Precision Positioning Systems

LSTEP *express*

LSTEP PCI *express*

LANG GMBH & CO. KG
Dillstrasse 4
D-35625 Hüttenberg
Tel. +49 6403 7009-0
Telefax +49 6403 7009-40

Dear Customer!

Thank you that you decide to buy a controller from our company!

With this unit you have selected a positioning system which executes automatically at low space ambitious positioning tasks. The high precision of the controller opens up wide application possibilities. The step resolution up to 1.638.400 steps per motor revolution, for a 200-step motor, offer resolutions in sub-µm-range. The „closed loop“ operation in combination with a high-resolution encoder evaluation at optical and magnetic measuring systems, offers very high positioning accuracy.
The many additional functions, as for ex. Snapshot, Trigger output, clock pulse, direction of rotation in- and outputs, make the controller to an ideal partner for many applications.

Before installation of your controller you are kindly asked to read carefully this manual.

Please pay particular attention to the safety instructions!

We reserve the right for modifications. We are not liable for any errors in this documentation. Due to continuous, further technical development of our products, there may be minor differences between the descriptions in this documentation and your actual controller. A liability for consequential damages, which occur in connection with the shipment or the use of this documentation, is excluded, as far as it is legally admissible.

Protection notice according to DIN 34

Contents

Page

Contents

1 Safety Instructions

- Maintenance and repair work must only be performed by qualified personnel, which has a special training and is very familiar with the controller, and only after written acceptance of the company Lang GmbH & Co. KG. All other person repairs are prohibited.

- **LSTEPexpress:** Before opening the unit, the power plug has to be pulled!

- **LSTEP-PCIexpress: -**Switch before and during installation of the card in the PC, as well as during installation of the accessories all parts to neutral. –Install the card in such a way so that no chips, liquids or other objects come into contact with the card. –Install the card in such a way that no heat accumulations arise. The max. ambient temperature can be found in the technical data. –Make sure that the current supply of your PC and, if available, the external motor voltage power rack is sufficient for the operation of the card. –Before switch-on make sure that the installation of all components is correct.

- The controllers have parts that react sensitive to electrostatic discharge (ESD-sensitive). Earth all parts which come into contact with the controller, also oneself.

- Only units which are specified by company Lang may be connected. Infringements can lead to destruction of the controller resp. the connected unit!

- Please make sure that the controller in connection with its application corresponds with the applicable safety regulations and legislation. This controller is compliant to EN61010-1:2001 (Safety Regulations for electrical measurement-, control-, control- and laboratory units). Please bear in mind that the effective motor voltage can correspond in its height the DC-tension on power connector for motor voltage.

- Dangerous movement: Depending on the configuration it might be that the Joystick is active after switch-on. Please make sure that no dangerous movements may arise, neither in this, nor in another operation mode.

- Parts of the controller can be very hot during operation. Prevent from touching the board during and shortly after operation. A cooling time of min. two minutes at ambient temperature is observed. Avoid the contact of subjects with the surface of the board.

- The power switch of the controller or the socket where the controller is connected, should always be accessible, so that at any time the controller can be separated from the network at all poles!

- **Under power no cable may inserted or removed!**

2 Description of the function

The stepper motor controller LSTEPexpress (-PCIexpress) are used for coordinate tables for for ex. microscopes or production processes with resolutions up to 0.00001 mm. The controller is characterized by smooth running of the motors. Due to the dynamic micro step-drive-principle one can reach high motor speeds up to 70 R/sec for a 200-step motor, despite a high resolution up to 1.638.400 micro steps each motor revolution.
The controller works with linear interpolation (all axes reach the target position at the same time), whereby the axes are started jointly. All axes can also be positioned asynchronous. The controller contains an automatic, free programmable ramp generation and jerk limitation. Acceleration and delay can be adjusted separately. The LSTEPexpress can be single operated or via a PC. A position display (option) on the front panel as well as a „Joy-Stick“ complements the unit. The instruction set of the LSTEPexpress (-PCIexpress) follows the instruction set of the LSTEP series, which has been extended by significant new functions.
For a clean process and a safe positioning one should use motors with a Stepp angle error of < ± 3%. In order to reach the max. number of revolution, one should use motors with low resistance with low inductance.
In order to avoid an unnecessary heating of the motors, the LSTEPexpress (-PCIexpress) reduces in each production stop (also in Joy-Stick - operation) the motor current to the adjusted standby-current.

2.1 Interface

A USB , the RS232 interface, or the PCIexpress bus are available as standard interfaces to the higher-level PC.

2.1.1 Operation without controller calculator

Simple movements are feasible with the LSTEPexpress also without controller calculator. The joystick is configured so that it is activated when the controller is switched on. Now you can position with the Joy-Stick any positions. On controller with LC-Display the current absolute position is constantly displayed. On top of that the axes can individually be set to zero using the button "CLEAR".

2.2 Control elements

The display (only for units with display) and all control elements, except the power switch, are on the front panel.

Not presently available.

3 Commissioning

3.1 Connections

- Connect motors via the supplied cable.
- Connect incremental measuring systems (if available).
- Connect Joy-Stick and lock with the slide valves.
- Connect processor via interface cable (only for LSTEPexpress).
- Connect network (only for LSTEPexpress) .
- 12V supply via 4pol PC-power pack plug to ST10 (only for LSTEP-PCIexpress)
- If necessary motor voltage >12V…48V from ext. power pack to ST17 (only for LSTEP-PCIexpress)

3.2 Connection incremental measuring systems

Incremental rotary- or linear encoder systems for recognition resp. avoidance of a step displacement can be connected on the controller. This enables a closed-loop operation. In this case the applications are not limited to optical measuring systems. For ex. also inductive or magneto resistive systems can be analysed, provided that their output signals keep the specified limit values. The optional encoder interface enables the connection of encoder systems with sinusoidal output.

3.3 Function test

- Switch-on device
  After each switch-on, the controller makes automatically a calibration of the connected Joystick, which needs about 5s. In order to guarantee a correct calibration, the Joystick may not deflect during that time.
- The LSTEPexpress does not standardly have a joystick switch, but can be configured such that the joystick is activated when the controller is switched on. If the joystick is not active, additional tests should be performed with the WinCommander.
- Deflect Joy-Stick in all directions: the motors run according to the deflection. If no response, check motor and Joy-Stick connections. If the connections are okay, then check via the supplied WinCommander the limit switch polarity. Will this also not be of any success; the unit should be checked for hidden transport damages.
- Test with the supplied WinCommander or function call (see instruction) via your own software.

3.4 Firmware update

The controller can be updated very easy with the latest program versions.

- Copy the Flashtool„TIC2000“on your PC.
- Copy the new firmware „*.hex“ on your PC
- Open the Flashtool
- Select the interface of the PC you have connected with the controller,
- Make all settings as described under support.
- The Dip-button "1" on the back wall of the LSTEPexpress must be placed on ON and the controller has to be switched on. At LSTEP-PCIexpress you make after switch-on of Dip-button "1" with Dip-button "2" a Reset (switch on and out) ".
- Click on the button „Program*.hex
- Select a new firmware and click „Open“.
- If the download process is finished, the Dip-button "1" must be restored to its original position.
- After a second pressing of the Reset button or the switch-on and –out of the controller, the controller works with the new firmware.

4 The LSTEP Controller Instruction Set

For better clarity, all instructions and parameters, which are sent to the controller and all acknowledgements/feedback's from the controller, are transmitted as ASCII characters. The advantage of this is that on the one hand, the commands can be input manually at a normal terminal. On the other hand, these plain language commands make troubleshooting easier, when a customised program sets the commands.

