor dissipation*** (p. 482) on how to inquire the current average dissipation.

Commands

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 *resistance* to 4 Ohms.
2:	10 5 1 ***setmotorpara***	Sets *max. average dissipation* to 10 Watts.
3:	3 6 1 ***setmotorpara***	Sets *dissipation check period* 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 ***getmotorpara***	Returns *motor winding inductance* setting at axis 2.

Motor phase current

read-only

Measured current at 2 motor phases. See also ***Absolute motor current*** (p. 467) .

Commands

Properties

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

---

gc

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

Syntax

{device} ***gc***

Reply

[current phase1] [current phase2]

Examples

Example

	Command	Description
1:	1 ***gc***	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

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 ***setphases***	Sets the ***Motor phase number*** at device 1 to 2.
2:	***save***	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 ***getphases***	Returns ***Motor phase number*** 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. 484) 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

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 up to Rev. 4.14. Void with linear, AC, and DC motors since Rev. 4.20. See ***Motor form*** (p. 484) for motor type inquiry.

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. 494) and ***gi*** (p. 467) for current observation.

Commands

Properties

Type: double
Unit: Vs/rev.

---

setumotgrad

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

Syntax

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

Examples

Example

	Command	Description
1:	1.2 1 ***setumotgrad***	Set ***Motor voltage gradient*** 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 ***getumotgrad***	Returns ***Motor voltage gradient*** at axis 1.

Motor voltage minimum


storable

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. 494) and ***gi*** (p. 467) for current observation.

Commands

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 ***getumotmin***	Returns ***Motor voltage minimum*** at axis 2.

---

setumotmin

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

Syntax

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

Examples

Example

	Command	Description
1:	2.5 1 ***setumotmin***	Set ***Motor voltage minimum*** at axis 1 to 2.5 V.

Safety functions

Motor powerdown


not storable

Powerdown of motor to zero current. Same impact as a powerdown caused by a ***Machine error*** (p. 170) . Normal operation can be recovered by execution of ***Motor restart*** (p. 506) .

Commands

---

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.

Note that the restart will take a short period of time during which all move requests will be discarded. This is indicated by the *device busy* flag in the ***Axis status*** (p. 575) register.

Void if position sensor detection has failed at any time beforehand (resulting in ***Machine error*** (p. 170) code 101). In this case, the axis is in irreversible emergency-off state. This can be detected by evaluation of ***Axis status***, bit 15.

Commands

---

init

Upon execution, the following will happen:

- Motor powerup according to actual motor parameters (***Motor voltage minimum*** (p. 502) , ***Motor voltage gradient*** (p. 500)). Unwanted motion is reduced to a minimum.
- Reactivation of positioning controller according to currently requested ***Positioning control mode*** (p. 547) .

Syntax

{device} ***init***

Examples

Example

	Command	Description
1:	1 ***init***	Initiate ***Motor restart*** at axis 1.

Sensoric

Before use of position sensors, please take at a look at introductory chapter "Position sensor routing".

Position correction activation

not storable

Current activation state of position correction facility during operation.

Valid since ***Controller version*** (p. 163) 4.0000.

Note that availability and initial activation state depend on the storable ***Position correction parameters*** (p. 517) .

Commands

Properties

Type: int

value	description
0	inactive
1	active

---

getposcorr

Returns the current setting of the ***Position correction activation***.

Syntax

{device} ***getposcorr***

Reply

[state]

Examples

Example

	Command	Description
1:	1 ***getposcorr***	Returns ***Position correction activation*** of axis 1.

---

setposcorr

Activates or deactivates the position correction facility.

Blocked by default - available with a valid release code file only. See ***Released options*** on how to inquire available options.

Note that the position correction is protected by an activation mask. An activation request will only be granted on the following conditions:

- a valid correction data file must be available
- correction data must have been loaded
- with incremental position measurement, the correction home position must have been found

Syntax

[state] {device} ***setposcorr***

Examples

Example

	Command	Description
1:	0 2 ***setposcorr***	Deactivate position correction facility of axis 2.

Position correction argument

read-only

Current position argument fed into position correction facility (position measurement raw value).

Valid since ***Controller version*** (p. 163) 4.0000.

Commands

Properties

Type: double
Unit: mm

---

getposcorrarg

Returns current ***Position correction argument***.

Syntax

{device} ***getposcorrarg***

Reply

[argument]

Examples

Example

	Command	Description
1:	1 ***getposcorrarg***	Returns ***Position correction argument*** of axis 1.

Position correction file parameters

read-only

Correction table properties read from correction file header.

Valid since ***Controller version*** (p. 163) 4.0000.

Commands

Properties

i	Name	Type	Unit	Description
1	[alignment]	int	-	Selection of correction table home location. **value** / **table home location**: 1 scale origin; 2 calibration switch; 3 reference mark
2	[type]	int	-	Correction type selection. Invariably 1 with stand-alone Hydra controllers.
3	[start]	double	mm	Location of first node relative to home location.
4	[stepwidth]	double	mm	Distance between adjacent nodes.
5	[count]	int	-	Number of nodes.
6	[end]	double	mm	Location of last node relative to home location.

getposcorrfilepara

Returns the current setting of the ***Position correction file parameters***.

Syntax

[*i*] {device} ***getposcorrfilepara***

Reply

[*Value*]

Examples

Example

	Command	Description
1:	4 1 ***getposcorrfilepara***	Returns *stepwidth* entry of ***Position correction file parameters*** of axis 1.

Position correction parameters


storable

User accessible set of parameters for position correction.

The *autoload* entry defines whether correction data are loaded from the flash file system during powerup or not.

The *autoactivate* entry defines whether correction will automatically be activated as soon as the given activating condition is fulfilled; void if the *autoload* entry is set to 0 (because then the correction facility is not available).

Valid since ***Controller version*** (p. 163) 4.0000.

See ***Position correction activation*** (p. 510) on subsequent activation and deactivation.

Commands

Properties

i	Name	Type	Description
1	[autoload]	int	value: 0 = data load skipped during powerup; 1 = data loaded during powerup

<table>
<tr><th>i</th><th>Name</th><th>Type</th><th>Description</th></tr>
<tr><td>2</td><td>[autoactivate]</td><td>int</td><td>
<table>
<tr><th>value</th><th>home location*</th><th>correction activating condition</th></tr>
<tr><td>0</td><td>scale origin</td><td>faultless completion of powerup plus manual activation by setposcorr (p. 511)</td></tr>
<tr><td>0</td><td>calibration switch</td><td>faultless completion of Calibration move (p. 402) plus manual activation by setposcorr</td></tr>
<tr><td>0</td><td>reference mark</td><td>faultless completion of Reference move (p. 453) plus manual activation by setposcorr</td></tr>
<tr><td>1</td><td>scale origin</td><td>faultless completion of powerup</td></tr>
<tr><td>1</td><td>calibration switch</td><td>faultless completion of Calibration move</td></tr>
<tr><td>1</td><td>reference mark</td><td>faultless completion of Reference move</td></tr>
</table>
*see Position correction file parameters (p. 515) , alignment entry
</td></tr>
</table>

getposcorrpara

Returns the current setting of the specified ***Position correction parameters*** entry.