Commands or parameters which are transmitted to the controller begin with an exclamation mark "!". Inquiries are denoted with a question mark "?" . For example, the following mean:

!cal	***Calibrate***
?status	***Read out status***

Information: For write-only or read-only instructions, the characters "!" or "?" may be omitted.

Some instructions, e.g. specification of travels, require the transmission of parameters. These are then transmitted after the instruction itself. A space must be inserted and transmitted between the command text and the parameters and between the various individual parameters to separate them.

moa 45 13 20 ***Move x, y and z to the positions 45, 13 and 20***

Each instruction must be concluded with a carriage return (CR). This character is shown in the ASCII character set as follows:

Symb. Name	dec. value	Hex. value	Bin. value
CR	13	0xD	00001101

4.1 Short description of the LStep express Instruction set

Command	Example	Note	Chap. 4 Page
Interface			
baud	(?) !baud 9600	set baud rate to 9600	12

Interface			
ver	?ver	read version number	9
iver	?iver	Read out Internal Version number	9
det	?det	read out detailed version number	9
readsn	?readsn	Read serial number of the controller	11

Settings			
configmaxaxis	(?)!configmaxaxis 4	A 4-axis controller is operated with 4 axes	17
dim	(?) !dim 1	setting the unit in µm	18
pitch	(?) !pitch 1 1 1 (y 4)	setting the spindle lead X Y Z or only Y	19
geardenominator	(?) !geardenominator	gear factor / Denominator	19
gearnumerator	(?) !geardenominator	gear factor / Enumerator	20
accel	(?) !accel 1 1 1 (x 1)	setting the acceleration X Y Z or only X	21
acceljerk	(?) ! acceljerk	Jerk acceleration	22
decel	(?) ! decel	Delay	23
deceljerk	(?) ! deceljerk	jerk delay	24
vel	(?) !vel 10 10 10 (x 20)	setting the speed X Y Z or only X	25
pot	(?) !pot 1(0)	switch Speedpoti ON/OFF	51
motorcurrent	(?) !motorcurrent 1 2 2.5	motor current setting: X=1A Y=2A Z=2.5A	26
maxcur	?maxcur	all max. currents are indicated (=configuration)	26
reduction	(?) !reduction 0.5 0.5 0.5	current reduction to 50% in all axes'	27
curdelay	(?) !curdelay 1000	deceleration of the current reduction (0 - 10000 ms)	27
axis	(?) !axis1 0 1 (y 1)	switch axis ON/OFF	28
axisdir	(?) !axisdir 0 1 0	motor turning direction turned of Y-axis incl. limit switch	28
Motorfielddir	(?) ! Motorfielddir	only exchange motor direction of rotation	29
Swchange	(?) ! Swchange	only exchange limit switch	30
caliboffset	(?) !caliboffset 1 1 1 1	The zero position will be shifted 1mm with Dim 2	43
rmoffset	(?) !rmoffset 1 1 1 1	The end position will be shifted 1mm with Dim 2	43
caldir	(?) !caldir z 1	the Z-axis will be calibrated in positive direction	44
calibrmbspeed	!calibrmbspeed 10	at "cal"+"rm" will be moved with 0.1R/s out of the limit switches (5...100)	45
calibrmaccel	(?) ! calibrmaccel	acceleration for calibration and stroke measurement	46
calibrmjerk	(?) ! calibrmjerk	jerk for calibration and stroke measurement	47
calibrmvel	(?) ! calibrmvel	speed for calibration and stroke measurement	48
Save	save	the current parameters are burned into the Flash	12

Command	Example	Note	Chap. 4 Page
Settings			
reset	reset	the software is set back to the start condition	11
pa	pa 1	Power amplifier 1 = Power amplifier switch on 0 = switched off	11
motorbrake	(?)!motorbrake z 1	The brake is activated when the Z - power amplifier is switched off	33
motorbrakeswitchondelay	(?)!motorbrakeswitchondelay	Delays the output when the power amplifier is switched on	34
motorbrakeswitchoffdelay	(?)!motorbrakeswitchoffdelay	Delays the output when the power amplifier is switched off	34
stoppol	(?) !stoppol 0 oder 1	The stop input (MFP) is low or high active	39
stopdecel	(?) ! stopdecel 2	the brake acceleration, when Stopp entry will be active, is 2m/s²	40
stopdeceljerk	(?) ! stopdeceljerk	Adjusted jerk, for active Stop entry	41
quit	!quit	After a fault has occurred and has been rectified, confirmation must be given with "quit"	35
validconfig	!validconfig	Take over the settings	35
validpar	!validpar	Activation: of the controller parameters	35
motorpolepairs	!motorpolepairs 50	Setting for a 200-step (1.8°)motor	30
motorpolepairres	!motorpolepairres 1000	The resolution is 1000 microsteps / signal period	31
motormaxvel	!motormaxvel x 1000	The variable speed for the X-motor is limited to 1000 rpm	31
motortype	!motortype 0 or 1	0 = 2 Phase rotary stepper motor 1 = 2 Phase linear stepper motor	32

Status request			
Autostatus	(?) !autostatus 0 (0-4)	setting the acknowledgement of the controller	13
Status	?status	shows the current condition of the controller	13
Statusaxis	?statusaxis	current condition of each axis (@,M,J,C,S,A,D,-,)	15
err	?err	shows the current error number	16
statuslimit	?statuslimit	A=calibrated; D=measured table stroke; L=Software end position;-=basic setting.	37
Sysstatus	?sysstatus	System status with text feedback	13
Sysstat	?sysstat	System status with numeric feedback	14

Command	Example	Note	Chap. 4 Page
Moving commands and Position administration			
cal	!cal	Calibrate	42
rm	!rm	measure table stroke	42
moa	!moa 10 10 10 (x 10)	drive to absolute position X Y Z (only X)	49
mor	!mor 4 4 4 (y 4)	relative-positioning X Y Z (only Y)	49
m	!m	start of a move (track with mor or.distance)	50
distance	(?) !distance	set the track for X Y Z (start with "m")	50
a	!a	Abort (Stop)	52
pos	(?) !pos 0 0 0 (z 0)	set or read position	51
Programming of and movement to table positions			
tpos	!tpos 1 10 20 30 40	programming of table position 1 for all 4 axes to the position x y z a / 10 20 30 40	52
	?tpos 2 2 2 2	query table position 2 of all axes	
mtpa	!mtpa 1 1 1 1	moves all 4 axes to table position 1	53
idt	!itd a 50	One revolution of the a-axis is divided up into 50 equal angles of 7.2° each	53
mita	!mita a 10	The a-axis is moved to position 10 (72° for itd=50).	54
Joystick			
speed	(?) !speed 5 5 5 (y 10)	digital Joystick, all axes are turning with 5R/s od.Y with 10	55
joydir	(?) !joydir 1 1 -1	set motor turning direction for the Joystick	56
joyvel	(?) ! joyvel 10 10 10	the max. speed in Joystick operation is 10 rps	56
joyoutpass	(?)!joyoutpass x 100	The filter time constant for input values is 100µs	57
joyenable	!joyenable 0 or 1	Joystick is not enabled or not connected	58
joyredcur	!joyredcur 0 or 1	Joystick without or with current reduction	57
joy	(?) !joy 1 or 0	switch joystick ON/OFF	59
joywindow	(?) !joywindow 10	set range where the axis move (0-100)	58
joytoaxis	!joytoaxis 1 2 3	Joystick X to Axis 1, Y to Axis 2, Z to Axis 3 Each joystick axis can be freely assigned 0 = no joystick axis	59