Syntax

[*i*] {device} ***getposcorrpara***

Reply

[*Value*]

Examples

Example

	Command	Description
1:	2 2 ***getposcorrpara***	Returns *autoactivate* entry of ***Position correction parameters*** of axis 2.

---

setposcorrpara

Sets the specified ***Position correction parameters*** entry.

Blocked by default - available with a valid release code file only. See ***Released options*** on how to inquire available options.

Syntax

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

Examples

Example

	Command	Description
1:	1 1 1 ***setposcorrpara***	Enable autoload function at axis 1.
2:	0 2 1 ***setposcorrpara***	Disable autoactivate function at axis 1.
3:	1 ***nsave*** (p. 365)	Store axis 1 parameters.

Upon next ***Reset***, the correction table will be loaded (if available). The correction facility, however, will remain inactive until activated manually using ***setposcorr*** (p. 511).

Position correction value

read-only

Current position correction value read from position correction table and applied to measured position; 0 if correction is inactive.

Valid since ***Controller version*** (p. 163) 4.0000.

Commands

Properties

Type: double
Unit: mm

---

getposcorrval

Returns current ***Position correction value***.

Syntax

{device} ***getposcorrval***

Reply

[value]

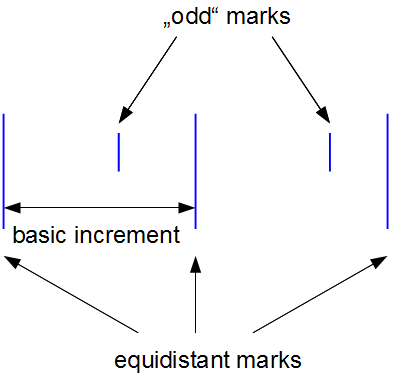
Examples

Example

	Command	Description
1:	2 ***getposcorrval***	Returns ***Position correction value*** of axis 2.

Scale basic increment

Dimension equivalent of the *equidistant* reference mark grid at the position scale assigned, measured in entire scale periods.



For instance, with the ***Scale period*** (p. 526) being 0.02 mm, the equidistant grid distance being 20 mm, and scale and distance coding counting in the same direction, the correct value would be +1000.

Written to ***Sensor cache*** (p. 532); not permanently stored by ***Configuration storage*** (p. 362) .

Must match the physical scale basic increment and effective coding direction. A mismatched setting will lead to erroneous absolute position determination.

Applicable to scales featuring multiple distance coded reference marks only.

Can be set with a negative sign in order to revert the count direction of distance coding while maintaining the sign of the ***Scale period***.

Available since firmware rev. 4.4000.

Commands

Properties

Type: int

---

setscalebasicinc

Sets the ***Scale basic increment***.

Syntax

[number of periods] {device} ***setscalebasicinc***

Examples

Example

	Command	Description
1:	1000 1 ***setscalebasicinc***	Sets the ***Scale basic increment*** of the scale assigned to axis 1 to 1000.

---

getscalebasicinc

Returns the current setting of the ***Scale basic increment***.

Syntax

{device} ***getscalebasicinc***

Reply

[number of periods]

Examples

Example

	Command	Description
1:	2 ***getscalebasicinc***	Returns the ***Scale basic increment*** of the position scale assigned to axis 2.

Scale period

Scale period of position sensor assigned to specified axis.

Written to ***Sensor cache*** (p. 532); not permanently stored by ***Configuration storage*** (p. 362) .

Must match the physical scale 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. 547) on how to open and close the position control loop.

Can be set with a negative sign in order to revert the count direction of position measurement.

Commands

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 ***getclperiod***	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

read-only

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

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. 529), 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. 526) and ***Scale basic increment*** (p. 522) .

Commands

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

Properties

Type: double
Unit: °C

---

gettemp

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

Syntax

{device} ***gettemp***

Reply

[temperature]

Examples

Example

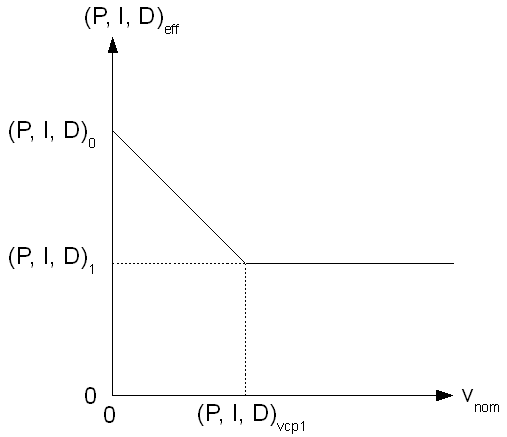
	Command	Description
1:	3 ***gettemp***	Returns ***Sensor temperature*** at Star interface port 3.

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. 565) 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. 547) for positioning controller mode selection.

Commands

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 ***getadaptive***	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 ***setadaptive***	Set integral portion final value of positioning controller axis 2 to 0.01.

FRT parameters


storable

Fast response technology features.

Valid since ***Controller version*** (p. 163) 4.0000.

Not applicable with stepper motors (s. ***Motor form*** (p. 484)) or ***Clock and direction function*** (p. 441) .

Commands

Properties

i	Name	Type	Unit	Description
1	[parameter1]	double	Vs²/mm	nominal acceleration to force feedforward coefficient
2	[parameter2]	double	Vs/mm	nominal velocity to force feedforward coefficient

---

getfrtpara

Returns the current setting of the ***FRT parameters***.

Syntax

[*i*] {device} ***getfrtpara***

Reply

[*Value*]

---

setfrtpara

Sets the ***FRT parameters***.

Syntax

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

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.

Do *only* use with stepper motors in order to protect motor and driver from sustaining damage!

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

Properties

Type: int

---

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 ***freeze***	Activates ***Positioning control freeze*** 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. 570) and ***Time on target*** (p. 572) 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 bidirectional function (all other register bits being set to default).
5:	4 1 ***setinitactmaskint*** (p. 320)	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.

---

getfreeze

Returns current ***Positioning control freeze*** mode activation state.

Syntax

{device} ***getfreeze***

Reply

[state]

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. 565) 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. 537) .

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

Commands

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. 506) *or*
- next ***Reset*** (when preceeded 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. 506)	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

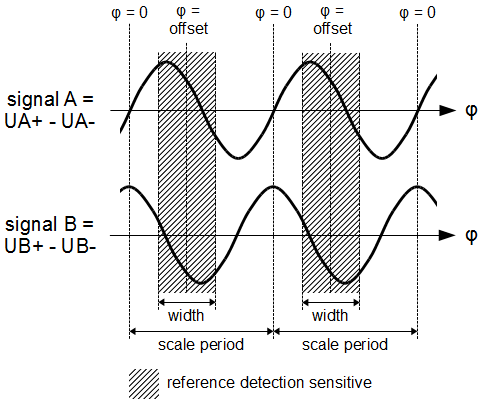
Example

	Command	Description
1:	1 ***getcloop***	Returns ***Positioning control mode*** at axis 1.