Command	Example	Note	Chap. 4 Page
Manual operation using a trackball, handwheel or jog mode			
tippvel	(?)!tippvel 10 10 10 10	Jog mode speed	60
tippoutpass	(?)!tippoutpass x 10	The filter time constant is 10µs	60
tippdir	(?)!tippdir 0 1 0 0	Jog mode direction	61
tippenable	(?)!tippenable 1 1 1 0	Enable jog mode	61
tippredcur	(?)!tippredcur 1 1 1 0	Jog mode with current reduction	62
tipp	!tipp o or 1	Switch on jog (or inching) mode	62
tbenable	!tbenable x 0 or 1	Switch trackball mode off and on	64
tbtoaxis	!tbtoaxis x 0 or 1 or 2	Assignment of the trackball axis - Off - horizontal - vertical	65
tbdir	!tbdir x 0 or 1	Direction select of the trackball axis	64
tbredcur	!tbredcur x 0 or 1	Current reduction for trackball mode Off or On	65
tbvel	!tbvel x 10	Specification of the max. speed	63
tboutpass	(?)!tboutpass x 100	The filter time constant for input values is 100µs	63
tb	!tb 0 or 1	Switch on trackball	66

Limit switch (Hardware a. Software)			
lim	(?) !lim x 0 100	Travel limits for the X-axis 0+100mm ?lim x	36
limctr	(?) !limctr x 1	Range monitoring for all axes X is active	36
nosetlimit	(?) !nosetlimit 1 1 1 1	No value range is set for all axes	37
swpol	(?) !swpol 1 0 1 (z 1 0 1)	Assign polarity of the limit switch for all axes or only for Z.	38
swact	(?) !swact 1 0 1 (z 1 0 1)	Switch ON/OFF limit switch for all axes or. only Z	38
readsw	?readsw	read the status all limit switches	39

Digital and analogue Input and Output			
digin	?digin or ?digin 8	read all inputs or input 8	67
digout	!digout 5 1/?digout	output 5 is set to 1 / read status of all outputs	67
anain	?anain c 2	read current status of the analogue channel 2	68
anaout	(?) !anaout c 1 0	set analogue cannel 1 to 0	68

Clock pulse Forward / Back			
tvrtoaxis	(?)!tvrtoaxis 1 2 0 0	Clock forward/ backward axis assignment	72
tvr	(?) !tvr 1 1	activates clock pulse F/B for X + Y	72
tvrf	(?) !Tvrf 1	Clock forward/backward factor 1 = 1 clock pulse is 1 motor increment	74
tvrm	(?) !tvrm 1	Clock pulse Forward/Back Mode 0 to 4	73

Command	Example	Note	Chap. 4 Page
Encoder-Settings			
twi	(?) !twi 10 10 10	The target window for all axes=10µ (for dimension=1)	83
poswindowrange	(?)!poswindowrange	Alternative to "twi" (for standardization)	84
enctype	(?)!enctyp 4 4 4 4	TTL encoder for all axes	76
enctoaxis	(?) !enctoaxis	Positioning of transmitter input and axis	75
encdir	(?)!encdir 1 1 1 1	Counting direction of the encoder reversed	76
enc	(?) enc	Answer: 1 0=Encoder-X = active Encoder-Y = deactivated	77
encperiod	(?) !encperiod 0.1	division period of the X-encoder = 0,1mm	77
encpolepairs	!encpolepairs x 500	Shows the amount of encoder signal periods per motor revolution.	78
encref	(?) !encref 0	no reference signal evaluation	78
encrefpol	(?) !encrefpol	Polarity of reference marks of QEP-Encoder	79
encpos	(?) !encpos 1	the encoder values are indicated by the inquery of the positions	79
encerr	(?) !encerr 0	clear encoder error messages X; (acknowledgement= 0 or. e)	80
posconkp	(?) !posconkp	Kp- band of the position controller	81
posconoutpass	(?) !posconoutpass	Filter time constant for the output filter (low pass) of the position controller	81
posconenable	(?) !posconenable	Axis enable for position controller	81
poscon	(?) !poscon	Switch-on and switch-off of the position controller	82
deviationsrange	(?) !deviationsrange	Deviation check / range	82
deviationtime	(?) !deviationtime	Deviation check / time frame	82
deviationcheck	(?) !deviationcheck	Switch deviation check On or Off	83

Trigger-Output			
trig	(?) !trig 1	switches the trigger on	85
triga	(?) !triga X	selects the axis, that should be triggered (i.e. X)	85
trigm	(?) !trigm 1	sets the trigger-mode to 1	85
trigs	(?) !trigs 4	sets the trigger-signal-length to 4µs	87
trigd	(?) !trigd 1	sets the trigger-distance to 1mm (for Dim 2)	87
Trigcount	(!)?trigcount	-reads counter setting Trigger 1	88

Command	Example	Note	Chap. 4 Page
Snapshot-Input			
sns	(?) !sns 1	Snapshot "ON"	88
snsl	(?) !snsl 1	snapshot is high-active	89
snsm	(?) !snsm 1	Auto-snapshot	90
snsc	?snsc	delivers the amount of the released snapshots	90
snsp	(!) ?snsp	delivers the saved position	91
snsa	?snsa 11	Inquiry of the snapshot-position 11	91
snsf	(?)!snsf 10	serves as input filter for all rebound switches (value 0-100)	89

Explanations	
!	write only ("!" can also be left out)
(?) !	write and read
?	read only ("?" can also be left out)

Possibilities For Input	
Command Value Value Value Value	all axes will be set or read
Command Value Value	only X + Y will be set or read
Command Axis Value	only the selected axis will be set or read

Error messages LSTEP-PCI *express*, LSTEP *express* and LSTEP-PP

Error Message	Error Description
1031	Axis not configurated
1032	Internal error
1033	Axis still in use
1034	Axis in error/fault state
1035	Axis not calibrated
1036	Axis without RoomMeasure
1037	Min. limit unknown
1038	Max. limit unknown
1039	Emergency Stop triggered
1040	Limit switch moved up
1041	Travel too small
1042	Speed too small
1043	Jerk too small
1044	No Trigger limit switch in
1045	No Trigger limit switch out
1046	Travel way clipped
1064	Distance too great
1065	Brake and power supply for limit switch not possible at the same time
1066	No commutation necessary

Error messages LSTEP-PCI *express*, LSTEP *express* and LSTEP-PP
Axis error

Error Message	Error Description
1096	Limit switch active
1097	Reference switch active
1098	Not ready for auto commutation
1099	No interpolative transmitter found
1100	I²T Monitoring addressed (long-term)
1101	I²T Monitoring addressed (short-term)
1102	Over-current power amplifier
1103	Over-current at power
1104	Over-current
1105	Fuse intermediate voltage defect
1106	Encoder error: Amplitude too small
1107	Encoder error: Amplitude too great
1108	Contouring error too great
1109	Speed too great
1110	Motor blocked
1111	Motor brake faulty
1112	Over-temperature of power amplifier
1113	Motor overheated
1114	Limit switch switched at auto commutation
1115	Read error temperature of power amplifier
1116	Target window not reached

Error messages LSTEP-PCI *express*, LSTEP *express* and LSTEP-PP
Monitoring

Error Message	Error Description
1117	Axis still moving
1118	Button activated for min. moving range
1119	Button activated for max. moving range
1120	Target position outside min. moving range
1121	Target position outside max. moving range
1122	Several limit switches activated at the same time
1123	Power amplifier deactivated through hardware monitoring
1124	Tracking error Encoder
1125	Amplitude of encoder too small, eventuallly no transmitter connected
1126	Angle at auto commutation too small, axis eventually blocked

Error messages LSTEP-PCI *express*, LSTEP *express* and LSTEP-PP
Axe messages

Error Message	Error Description
1192	#TRACKING_WARNING
1193	Warning over-temperature power amplifier
1194	Warning: Motor temperature too high
1195	Driver voltage below