Reference window

storable

Window to define when the reference signal detection is sensitive during a ***Reference move*** (p. 453) , provided the *distance decoding* mode (***Motion function*** (p. 415) , bit 5) is selected. The reference signal detection is then masked periodically according to the following scheme:

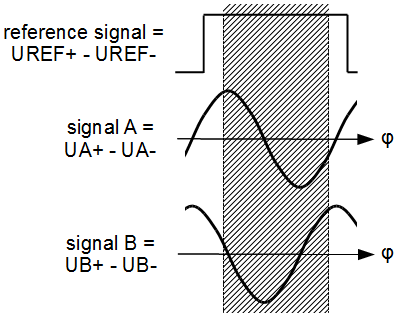


The parameters are both normalized to one entire scale period (value = 1.0). When distance decoding is off, window will be void.

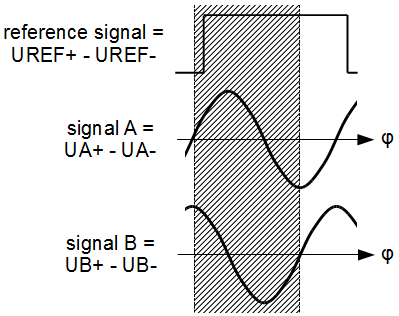
Note that correct parameter settings (i.e. matching the scale and sensor wiring) are mandatory for error-free, non-ambiguous and repeatable distance decoding, meaning that

- the set window must be covered *entirely* by *all* reference marks within the travel range to avoid distance decoding errors emerging from detection misses (as shown in *b)* below)
- the *width* parameter must be set smaller than 1.0; otherwise, the masking will be void, and the distance decoding will be ambiguous, especially if reference marks extend to an entire scale period or more (as shown in *c)* below)

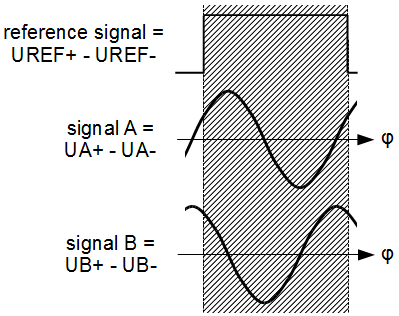
a)


reference signal =
UREF+ - UREF-
signal A =
UA+ - UA-
φ
signal B =
UB+ - UB-
φ

b)


reference signal =
UREF+ - UREF-
signal A =
UA+ - UA-
φ
signal B =
UB+ - UB-
φ

c)


reference signal =
UREF+ - UREF-
signal A =
UA+ - UA-
φ
signal B =
UB+ - UB-
φ

The best way to parameterize the window is to set the *width* sufficiently smaller than the reference mark width and align the window center to the the reference mark center by the *offset*, thus making the detection symmetrical (as shown in *a)* above).

Note that the ***Reference velocity*** (p. 431) setting must match the *width* parameter; *width* reduction will limit the possible maximum speed according to the following:

$$\mathbf{v < v_{max} = width * l_p * 4000 / [s]}$$

with l_p being the ***Scale period*** (p. 526) .

Example: With *width* set to 0.5 and a ***Scale period*** value of 0.02 (=> 20 µm),

v_{max} = 0.5 * 20 µm * 4000/s = 40 mm/s

Valid since firmware rev. 5.0300.

Commands

Properties

i	Name	Type	Unit	Description
1	[offset]	double	range: 0.0 .. 1.0	window center offset
2	[width]	double	range: 0.0 .. 1.0	window width

setrefwindow

Sets the selected ***Reference window***.

Syntax

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

Examples

Example

	Command	Description
1:	0.5 1 2 ***setrefwindow***	Sets the reference window center to the middle of the scale period (=> 180°) at axis 2.
2:	0.5 2 2 ***setrefwindow***	Sets the reference window width to half the scale period (=> 180°) at axis 2.

Result: The axis 2 reference detection will be sensitive in the 2nd and 3rd scale period quadrants (=> 90° through 270°).

---

getrefwindow

Returns the selected ***Reference window*** parameter setting.

Syntax

[*i*] {device} ***getrefwindow***

Reply

[*Value*]

Examples

Example

	Command	Description
1:	1 1 ***getrefwindow***	Returns the reference window width at axis 1.

Sensor assignment


storable

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

For proper operation of positioning control, make sure that

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

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

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

Commands

Properties

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

---

setassignment

Sets the ***Sensor assignment***.

For changing the assignment,

- first, open the positioning control loop using ***setcloop*** (p. 548)
- 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*** and wait until axis is powered up again
- check for proper and consistent connections and parameterization, correct if need be
- close the positioning control loop using ***setcloop***
- check if loop works properly, fix problems if necessary
- last, store the new setting with the position control loop now closed using ***save***