4.2 Firmware and Hardware information

The Firmware version can be inquired with the "ver" instructions. Which options are released in the Firmware can be inquired with the "det"instruction. Each LStep has its own individual internal serial number. This serial number can be read out with the instruction "readsn".

read version number	
Command:	?ver or ver
Parameter:	none
Description:	gives the current Firmware version number
Feedback:	LS44.xx.xxx
Error code:	--
Example:	?ver

Read out Internal Version number	
Command:	?iver or iver
Parameter:	none
Description:	gives detailled information of the version number
Feedback:	weekday_calendar week_year-consecutive number
Error code:	--
Example:	?iver feedback for ex.: T04_35-02-0004

Read out version number in detail		
Command:		?det or det
Parameter:		none
Description:		gives the detailed Firmware version number-
Feedback:		A decimal value is given which has to be converted to a hexadecimal value:
		0x0 - - - 1 ➔ 1Vss encoder configured
		0x0 - - - 2 ➔ MR encoder configured
		0x0 - - - 4 ➔ TTL encoder configured
		0x0 - - - 3 ➔ The second number specifies the number of axes (here 3)
		0x0 - 1 - - ➔ Display configured
		0x0 - 2 - - ➔ Speed poti configured
		0x0 - 4 - - ➔ Handwheel (man. encoder) configured
		0x0 - 8 - - ➔ Snapshot configured
		0x01 - - - ➔ TVR configuredt
		0x02 - - - ➔ Triggerout configured
		0x08 - - - ➔ TVRout configured
		0x1 - - - - ➔ 16 digital I/O configured
		0x2 - - - - ➔ 32 digital I/O configured
		0x4 - - - - ➔ Trackball
		The appropriate combination of the information gives the present configuration.
Error code:		--
Example:		?det = 81697 ➔ $13F21_H$
Description 13F21	**1**	16 digital I/O configured
	3	TVR and Triggerout configured
	F	display; speedpoti; handwheel and snapshot configured
	2	2 axis
	1	1Vss encoder configured

Read Serial Number	
Command:	?readsn
Parameter:	none
Description:	Read out serial number of the controller.
Feedback:	9-characters
Error code:	--
Example:	?readsn

4.3 Reset

There are three ways to reset the control program:

- The hardware reset at the main power switch (for controllers without a display).
- The hardware reset with the Reset button (for controllers with display only).
- The hardware reset with the Dip-switch 1 for the PCI-card
- The software reset

Software - Reset	
Command:	Reset
Parameter:	none
Description:	The controller is reset to starting status
Feedback:	none
Error code:	--
Example:	Reset
Activation:	immediately

Power amplifier	
Command:	!poweramplifier oder !pa
Parameter:	0 or 1
Description:	0 = Power amplifier Off 1 = Power amplifier On
Feedback:	--
Error code:	--
Example:	!pa 1 1 1 1 All axes power amplifier On
Activation:	immediately

4.4 Interface Configuration

Baud - Rate	
Command:	!baud or ?baud
Parameter:	9600, 19200, 38400, 57600
Description:	!baud 19200 ➔ The transmission rate of the interface is set at 19200.
	?baud ➔ gives the present transmission rate
Feedback:	Present transmission rate
Error code:	--
Example:	?baud
Activation:	immediately

Parameter Save function	
Command:	Save
Parameter:	--
Note:	Save means: The current parameter (spindle pitch etc.) will be programmed in the Flash and immediately available at a restart.
Description:	save => The current parameter will be programmed into the Flash.
Feedback:	Notification in display With ?err success can be controlled. i.e.: Feedback = 0 => Save OK Feedback dissimilar 0 => Save not OK (see controller-manual)
Error code:	--
Example:	--

4.5 Instruction Set Used and save Functions

AutoStatus	
Command:	!autostatus or ?autostatus
Parameter:	0,1, 2, 3 or 4
Description:	0 ➔ No status is transmitted by the controller.
	1 ➔ "Position reached" signals are transmitted automatically by the controller.
	2 ➔ "Position reached" and status signals are transmitted automatically by the controller.
	3 ➔ For "Position reached" , only a carriage return is returned.
	4 ➔ Returns all write errors with parameters.
Feedback:	
Error code:	--
Example:	!autostatus 1 ?autostatus
Activation:	immediately

Status	
Command:	?status or status
Parameter:	--
Description:	gives the present status of the controller
Feedback:	OK... or ERR and error message
Error code:	--
Example:	?status

System status	
Command:	?sysstat or ?sysstatus
Parameter:	--
Description:	Provides the present system status of the axes
Feedback:	Number or text message
Error code:	--
Example:	?sysstatus

System status feedbacks		
sysstat	**sysstatus**	**Note**
0	Just turned On	
1	not Configured	
2	Configured	
3	Ready for Komm	
4	Autokommutating	
5	Ready for Power	Power amplifiers are switched off
6	Positioncontrol	
7	Speedcontrol	
8	Manual Mode	
9	Calibrating	
10	Error-Power-Off	Driver voltage interrupted
11	Error-Stop-Off	
12	Error-Stop-On	Controller off after deviation (lag error)
13	System Error	
14	Undef	
15	Changing	

StatusAxis	
Command:	?statusaxis or statusaxis
Parameter:	--
Description:	gives the present status of the individual axes.
Feedback:	e.g.: @ - M –
	@ ➔ Axis stands and is ready
	M ➔ Axis is moving (Motion)
	J ➔ Joystick mode
	C ➔ in control
	S ➔ Limit switch tripped
	A ➔ Acknowledgement after calibrating
	E ➔ Acknowledgement after calibrating if a fault occurs. (Limit switch was not set free correctly).
	D ➔ Acknowledgement after table stroke measuring
	U ➔ Setup mode (Setting **U**p)
	T ➔ Timeout
	- ➔ Axis is not enabled
Error code:	--
Example:	?statusaxis

Error	
Command:	?err or err
Parameter:	--
Description:	Error gives the present error number (see error description)
Feedback:	Decimal value
Error code:	--
Example:	?err

Error_Nr	Description of the error messages
0	no error
1	Valid axis designation missing
4	Invalid command
5	outside valid number area
6	Incorrect number of parameters
7	No ! Or ?
8	TVR not possible because axis is active
9	Axes cannot be switched on or off because TVR is active
10	Function not configured
11	no Move-command possible, because joystick-hand
12	Limit switch tripped
27	Emergency-STOP

4.6 Settings

The controller can be adapted to the mechanical components which are being used and to the desired requirements with the instructions described below.

ConfigMaxAxis	
Command:	(?)!configmaxaxis
Description of function:	*The number of axes of a controller can be configured with this command, e.g.: If you have a 4-axis controller but only have a 3-axis application, you can configure the controller as a 3-axis controller.*
Parameter:	1 through 4
Description:	?configmaxaxis => provides the number of configured axes !configmaxaxis 3 => the controller is configured as a 3-axis controller
Feedback:	1 through 4
Error code:	--
Example:	!configmaxaxis 4
Activation	!validconfig resp. !validpar

Dimension	
Command:	!dim or ?dim
Parameter:	X, y, z and a 0, 1, 2, 3 or 4 The units for specification of lengths for input and output are:
	0 ➔ Microsteps
	1 ➔ µm
	2 ➔ mm
	3 ➔ 360°
	4 ➔ Number of revolutions
	5 ➔ inch
Description:	
	!dim 4 1 ➔ The dimensions for the X- and Y-axes are "Number of revolutions" and "µm".
	!dim z 2 ➔ The dimension for the Z-axis is "mm".
	?dim ➔ All dimensions are displayed.
	?dim a ➔ The dimension of the A-axis is displayed.
Feedback:	Present setting
Error code:	
Example:	!dim 1 1 1 1 (all values in µm) ?dim
Activation:	immediately

Information:	The Spindle pitch should be set at 1 mm for dimensions 3 (degrees) and 4 (revolutions).

Spindle Pitch	
Command:	!pitch or ?pitch
Parameter:	X, y, z and a 0.0001 – 68
Description:	!pitch 4.0 1.0 ➔ Spindle pitches X = 4mm and Y = 1mm are programmed.
	!pitch z 1.0 ➔ Spindle pitch Z = 1mm is programmed.
	?pitch ➔ All spindle pitches are displayed.
	?pitch a ➔ The spindle pitch of the A-axis is displayed.
Feedback:	Present spindle pitch
Error code:	--
Example:	!pitch 10 (spindle pitch X = 10mm) ?pitch
Activation:	!validconfig resp. !validpar

Gear settings - geardenominator	
Command:	!geardenominator or ?geardenominator
Description of function:	*The command will be used at a gearing. With the command geardenominator the value Y will be adjusted in a break resp. in a gearing factor $^{X}/_{Y}$.*
Parameter:	X, y, z and a 1, 2, 3, ... ∞ (Natural numbers)
Description:	!geardenominator 4 1 ➔ Gearing-nominator X $^{X}/_{4}$ *and* $^{X}/_{1}$ *at Y* will be programmed.
	!geardenominator z 8 ➔ Gearing-*ratios* $^{X}/_{8}$ at z will be programmed.
	?geardenominator r ➔ All gearing-ratios will be displayed.
	?geardenominator a ➔ Current nominator of A-axis will be displayed.
Feedback:	The command ?geardenominator returns the current nominator (denominator) of the gearing-ratios.
Error code:	--
Example:	!geardenominator 2 (gearing-ratio $^{x}/_{2}$ at x) ?geardenominator
Activation:	!validconfig resp. !validpar

Gear Settings - gearnumerator		
Command:	!gearnumerator or ?gearnumerator	
Description of function:	*The command will be used at a gearing. With the command gearnumerator the value X will be adjusted in a break resp. in a gearing factor $^{X}/_{Y}$.*	
Parameter:	X, y, z and a 1, 2, 3, ... ∞ (Natural numbers)	
Description:	!geardenumeratorator 4 1	➔ Gear numerator X $^{4}/_{y}$ *and* $^{1}/_{y}$ *for Y* are programmed.
	!gearnumeratorator z 8	➔ Gearing-ratios $^{8}/_{y}$ at z will be programmed.
	?gearnumeratorator r	➔ All gearing-ratios will be displayed.
	?gearnumeratorator a	➔ The present numerator for the a-axis is displayed.
Feedback:	The command ?gearnumerator returns the current nominator (counter) of the gearing-ratios.	
Error code:	--	
Example:	!gearnumeratorator 2 (gearing-ratio $^{2}/_{Y}$ at x) ?gearnumeratorator	
Activation:	!validconfig resp. !validpar	

Acceleration		
Command:	!accel or ?accel	
Description of function:	*The acceleration of the individual axes can be configurated through this command.*	
Parameter:	X, y, z and a 0.01 – 20.00 [m/s²]	
Description:	!accel 1.00 1.50	➔ At axis X and Y the accelerations (x=1.00, y=1.50 [m/s²]) will be adjusted, the other axes remain unchanged.
	!accel x 1	➔ The acceleration for the X-axis is set to 1.00 [m/s²].
	?accel	➔ All preset accelerations are displayed.
	?accel z	➔ The adjusted acceleration of the Z-axis is displayed.
Feedback:	Adjusted acceleration	
Error code:	--	
Example:	!accel 1.00 (set accelerations for X-axis to 1 m/s²) ?accel	
Activation:	immediately	

Jerk adjustment during acceleration [j]	
Command:	!acceljerk oder ?acceljerk
Description of function:	*These profiles are to use at vibration critical machine mechanics. The ramp times for acceleration set-up and run-down should only be adjusted as big as necessary and as small as possible, since these parameters have enormous influence on the positioning times. This command enables the configuration of the jerk for each axis during acceleration.*
Parameter:	X, y, z and a 1, 2, 3, ... ∞ [m/s³] (Natural numbers)
Description:	!acceljerk 4 1 ➔ Jerk 4 m/s³ at X and 1 m/s³ will be programmed at Y
	!acceljerk z 10 ➔ Jerk 10 m/s³ will be programmed at Z.
	?acceljerk ➔ All configurated values are displayed.
	?acceljerk a ➔ Jerk adjustment of A-axis is displayed.
Feedback:	Jerk acceleration
Error code:	--
Example:	!acceljerk 10 (A jerk of 10m/s³ will be adjusted at x) ?acceljerk
Activation:	immediately

Delay		
Command:	!decel or ?decel	
Description of function:	*The delay of the individual axes can be configurated through this command. With this the value to slow down the axes, can be adjusted.*	
Parameter:	X, y, z and a 0.01 – 20.00 [m/s²]	
Description:	!decel 1.00 1.50	➔ For axis x and y the delays (x=1.00, y=1.50 [m/s²]) are adjusted, the other axes remain unchanged
	!decel x 1	➔ The delay for X-axis will be adjusted to 1.00 [m/s²]
	?decel	➔ All adjusted delays are displayed.
	?decel z	➔ The adjusted delay of the Z-axis will be displayed.
Feedback:	Delay	
Error code:	--	
Example:	!decel 10 (Delay in the axis at 10m/s²) ?decel	
Activation:	immediately	

Jerk adjustment during delay			
Command:	!deceljerk oder ? deceljerk		
Description of function:	*These profiles are to use at vibration critical machine mechanics. The ramp times for acceleration set-up and run-down should only be adjusted as big as necessary and as small as possible, since these parameters have enormous influence on the positioning times. This command enables the configuration of the jerk for each axis during acceleration.*		
Parameter:	X, y, z and a 1, 2, 3, ... ∞ [m/s³] (Natural numbers)		
Description:	!deceljerk 4 1	➔	Jerk 4 m/s³ at X and 1 m/s³ will be programmed at Y
	!deceljerk z 10	➔	Jerk 10 m/s³ will be programmed at Z.
	?deceljerk	➔	All configurated values are displayed.
	?deceljerk a	➔	Jerk adjustment of A-axis is displayed.
Feedback:	Jerk delay		
Error code:	--		
Example:	!deceljerk 10 (A jerk of 10m/s³ will be adjusted at x) ?deceljerk		
Activation:	immediately		

Speed (Velocity)		
Command:	!vel or ?vel	
Description of function:	*The speeds of travel of the individual axes are set with this command.*	
Parameter:	X, y, z and a 0 – maximum speed	
Description:	!vel 1.0 15	➔ At X and Y-axis the speed values (x=1.0, y=15 [R/s]) are described, the other axes remain unchanged.
	!vel z 0.1	➔ At the Z-axis the speed will be adjusted to 0.1 [R/s].
	?vel	➔ All adjusted speeds are displayed.
	?vel x	➔ Display of the adjusted speed of axis x.
Feedback:	Adjusted speed	
Error code:	--	
Example:	!vel 10 (The X-axis is run at max. 10 r/s) ?vel	
Activation:	immediately	

The speed of rotation of the motors can be set in steps (St) of 0.0001 r/sec. to 70 r/sec. The top speed ranges can be reached only if the motors and the mechanical components are optimally synchronized on the LSTEPexpress.
The lowest speed is possible with the dimension microsteps and the max. step resolution of 32768 MI/pole pair. The following lowest speeds apply for the various different dimensions: 1MI/s; 0.1µm/s; 0.0001mm/s; 0.001°/s; 0.0001R/s.