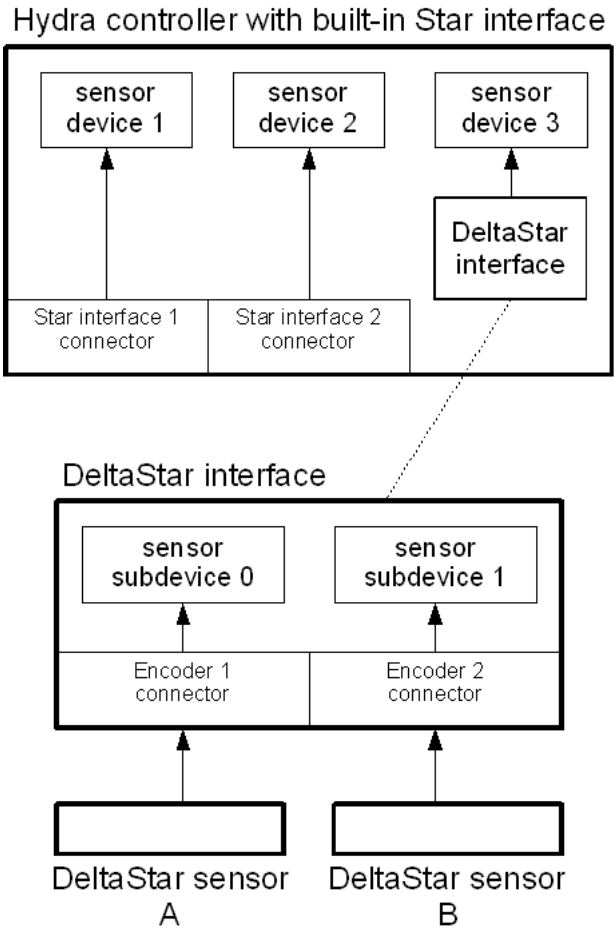
Syntax

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

Examples

Example 1

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

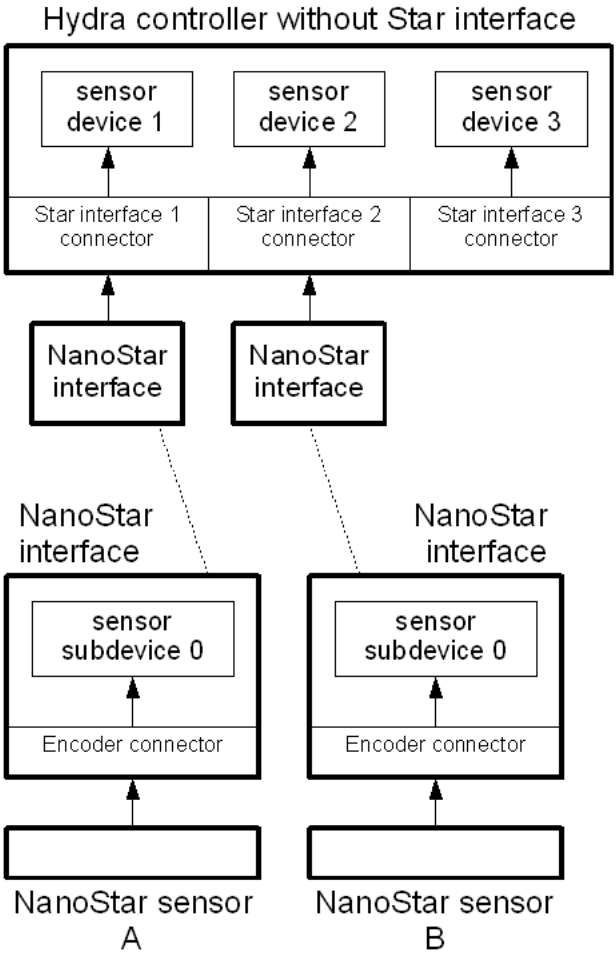


Task: The sensor A is to be assigned to axis device 1, sensor B to axis device 2.

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

Example 2

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



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

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

---

getassignment

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

Syntax

{device} ***getassignment***

Reply

[sensor subdevice] [sensor device]

Examples

Example

	Command	Description
1:	1 ***getassignment***	Returns ***Sensor assignment*** 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

Properties

Type: int

---

getsensorstatus

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

Syntax

{device} ***getsensorstatus***

Reply

[sensor status]

Examples

Example

	Command	Description
1:	2 ***getsensorstatus***	Returns ***Sensor status*** at axis 2.

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)*** and the differential portion ***(D limit)*** . These limiters are ineffective if respective parameters are set to 0. If nominal position and executive position differ by more than what ***max pos error*** specifies, the device will run into emergency off state (see also ***Motor restart*** (p. 506)).

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. 537) is an option to vary P, I and D factors dynamically depending on the nominal velocity.

The ***load distance*** parameter has a different function since version 3.000 where it specifies an overall output limitation of the positioning control. It is not effective with stepper motors; note that with motors of any other type, it is effective regardless whether or not the respective axis operates in closed loop mode.

The ***boost value*** and ***load distance*** parameters have to be handled with care. It is advised to use the ***Motor current limit*** (p. 478) option before changing these.

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

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

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

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

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. 547) .

Commands

Properties

i	Name	Type	Unit	Description
1	[P factor]	double	-	coefficient of proportional portion
2	[I factor]	double	-	coefficient of integral portion
3	[D factor]	double	-	coefficient of differential portion
4	[I limit]	double	[mm] or [V]	absolute integral portion limit
5	[boost value]	double	-	factor for load dependent voltage boost (linear motor drive only) - *void since firmware rev. 3.000*
6	[motor voltage limit]	double	s.above	*up to firmware rev. 2.411:* motor load distance limit (s. above) *since firmware rev. 3.000:* overall output limitation (s. above); not applicable to stepper motors

i	Name	Type	Unit	Description
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
13	[PID voltage limit]	double	[V]	PID sum limitation; void with stepper motors *valid since firmware rev. 4.2100*
14	[D limit]	double	[mm] or [V]	absolute derivative portion limit *valid since firmware rev. 4.4000*

---

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 ***getsp***	Returns differential portion of positioning controller at axis 1.

Target window


storable

Width of entrance and exit windows for the position settlement indication or in-window bit (see ***Axis status*** (p. 575) 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. 572) , the in-window bit changes from inactive to active state to indicate that the position has settled. As soon as the absolute position aberration once exceeds the exit window width, the indication bit changes from active to inactive state. Useful for closed loop operation only. Inactive if any parameter is set to 0.

Commands

Properties

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

---

getclwindow

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

Syntax

{device} ***getclwindow***

Reply

[entrance width] [exit width]

Examples

Example

	Command	Description
1:	1 ***getclwindow***	Returns ***Target window*** at axis 1.

---

setclwindow

Sets the ***Target window***.

Syntax

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

Examples

Example

	Command	Description
1:	.01 .05 1 ***setclwindow***	Set ***Target window*** 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. 570)).

Commands

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 ***getclwintime***	Returns ***Time on target*** at axis 2.

Status and position

Axis status

read-only

Actual status of the axis device. The flags of the 32 bit status word in detail:

Bit number	description
0	axis moving
1	manual move running
2	one or more machine errors occurred *
3	lower limit switch (Cal) position found
4	upper limit switch (RM) position found
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 - irreversible (reset required) **

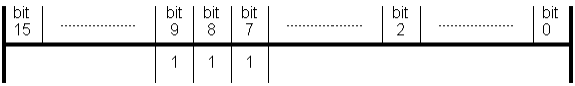
* status depends on controller, not on singular axis device

** indicates missing, improperly connected, misaligned or deficient position sensor, or a corrupt parameter file

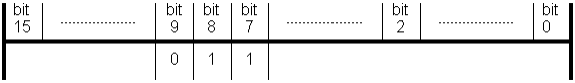
Note that the status is a bit-coded value which may be subject to functional extension (i.e. making use of hitherto unused bits) anytime. Therefore, status replies should always be evaluated bitwise instead of merely polling for absolute values!

Indication of reversible emergency conditions:

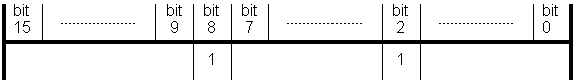
Current engagement of the emergency switch is shown by simultaneous activation of bits 9, 8 and 7:



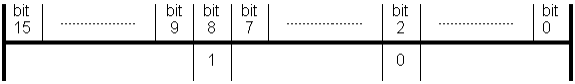
Recent engagement of the emergency switch is shown by bits 8 and 7 (as long as the motor has not been repowered):



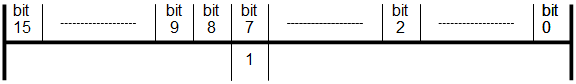
Emergency condition due to a machine error is shown by simultaneous activation of bits 8 and 2:



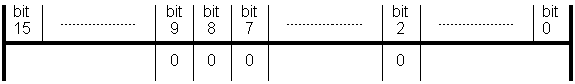
If the machine error cause has been removed and all machine errors entries have been popped off the machine error stack, bit 2 will be cleared:



Emergency condition due to a stop input going active is shown by isolated activation of bit 7:



As soon as the respective emergency condition has been removed, the ***Motor restart*** (p. 506) 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:



The status register is supplemented by a 16 bit machine error extension with ***est*** (p. 578) and ***ast*** (p. 579).