MaxCurrent (max. possible motor current)	
Command:	?maxcur
Description of function:	*The maximum motor current per motor phase can be enquired with this command.*
Parameter:	X, y, z or a
Description:	?maxcur y =>gives the maximum possible motor current for the Y-axis ?maxcur => gives the maximum possible motor current for all axes
Feedback:	Motor current in amps
Error code:	--
Example:	?maxcur

Adjust Motor Current For Each Axis			
Command:	!motorcurrent or ?motorcurrent		
Description of function:	*This command enables the adjustment independent of the axes of each motor current.*		
Parameter:	X, y, z and a 0.5 – 5 (fourth axis maximal 10A)		
Description:	!motorcurrent 1 2 2.5	➔	Sets the following values in the control: X=1A, Y=2A, Z=2.5A
	!motorcurrent z 5	➔	The Z-axis has a max. motor current of 5A.
	?motorcurrent	➔	All adjustments will be displayed.
	?motorcurrent a	➔	The adjusted value of the A-axis will be displayed.
Feedback:	Value indicated the max. motor current.		
Error code:	--		
Example:	!motorcurrent1 2 2.5 (Adjustment motor current: X=1A, Y=2A, Z=2.5A)		
Activation:	!validconfig resp. !validpar		

Current Reduction	
Command:	!reduction or ?reduction
Parameter:	X, y, z and a 0 – 100 %
Description:	In quiescent state, the rated motor current is reduced to the parameterized ratio.
	!reduction 10 70 ➔ X-axis = 0.1*rated current and Y-axis = 0.7*rated current
	!reduction z 50 ➔ Z-axis = 0.5*rated current
	?reduction ➔ Display of the preset current reductions of all axes
	?reduction x ➔ Display of the preset current reduction for the X-axis.
Feedback:	Preset current reduction
Error code:	--
Example:	!reduction 30 50 (X- and Y-axes are reduced) ?reduction
Activation:	immediately

Delay Current Reduction (Delay Reduction)	
Command:	!curdelay or ?curdelay
Parameter:	X, y, z and a 0 – 10000 (ms)
Description:	After moving a vector the motor current is maintained for the time that is set in curdelay. Afterwards it is reduced to the specified value of the current reduction. !curdelay 100 300 = x-axis = 100 ms delay and y-axis = 300 ms delay !curdelay z 450 = z-axis = 450 ms delay ?curdelay = indication of the set current reduction of all axes ?curdelay x = indication of the set current reduction of the x-axes
Feedback:	set delay of the current reduction
Error code:	--
Example:	!curdelay 100 300 (x- and y-axis are delayed) ?curdelay
Activation:	immediately

Enable axes	
Command:	!axis or ?axis
Parameter:	x, y, z and a 0 and 1
Description:	!axis 1 0 1 0 ➔ The X- and Z-axes are enabled; the Y- and A- axes are not enabled.
	!axis y 1 ➔ Y-axis enabled
	?axis ➔ Show status of all axes
	?axis a ➔ Show status of axis A
Feedback:	Present operating status
Error code:	--
Example:	!axis 1 1 1 1 (enable all axes) ?axis x (read status of X-axis)
Activation:	immediately

Axis Direction	
Command:	!?axisdir
Description of function:	*The direction of rotation and the limit switch can be changed with this command.*
Parameter:	X, y, z and a 0 or 1
Note:	With axisdir the motor – turning direction can be turned, the corresponding limit switches are turned also.
Description:	!axis 0 1 0 1 => the turning direction of the axis y and a are turned. ?axisdir x = indication, if the turning direction of the x-axis is activated.
Feedback:	0 = no change turning direction 1 = Change of rotation direction
Error code:	--
Example:	!axisdir 0 0 0 0 (Cancel all changes of turning direction)
Activation:	!validconfig resp. !validpar

Adjust Motor Rotation Direction		
Command:	!motorfielddir or ?motorfielddir	
Description of function:	*The direction of rotation of the motor can be influenced with this command. The individual axes can have different motor rotation directions.*	
Parameter:	X, y, z and a Values 0 and 1	
Description:	!motorfielddir 0 1	➔ Rotation direction of the motor is exchanged in Y, X is normal
	!motorfielddir Z 1	➔ Rotation direction of the motor is exchanged in Z
	?motorfielddir	➔ All adjusted rotation directions will be displayed.
	?motorfielddir x	➔ Rotation direction of X-axis will be displayed.
Feedback:	Current motor rotation direction of the individual axes 0 = no change turning direction 1 = Change of rotation direction	
Error code:	--	
Example:	!motorfielddir 0 1 (sets the corresponding values and executes a reversion of rotation direction in Y. X remains unchanged)	
Activation:	!validconfig resp. !validpar	

Change limit switch	
Command:	!swchange or ? swchange
Description of function:	*With this command the limit switches can be changed with the software. A mechanical effort is unnecessary with this command.*
Parameter:	X, y, z and a Values 0 and 1
Description:	!swchange 0 1 ➔ Limit switch is exchanged in Y, X is normal
	!swchange Z 1 ➔ Limit switch is exchanged in Z
	?swchange ➔ All adjustments of the limit switches are displayed.
	?swchange x ➔ Adjustment of the limit switch for X-axis is displayed.
Feedback:	Current adjustment of limit switch for each axis 0 = No exchange limit switch 1 = Exchange limit switches
Error code:	--
Example:	!swchange 0 1 (sets the corresponding values and executes an exchange in Y. X remains unchanged)
Activation:	!validconfig resp. !validpar

Set number of motor pole pairs	
Command:	!?motorpolepairs
Description of function:	*Stepper motors with different step angles can be adapted to the controller with this command.*
Parameter:	x, y, z and a Number of pole pairs
Note:	A 200-step motor with a step angle of 1.8° has 50 pole pairs.
Description:	!motorpolepairs x 50 (The X-motor has 50 pole pairs = 1.8°/full step) ?motorpolepairs x = Display of the number of pole pairs set for the X-motor
Feedback:	Number of pole pairs
Error code:	--
Example:	!motorpolepairs 50 50 100 100
Activation:	!validconfig resp. !validpar

Number of microsteps per motor pole pair	
Command:	!?motorpolepairres
Description of function:	*The step resolution can be set with this command.*
Parameter:	x, y, z and a Step resolution: 500; 1000; 2000; 32768
Note:	For a 200-step motor, a step resolution of 1000 results in a resolution of 50,000 microsteps / revolution
Description:	!motorpolepairres x 32768 (The X-motor has a resolution of 1,638,400 microsteps for a 1.8° motor) ?motorpolepairres x = Display of the set step resolution
Feedback:	Set step resolution
Error code:	--
Example:	!motorpolepairres 1000 1000 1000 1000
Activation:	!validconfig resp. !validpar

Maximum motor drive	
Command:	!?motormaxvel
Description of function:	*The maximum speed of rotation of the motor can be limited with this command*
Parameter:	x, y, z and a Speed of rotation 1/min.
Note:	For a 200-step motor, a step resolution of 1000 results in a resolution of 50,000 microsteps / revolution
Description:	!motormaxvel x 1000 (For the X-motor, 50 has a max. speed of 1000 revs./min) ?motormaxvel x = Display of the set velocity
Feedback:	Adjusted speed
Error code:	--
Example:	!motormaxvel 1000 1000 1000 1000
Activation:	!validconfig resp. !validpar

Motortype	
Command:	!?motortype
Description of function:	*The type of motor can be set with this command.*
Parameter:	x, y, z and a Motor type 0 = rotary 2-phase stepper motor Motor type 1 = linear 2-phase stepper motor Motor type 4 = rotary 2-phase servomotor Motor type 5 = linear 2-phase servomotor
Note:	For a 200-step motor, a step resolution of 1000 results in a resolution of 50,000 microsteps / revolution
Description:	!motortype x 0 (a 2-phase rotary stepper motor is connected) ?motortype x Display of the set motor type
Feedback:	Preset motortype
Error code:	--
Example:	!motortype 0 0 0 0 (all 4 axes have 2-phase rotary stepper motor)
Activation:	!validconfig resp. !validpar

Motorbrake	
Command:	!?motorbrake
Description of function:	*With this command, an output can be switched for each axis, which can be used for a motor brake or for other applications, such as, e.g. a door interlock.*
Parameter:	x, y, z and a The output switches: Motorbrake 0 = to GND Motorbrake 1 = dependent on the power amplifier (pa 0 or pa 1) Motorbrake 2 = to +12V (24V)
Note:	For a 200-step motor, a step resolution of 1000 results in a resolution of 50,000 microsteps / revolution
Description:	!motorbrake x 0 (the output is switched to GND) ?motorbrake x Display of the setting
Feedback:	Preset mode
Error code:	--
Example:	!motorbrake 1 1 1 1 (for all four axes, the output is switched dependent on the power amplifier)
Activation:	!validconfig resp. !validpar

Motorbrakeswitchondelay	
Command:	!?motorbrakeswitchondelay
Description of function:	*With this command, a delay can be set in "motorbrake1" mode to delay switch-on of the output or of the power amplifier.*
Parameter:	x, y, z and a Value range from -5000 to 5000 ms
Note:	Negative values delay the switch-on of the power amplifier. Positive values delay the switching of the output
Description:	!motorbrakeswitchondelay z 1000 (e.g.: when the power amplifier for the Z-axis is switched on, the brake is only released after a delay of 1 second) ?motorbrakeswitchondelay z (Query the delay for Z)
Feedback:	Preset delay
Error code:	--
Example:	!motorbrakeswitchondelay x -1000 (e.g.: when the power amplifier for the X-axis is switched on, the door interlock is activated 1 second beforehand)
Activation:	!validconfig resp. !validpar

Motorbrakeswitchoffdelay	
Command:	!?motorbrakeswitchoffdelay
Description of function:	*With this command, a delay can be set in "motorbrake1" mode to delay switch-off of the output or of the power amplifier.*
Parameter:	x, y, z and a Value range from -5000 to 5000 ms
Note:	Negative values delay the switch-off of the power amplifier. Positive values delay the switching of the output
Description:	!motorbrakeswitchoffdelay z -1000 (e.g.: when the power amplfier for the Z-axis is switched off, the brake is activated 1 second beforehand) ?motorbrakeswitchoffdelay z (Query the delay for Z)
Feedback:	Preset delay
Error code:	--
Example:	!motorbrakeswitchoffdelay x 1000 (e.g.: when the power amplifier for the X-axis is switched off, the door interlock is deactivated 1 second beforehand)
Activation:	!validconfig resp. !validpar

Confirm error rectification	
Command:	quit
Parameter:	none
Description:	After an error has occurred and has been rectified, confirmation must be given with "!quit"
Feedback:	none
Error code:	--
Example:	!quit

Take over the settings	
Command:	validconfig
Parameter:	none
Description:	After changes are made to basic settings, such as e.g. the spindle pitch and gear factor numerator and gear factor denominator, these are applied with this command validconfig.
Feedback:	none
Error code:	--
Example:	!validconfig

Activation: of the controller parameters	
Command:	validpar
Parameter:	none
Description:	After changes are made to controller parameters, such as, e.g. "posconkp", these are activated with the command validpar.
Feedback:	none
Error code:	--
Example:	!validpar

Limit	
Command:	!lim or ?lim
Description of function:	*The travel limits of the individual axes can be set in the software with this command. The travel range can thus be adjusted to meet the requirements.*
Parameter:	x, y, z or a +- maximale positioning capacity
Note:	The values must be entered in pairs for each axis along with axis identification. The in- and output values are depended on the dimension.
Description:	!lim x -1000 1000 ➔ Moving range limits are assigned to x-axis
	?lim z ➔ Read travel range limits of the Z- axes
Feedback:	Current moving range
Error code:	--
Example:	!lim y 0 10 ?lim x
Activation:	immediately

Range monitoring	
Command:	!limctr or ?limctr
Parameter:	x, y, z or a 0 or 1
Description:	!limctr 1 1 1 ➔ Range monitoring active for the X-, Y- and Z-axes.
	!limctr z 1 ➔ Range monitoring active for the Z-axis.
	?limctr a ➔ Read range monitoring axis a
	?limctr Display of the status of the individual range monitoring.
Feedback:	0 = Range monitoring not active 1 = Range monitoring active
Error code:	--
Example:	! limctr y 0 (Deactivate Y-axis range monitoring) ? limctr
Activation:	immediately

statuslimit	
Command:	?statuslimit or statuslimit
Parameter:	--
Description:	Statuslimit delivers the current condition of the software-limits of each single axis.
Feedback:	A = Axis was calibrated
	D = table stroke was measured
	L = Software – Limit was set
	- = Software – Limit was not changed
	The sequence of the acknowledgement is for example: AA-A--DD-LL-L--L
	X,y and a = calibrated
	Z and a = measure table stroke Y and z = min. software limit set
	X and a = max. software limit set
Error code:	--
Example:	?statuslimit

NoSetLimit	
Command:	!?nosetlimit
Parameter:	x, y, z or a 0 or 1
Note:	When calibrating and measuring table stroke normally the internal software – limits are set, this can be avoided herewith.
Description:	nosetlimit 1 1 1 => No moving range limits are set for the axis x, y and z. !nosetlimit y 1 => No moving range limit is set for the y-axis. ?nosetlimit = Read settings of all axes ?nosetlimit a = Read settings of axis a
Feedback:	0 = Software – Limits will be set (calib/rm)
Error code:	--
Example:	?nosetlimit
Activation:	immediately

Limit Switch Polarity	
Command:	!swpol or ?swpol
Parameter:	x, y, z or a 0  or 1 
Description:	!swpol x 1 0 1 ➔ Assign polarity of the limit switch of the X-axis. (Order: E0 REF EE).
	?swpol z ➔ Read out polarity of limit switch of axis z.
Feedback:	Polarity of the limit switches
Error code:	--
Example:	!swpol y 1 1 1 (All Y-axis switches react to the positive edge) ?swpol x
Activation:	!validconfig resp. !validpar

Limit Switch On/Off	
Command:	!swact or ?swact
Parameter:	x, y, z or a 0 or 1
Description:	!swact x 1 0 1 ➔ X-axis limit switch : E0=On REF=Off EE=On
	?swact z ➔ Read out state of limit switch of axis z.
Feedback:	Status of the limit switches
Error code:	--
Example:	!swact y 1 1 1 (All Y-axis switches active) ?swact x
Activation:	!validconfig resp. !validpar

Read Limit Switches												
Command:	?readsw											
Parameter:												
Description:	?readsw ➔ Read status of all limit switches.											
Feedback:	Status of the limit switches.											
	Axis:	**x**	**y**	**z**	**a**	**x**	**y**	**z**	**a**	**x**	**y**	**z**	**a**
	Switch:	**E0**	**E0**	**E0**	**E0**	**R**	**R**	**R**	**R**	**EE**	**EE**	**EE**	**EE**
	E0 = Zero limit switch				R = Reference limit switch				EE = End limit switch			
Error code:	--											
Example:	?readsw (Read all limit switches)											