For watching the complete system without decoding each singular axis, see ***Controller status*** (p. 584) .

Commands

Properties

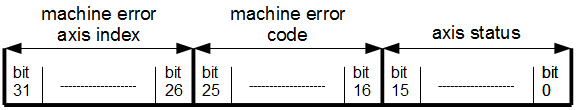
Type: int

---

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 10 bits)
- axis device index (upper 6 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 ***nst***	Returns ***Axis status*** at axis 1.

---

ast

Same as ***est*** (p. 578), 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.

Note that upon command execution, the interpreter will delay all command processing until the currently running move has finished, which means ***ast*** is not applicable with applications or environments requiring constant status, position or parameter inquiry during axis motion.

Syntax

{device} ***ast***

Reply

[status]

Examples

Example

	Command	Description
1:	2 ***ast***	Waits until axis 2 is in standstill. Then returns extended ***Axis status***.

Controller position

read-only

Integrated ***Device position*** (p. 588) of both axis devices. The single positions displayed conform the respective ***Position display selection*** (p. 594) settings.

Commands

Properties

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

---

p

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

Note that, with the ***Position cycle*** (p. 591) set to a valid value, the scale position may temporarily differ from the respective nominal position by one ***Position cycle*** near the reversal point.

Syntax

p

Reply

[position X] [position Y]

Examples

Example

	Command	Description
1:	***p***	Returns current ***Controller position***.

---

getrev

Returns ***Controller position*** equivalents in form of ***Position cycle*** based revolutions, reduced to whole turns (fractional parts discarded).

Will return zero if respective ***Position cycle*** is set to zero.

Available since firmware rev. 5.1000.

Syntax

getrev

Reply

[position X] [position Y]

Examples

Example 1

	Command	Description
1:	***getrev***	Returns the numbers of complete revolutions covered, related to the respective position origin.

Controller status

read-only

Partially a controller device status indication, partially a bitwise linkage of all ***Axis status*** (p. 575) 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	lower limit switch (Cal) position found	axis conjunction
4	upper limit switch (RM) position found	axis conjunction
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...14	reserved	-
15	invalid status - reset required*	axis disjunction

- 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. 584) registers.

Note that the status is a bit-coded value which may be subject to functional extension (i.e. making use of hitherto unused bits) anytime. Therefore, status replies should always be evaluated bitwise instead of merely polling for absolute values!

Commands

Properties

Type: int

---

st (status)

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

Syntax

st

Reply

[status]

Examples

Example

	Command	Description
1:	***st***	Returns ***Controller status***.

Device position


read-only

Actual device position, displayed selectively as the nominal or measured position (see ***Position display selection*** (p. 594)).

Commands

Properties

Type: double
Unit: mm

---

getnrev

Returns ***Device position*** equivalent in form of ***Position cycle*** based revolutions, reduced to whole turns (fractional part discarded).

Will return zero if ***Position cycle*** is set to zero.

Syntax

{device} ***getnrev***

Reply

[position]

Examples

Example 1

	Command	Description
1:	1 ***getnrev***	Returns the number of complete revolutions covered at axis 1, related to the position origin.

---

np

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

Note that - with the ***Position cycle*** set to a valid value - the scale position may temporarily differ from the nominal position by one ***Position cycle*** near the reversal point.

Syntax

{device} ***np***

Reply

[position]

Examples

Example

	Command	Description
1:	2 ***np***	Return ***Device position*** at axis 2.

Device target position

read-only

Target position of currently running move.

Commands

Properties

Type: double
Unit: mm

---

gettgtpos

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

Syntax

{device} ***gettgtpos***

Reply

[position]

Examples

Example

	Command	Description
1:	1 ***gettgtpos***	Returns ***Device target position*** at axis 1.

Position cycle


storable

Revolution interval of ***Device position*** (p. 588) and ***Controller position*** (p. 581) . Upon inquiry, the respective actual position will be subjected to a modulo operation with that interval before being returned, i.e. the displayed position will revolve (jump to zero) periodically, according to the parameter value. This is especially useful for rotational axes, where the displayed position can be parameterized to incorporate an equivalent of the motor angle.

Will invariably be zero with LMT specific firmware rev. 5.1100, thus disabling the rotatory axis function.

Constitutes a mere modulo operand, not a scaling factor; ***Pitch*** (p. 399) , and, if applicable, ***Scale period*** (p. 526) , must be matched accordingly.

With the parameter set to 0 (default), the position will never revolve.

Only affects position display; does not affect axis action or position handling. All move commands and commands handling position offsets and limits will still work with the standard position format; positioning is still absolute. With *c* being the parameter value, *r* being the axis turns counted and *p* being the displayed position (s. ***Device position*** / ***Controller position***), the actual position is *r* * *c* + *p*.

Available since firmware rev. 5.1000.

Commands

Properties

Type: double
Unit: mm

---

getposcycle

Returns the current setting of the ***Position cycle***.

Syntax

{device} ***getposcycle***

Reply

[cycle]

Examples

Example

	Command	Description
1:	2 ***getposcycle***	Returns ***Position cycle*** at axis 2.

---

setposcycle

Sets the ***Position cycle***.

Syntax

[cycle] {device} ***setposcycle***

Examples

Example

	Command	Description
1:	10 1 ***setposcycle***	Sets ***Position cycle*** at axis 1 to 10 mm.

Position display selection


storable

Selection of position displayed as ***Device position*** (p. 588) .

Commands

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 ***setselpos***	Set ***Position display selection*** 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 ***getselpos***	Returns ***Position display selection*** at axis 1.

Switches and reference mark

Reference configuration


storable

Defines the mode of operation for reference position mark detection. The setting takes effect upon execution of ***refmove*** (p. 446) and ***nrefmove*** (p. 456).

Commands

Properties

Type: int

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

---

setref

Sets ***Reference configuration***.

Syntax

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

Examples

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

Upon execution of ***refmove*** (p. 446) or 1 ***nrefmove***, device 1 will brake as soon as a reference signal high-to-low transition occurs, then return to respective position.

---

getref

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

Syntax

{device} ***getref***

Reply

[config]

Examples

Example

	Command	Description
1:	2 ***getref***	Returns ***Reference configuration*** at axis 2.

Reference status

read-only

Status of reference mark detection. Bits 0 and 1 are affected by ***Reference move*** (p. 453) only. Once set, they stay set until next execution of ***Reference move***.

Will invariably return zero with LMT specific firmware rev. 5.1100.

Commands

Properties

Type: int

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

---

getrefst

Returns ***Reference status***.

Syntax

{device} ***getrefst***

Reply

[status]

Examples

Example

	Command	Description
1:	1 ***getrefst***	Returns ***Reference status*** at axis 1.

Stop input configuration


storable

Bit-coded configuration of the input used for a special stop function triggered by the *Stop Motor input* (s. hardware manual for assignment). Polarity options are active high (AH) or active low (AL). Selectively induces a *real-time stop* or complete *motion blocking*.

When real-time stop function is armed, each signal level transition at the stop input matching the set polarity will induce a single stop of a *currently running programmed move* without blocking further moves. The stop request will be processed within 250 µs max. approx.

With motion blocking armed, the stop input signal going active will induce an emergency stop (which will not necessarily be executed in real time) and block all further programmed or manual motion. The motor axis will remain fixed. To remove the blocking condition,

- first, the stop input signal must return to inactive state
- second, ***Motor restart*** (p. 506) has to be executed

Bit number	description
0	polarity (0 = AH, 1 = AL)
1	mask (0 = enabled, 1 = masked)
2	function (0 = real-time stop, 1 = motion blocking)

Real-time stop available since firmware rev. 4.4000. Motion blocking available since firmware rev. 5.2200.

Commands

Properties

Type: int

---

setstopinconfig

Sets the ***Stop input configuration***.

Syntax

[stop input] {device} ***setstopinconfig***

Examples

Example

	Command	Description
1:	1 2 ***setstopinconfig***	Arm stop input at axis 2, active low polarity, for real-time stop function.

---

getstopinconfig

Returns the current setting of the ***Stop input configuration***.

Syntax

{device} ***getstopinconfig***

Reply

[stop input]

Examples

Example

	Command	Description
1:	1 ***getstopinconfig***	Return ***Stop input configuration*** 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. 408) alteration due to switch engagement.

Bit number	description
0	polarity (0 = NO, 1 = NC)
1	mask (0 = enabled, 1 = masked)
2	conditional mask (0 = enabled, 1 = masked)*

*switch will implicitly be enabled whenever a ***Calibration move*** (p. 402) is requested, and disabled upon move termination; available since firmware rev. 5.1000; void with LMT specific firmware rev. 5.1100.

Calibration move and ***Range measure move*** (p. 425) will not work properly when the respective limit switch is disabled. See ***Motion function*** (p. 415) 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. 607) .

Commands

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 ***getsw***	Returns ***Switch configuration*** of calibration switch at axis 1.

Switch status

read-only

Engagement states of limit switches.

Commands

Properties

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

---

getswst

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

Syntax

{device} ***getswst***

Reply

[calibration sw.] [range measure sw.]

Examples

Example

	Command	Description
1:	1 ***getswst***	Returns ***Switch status*** at axis 1.

Trigger

Before use of the trigger functions, please take at 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.

Available with *DeltaStar trigger* only.

Commands

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 ***gettrinsize***	Return ***Trigger capture buffer size*** at axis 1.

Trigger capture index


not storable

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

Available with *DeltaStar trigger* only.

Commands

Properties

Type: int

---

gettrinindex

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

Syntax

{device} ***gettrinindex***

Reply

[index]

Examples

Example

	Command	Description
1:	2 ***gettrinindex***	Return ***Trigger capture index*** at axis 2.

Trigger capture mode


not storable

Enable state of the trigger capture function.

value	capture function
0	off
1	on

Available with *DeltaStar trigger* only.

Commands

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 ***gettrin***	Return ***Trigger capture mode*** 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)

Available with *DeltaStar trigger* only.

Commands

Properties

Type: int

---

gettrinpol

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

Syntax

{device} ***gettrinpol***

Reply

[input polarity]

Examples

Example

	Command	Description
1:	1 ***gettrinpol***	Return ***Trigger capture polarity*** at axis 1.

---

settrinpol

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

Syntax

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

Examples

Example

	Command	Description
1:	1 2 ***settrinpol***	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).

Available with *DeltaStar trigger* only.

Commands

Properties

Name	Type
[index]	int
[position]	double

---

gettrinpos

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

Syntax

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

Reply

[position]

Examples

Example

	Command	Description
1:	7 2 ***gettrinpos***	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. 621) 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. 619), this saves communication time, and is particularly useful when a record contains a large number of captured values.

Available with *DeltaStar trigger* only.

Commands

Properties

Type: int

---

filetrinpos

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

The error code returned can be decoded as follows:

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

Syntax

{device} ***filetrinpos***

Reply

[error code]

Examples

Example

	Command	Description
1:	2 ***filetrinpos***	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. 640) instead.

Commands

Properties

Type: double
Unit: 0.5 µs 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 delay compensation


storable

Compensation for fixed delays within the measurement system and the user's electronics attached to the trigger output.

Available with *DeltaStar trigger* only.

Based on velocity estimation; yields best results while scale velocity is approx. constant. Will decrease in accuracy the more rapidly scale velocity changes.

Parameterization must match the real conditions. A mismatch might result in a loss of trigger events.

Commands

Properties

Type: int
Unit: µs
range: 0...255

---

settbcomp

Sets the ***Trigger delay compensation***.

Syntax

[ticks] {device} ***settbcomp***

Examples

Example

	Command	Description
1:	5 1 ***settbcomp***	Sets ***Trigger delay compensation*** at axis 1 to 5 µs.

---

gettbcomp

Returns the actual ***Trigger delay compensation***.

Syntax

{device} ***gettbcomp***

Reply

[ticks]

Examples

Example

	Command	Description
1:	1 ***gettbcomp***	Returns actual ***Trigger delay compensation*** at axis 1.

Trigger event setup


not storable

Trigger event generator setup, valid either with **DeltaStar trigger in equidistant mode** (see ***Trigger mode*** (p. 629)) or with **Onboard trigger**.

Commands

Properties

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

---

settrpara

Sets the ***Trigger event setup*** and arms the trigger. Void with **DeltaStar trigger** if ***Trigger mode*** (p. 629) 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

[start position] [stop position] [number] {device} ***settrpara***

Examples

Example

	Command	Description
1:	10 20 11 2 ***settrpara***	Arm trigger for a pulse sequence of 11 equidistant events 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 ***gettrpara***	Return ***Trigger event setup*** at axis 2.

Trigger mode

not storable

Output trigger operation mode and activation state. Distinguishes between *DeltaStar* and *Onboard* trigger functions. The *DeltaStar* trigger allows for several different trigger patterns, while event generation with the *Onboard* trigger is always based on equidistant position intervals.

Preliminary notes concerning the equidistant and table trigger function principles

With the equidistant and table trigger functions, 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.

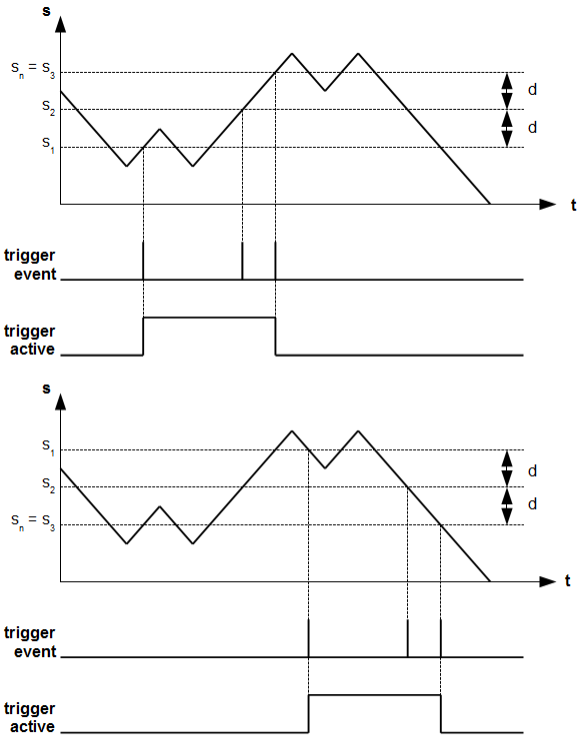
With the *equidistant* trigger function enabled and the slide/ rotor moving in a given direction, the trigger event generator

- gets active generating a trigger event as soon as the currently measured 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 currently measured position reaches a given stop position (trigger sequence termination)

The trigger event parameter set includes

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

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



Equidistant trigger principle

With the *table* 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 currently measured position reaches the first table trigger position (trigger sequence start)
- generates further single trigger events as specified by the table entries in consecutive order
- gets inactive generating a trigger event as soon as the currently measured position reaches the last table trigger position (trigger sequence termination)

The trigger event parameter set of a specified trigger point **n** includes

- the trigger position $\mathbf{s_n}$
- the output 2 level $\mathbf{l_n}$

The appropriate moving direction is determined by the distance between the first trigger positions $\mathbf{s_2}$ and $\mathbf{s_1}$. The trigger events are always processed in one direction from $\mathbf{s_1}$ through $\mathbf{s_n}$, as shown in the example s/t profile (**n** = 3) below. Accordingly, the course of the trigger positions has to be strictly monotonic over the table index; otherwise, trigger output behaviour will be indeterminate.

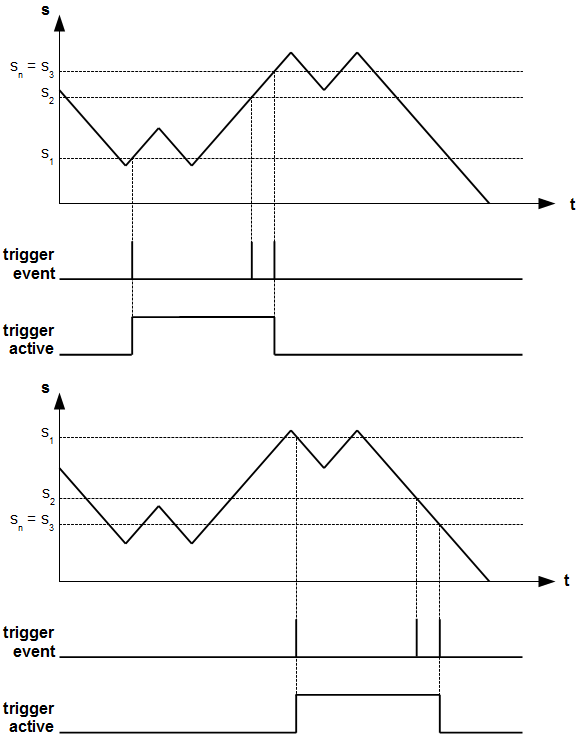


Table trigger principle

See ***Trigger event setup*** (p. 627) for equidistant trigger event configuration.

See ***Trigger table point setting*** (p. 656) for table trigger event configuration.

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.

DeltaStar trigger

As for trigger event generation, there are 3 modes:

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

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

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

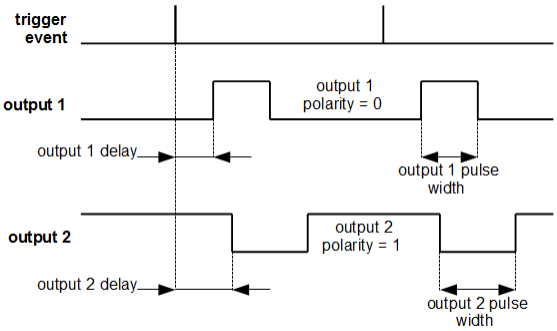
Available mode combinations and parameterization:

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

With the **equidistant** and **table** modes, the trigger event generator works as described in the preliminary notes above.

With the **continuous** mode, the trigger event generator puts out continuous asynchronous events at a constant frequency of approx. 50 Hz. It does not need further parameterization.

With the **standard** mode, both outputs operate equally, individually configurable regarding ***Trigger output delay*** (p. 640), ***Trigger output pulse width*** (p. 645) and ***Trigger output polarity*** (p. 642).

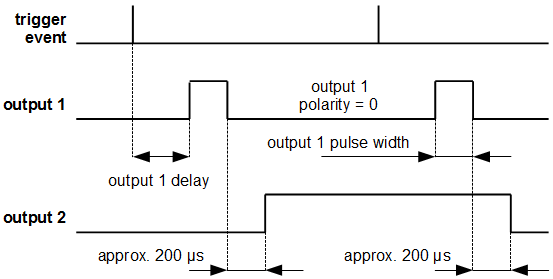


Standard mode timing scheme

With the **direction** mode, output 1 operates as it does in standard mode, whereas output 2 puts out a customer specific signal used to control a writing head. Output 2 behaviour depends on the trigger event generation selected:

- With equidistant trigger events, the direction signal at output 2 will alternate with each single trigger event, according to the timing scheme below. ***Trigger output polarity*** specifies the start polarity, while output 2 ***Trigger output delay*** and ***Trigger output pulse width*** are void.

- With table trigger events, the direction signal at output 2 will update according to the individual output 2 levels from the table and the timing scheme below. Via ***Trigger output polarity***, the overall signal polarity can be altered. All other parameters specified for output 2 (***Trigger output delay*** and ***Trigger output pulse width***) are void.



Direction mode timing scheme

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

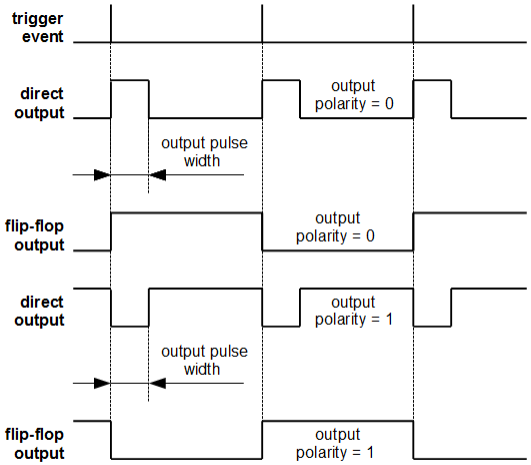
For query whether a trigger sequence is currently active (running), see ***Trigger status*** (p. 650) .

Onboard trigger

The onboard trigger pulse sequences are always based on equidistant trigger positions. Event generation works as described in the preliminary notes above. Mode provides trigger activation and output assignment, and optional

involvement of a virtual downstream flip-flop. Available modes:

value	output selection
0	no output (off)
1	Trigger 1
2	Trigger 2
3	Trigger 1 with downstream flip-flop
4	Trigger 2 with downstream flip-flop



Onboard trigger output action

With the flip-flop feature enabled, the logical state of the trigger output signal will be inverted upon arrival at every new trigger position as shown above, where the default state is passive according to ***Trigger output polarity***; it will go active with each odd and passive with each even trigger position index, starting at 1. In this case, ***Trigger output pulse width*** is void. Otherwise, each arrival at a new trigger position will trigger a single pulse according to ***Trigger output polarity*** and ***Trigger output pulse width*** settings.

Commands

Properties

Type: int

---

settr

Sets the ***Trigger mode***.

Syntax

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

Examples

Example

	Command	Description
1:	1 2 ***settr***	*DeltaStar trigger:* Set ***Trigger mode*** to equidistant/standard mode for axis 2 trigger sequences. *Onboard trigger:* Activate output Trigger 1 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 ***gettr***	Return ***Trigger mode*** at axis 2.

Trigger output delay

not storable

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

Available with *DeltaStar trigger* only.

Commands

Properties

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

---

settroutdelay

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

Syntax

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

Examples

Example

	Command	Description
1:	10 1 2 ***settroutdelay***	Set ***Trigger output delay*** at output 1 to 10 µs 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 ***gettroutdelay***	Returns ***Trigger output delay*** of output 1 for axis 2 trigger sequences.

Trigger output polarity

not storable

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

Commands

Properties

i	Name	Type	Unit	Description
1	[output 1 polarity]	int	-	value: 0 = active high; 1 = active low (All trigger modes)
2	[output 2 polarity]	int	-	value: 0 = active high (start low), 1 = active low (start high) (DeltaStar trigger standard mode / DeltaStar trigger direction mode) *void with Onboard trigger*

---

gettroutpol

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

With *Onboard trigger*, will return -1 if ***Trigger output polarity*** was attempted to be set with no valid trigger output assigned.

Syntax

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

With *Onboard trigger*, output index *i* must always be 1 because the trigger unit only drives a single trigger output at a time.

Reply

[*Value*]

Examples

Example

	Command	Description
1:	1 1 ***gettroutpol***	Returns ***Trigger output polarity*** of output 1 for axis 1 trigger sequences.

---

settroutpol

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

Syntax

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

With *Onboard trigger*, output index *i* must always be 1 because the trigger unit only drives a single trigger output at a time.

Examples

Example for DeltaStar trigger

	Command	Description
1:	0 2 1 ***settroutpol***	Set ***Trigger output polarity*** at output 2 to active high for axis 1 trigger sequences.

Example for Onboard trigger

	Command	Description
1:	0 1 2 ***settroutpol***	Set ***Trigger output polarity*** at output assigned to axis 2 to active high for trigger sequences.

Trigger output pulse width

not storable

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

Note that, with DeltaStar trigger, pulse width is limited to 127,5 µs max.

Commands

Properties

i	Name	Type	Unit	Description
1	[output 1 width]	double	µs	-
2	[output 2 width]	double	µs	*void with Onboard trigger*

---

settroutpw

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

Syntax

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

With *Onboard trigger*, output index *i* must always be 1 because the trigger unit only drives a single trigger output at a time.

Examples

Example for DeltaStar trigger

	Command	Description
1:	5 2 2 ***settroutpw***	Set ***Trigger output pulse width*** at output 2 to 5 µs for axis 2 trigger sequences.

Example for Onboard trigger

	Command	Description
1:	500 1 2 ***settroutpw***	Set ***Trigger output pulse width*** at output assigned to axis 2 to 500 µs for trigger sequences.

---

gettroutpw

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

With *Onboard trigger*, will return 0 if ***Trigger output pulse width*** was attempted to be set with no valid trigger output assigned.

Syntax

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

With *Onboard trigger*, output index *i* must always be 1 because the trigger unit only drives a single trigger output at a time.

Reply

[*Value*]

Examples

Example

	Command	Description
1:	2 1 ***gettroutpw***	Returns ***Trigger output pulse width*** 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. 645) instead.

Commands

Properties

Type: double
Unit: 0.5 µs 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


not storable

Current trigger output status. When the output trigger function is active, goes 1 after first event of a trigger output sequence, and 0 after the last. For output trigger activation, see ***Trigger mode*** (p. 629) . For trigger event configuration, see ***Trigger event setup*** (p. 627) .

With *DeltaStar trigger*, only available with firmware revisions higher than 4.3000.

Commands

Properties

Type: int

---

gettrst

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

Syntax

{device} ***gettrst***

Reply

[status]

Examples

Example

	Command	Description
1:	2 ***gettrst***	Return ***Trigger status*** at axis 2.

Trigger table emptying

Entry to clear the trigger table.

Available with *DeltaStar trigger* only.

Available since firmware rev. 5.1000.

Commands

Properties

Type: int

---

cleartrtable

Clears the trigger table. The table will be empty afterwards.

Syntax

{device} ***cleartrtable***

Reply

[reserved]

Examples

Example

	Command	Description
1:	1 ***cleartrtable***	Clears the trigger table at axis 1.

Trigger table point inquiry

not storable

Entry to inquire a specific point in the trigger table.

Available with *DeltaStar trigger* only.

Available since firmware rev. 5.1000.

Commands

Properties

Name	Type	Unit	Description
[index]	int	-	range: 0..3582
[position]	double	[mm]	-
[output 2 level]	int	-	0: low level 1: high level

---

gettrpoint

Inquires the parameters of the specified trigger point.

Syntax

[index] {device} ***gettrpoint***

Reply

[position] [output 2 level]

Examples

Example

	Command	Description
1:	23 2 ***gettrpoint***	Returns the settings of trigger point 23 at axis 2.

Trigger table point setting

not storable

Entry to set a specific point in the trigger table.

Available with *DeltaStar trigger* only.

Available since firmware rev. 5.1000.

Commands

Properties

Name	Type	Unit	Description
[output 2 level]	int	-	0: low level 1: high level
[position]	double	[mm]	-
[index]	int	-	range: 0..3000
[result]	int	-	-

---

settrpoint

Sets the parameters of the specified trigger point.

Syntax

[index] [position] [output 2 level] {device} ***settrpoint***

Reply

[result]

Examples

Example

	Command	Description
1:	56 123 0 2 ***settrpoint***	Sets the parameters of trigger point 56 at axis 2 to 123 mm and low level at trigger output 2.

Trigger table transmission

Entry to transmit the trigger table to the DeltaStar trigger unit.

Available with *DeltaStar trigger* only.

Available since firmware rev. 5.1000.

Commands

Properties

Type: int

---

sendtrtable

Transmits the trigger table to the DeltaStar interface assigned.

Syntax

{device} ***sendtrtable***

Reply

[reserved]

Examples

Example

	Command	Description
1:	1 ***sendtrtable***	Transmits the axis 1 trigger table to the DeltaStar interface assigned.

State Reference

Command Reference