Set stop input polarity.	
Command:	!stoppol or ?stoppol
Parameter:	0 = low active 1 = high active
Description:	Because the stop entrance has a Pull Up after 5V, a closer has to be set low active and an opener high active.
Feedback:	--
Error code:	--
Example:	!stoppol 1 (the stop input is high active)
Activation:	immediately

Stop Input Adjust Brake Acceleration			
Command:	!stopdecel oder ?stopdecel		
Description of function:	*The delay of the individual axes can be configurated through this command. With this the value can be adjusted, when the axes should break in case of a stop signal.*		
Parameter:	X, y, z and a 0.01 – 20.00 [m/s²]		
Description:	!stopdecel 1.00 1.50	➔	For axis x and y the delays (x=1.00, y=1.50 [m/s²]) are adjusted, the other axes remain unchanged
	!stopdecel x 1	➔	The delay for X-axis will be adjusted to 1.00 [m/s²]
	?stopdecel	➔	All adjusted delays are displayed.
	?stopdecel z	➔	The adjusted delay of the Z-axis will be displayed.
Feedback:	Adjusted value for delay at activated Stop input		
Error code:	--		
Example:	!stopdecel 10 (Delay in the axis at 10m/s²) ?decel		
Activation:	immediately		

Stop Input Jerk Limitation			
Command:	!stopdeceljerk or ? stopdeceljerk		
Description of function:	*These profiles are to use at vibration critical machine mechanics. The ramp times for acceleration set-up and run-down should only be adjusted as big as necessary and as small as possible, since these parameters have enormous influence on the positioning times. This command enables the configuration of the jerk for each axis during acceleration*		
Parameter:	X, y, z and a 1, 2, 3, ... ∞ [m/s³] (Natural numbers)		
Description:	!stopdeceljerk 4 1	➔	Jerk 4 m/s³ at X and 1 m/s³ will be programmed at Y
	!stopdeceljerk z 10	➔	Jerk 10 m/s³ will be programmed at Z.
	?stopdeceljerk	➔	All configurated values are displayed.
	?stopdeceljerk a	➔	Jerk adjustment of A-axis is displayed.
Feedback:	Jerk adjustment during delay of the system in case of Emergency Stop signal.		
Error code:	--		
Example:	!stopdeceljerk 10 (A jerk of 10m/s³ will be adjusted at X) ?stopdeceljerk		
Activation:	immediately		

4.7 Determination Of The Machanical Work Range

After initializing the controller, the instructions calibrate “cal“ and measure stroke “rm“should be performed. This will determine the maximum mechanical work range. This ensures that the axes cannot be moved into the limit switches.
The work stroke can only be measured when all axes have a zero and end limit switch.
So that the limit switches respond when the zero or the end position is reached if the mechanical components overshoot them, the work range can be limited with the instructions “caliboffset“ and “rmoffset“.

Calibrate	
Command:	!cal or cal
Parameter:	X, y, z or a
Description:	Cal ➔ Moves all enabled axes towards lower position values. Travel is stopped as soon as the limit switches have been tripped and is then resumed slowly in the opposite direction until the switch is no longer active. The position values are set to 0. The position is taken over as a software limit, as described in the instruction “Limit“.
	Cal y ➔ As above, however Y-axis only.
Feedback:	An ‘A’ for each calibrated axis or ‘E’ if fault occurs
Error code:	--
Example:	!cal

Measure Table Stroke	
Command:	!rm or rm
Parameter:	X, y, z or a
Description:	Rm ➔ Moves all enabled axes towards greater position values. Travel is stopped as soon as the limit switches have been tripped and is then resumed slowly in the opposite direction until the switch is no longer active. The position value is saved and is taken over as the software limit, as described in the instruction “Limit“.
	Rm z ➔ As above, however for the Z-axis only
Feedback:	A ‚D‘ for every axis
Error code:	--
Example:	!rm

RM Offset	
Command:	!rmoffset or ?rmoffset
Parameter:	X, y, z or a 0 – 32*50000 (32*spindle pitch)
Description:	!rmoffset 1 1 1 ➔ The X-, Y-, and Z-axes are each moved 1mm (for Dim. 2 2 2) away from the limit switch towards the center of the table when the table stroke is measured and the software limit is then set.
	?rmoffset y ➔ Read present offset of the Y-axis.
Feedback:	Distance
Error code:	--
Example:	?rmoffset
Activation:	immediately

Calibration Offset	
Command:	!caliboffset or ?caliboffset
Parameter:	X, y, z or a 0 – 32*50000 (32*spindle pitch)
Description:	!caliboffset 1 1 1 ➔ The X-, Y-, and Z-axes are each moved 1 mm (for Dim 2 2 2) away from the zero limit switch towards the center of the table when calibration is done and the zero position is then set (software limit).
	?caliboffset y ➔ Read present offset of the Y-axis
Feedback:	Distance
Error code:	--
Example:	?caliboffset
Activation:	immediately

Calibration Direction	
Command:	!?caldir
Description of function:	*With this command the direction of calibration can e adjusted.*
Parameter:	X, y, z or a 0 or 1
Note:	When calibrating in positive direction, the positive software limit is set.
Description:	!caldir 0 0 1 => The axes X, Y are calibrated in negative direction and the Z-axis in positive direction. ?caldir => read current direction for calibrating.
Feedback:	0 = negative direction 1 = positive direction
Error code:	--
Example:	!caldir y 1 (The Y-axis will be calibrated in positive direction)
Activation:	immediately

Speed (Velocity) when moving out of the limit switch			
Command:	!calibrmbspeed or ?calibrmbspeed		
Description of function:	*After the axis has reached the limit switches after calibration, the axis moves again out of the limit switch with the speed adjusted through command.* ***Information:*** *The present command calbspeed will be replaced by the new command* calibrmbspeed.		
Parameter:	X, y, z and a 0 – maximum speed		
Description:	!calibrmbspeed 1.0 15	➔	At X and Y-axis the speed values (x=1.0, y=15 [R/s]) are described, the other axes remain unchanged.
	!calibrmbspeed z 0.1	➔	At the Z-axis the speed will be adjusted to 0.1 [R/s].
	?calibrmbspeed	➔	All adjusted speeds are displayed.
	?calibrmbspeed x	➔	Display of the adjusted speed of axis x.
Feedback:	Adjusted speed		
Error code:	--		
Example:	!calibrmbspeed 10 (The x-axis will be operated with maximal 10 R/s when moving out of the limit switch). ?calibrmbspeed		
Activation:	immediately		

Acceleration During Calibration			
Command:	!calibrmaccel or ? calibrmaccel		
Description of function:	*This command enables the configuration of the acceleration during calibration.*		
Parameter:	X, y, z and a 0.01 – 20.00 [m/s²]		
Description:	!calibrmaccel 1.00 1.50	➔	At axis X and Y the accelerations (x=1.00, y=1.50 [m/s²]) will be adjusted, the other axes remain unchanged.
	!calibrmaccel x 1	➔	The acceleration for the X-axis is set to 1.00 [m/s²].
	?calibrmaccel	➔	Display of al adjusted accelerations
	?calibrmaccel z	➔	The adjusted acceleration of the Z-axis is displayed.
Feedback:	Adjusted acceleration		
Error code:	--		
Example:	!calibrmaccel 1.00 (ets accelerations at X-axis to 1 m/s²) ?accel		
Activation:	immediately		

Adjustment Jerk During Calibration	
Command:	!calibrmjerk or ? calibrmjerk
Description of function:	*These profiles are to use at vibration critical machine mechanics. The ramp times for acceleration set-up and run-down should only be adjusted as big as necessary and as small as possible, since these parameters have enormous influence on the positioning times. This command enables the configuration of the jerk for each axis during acceleration.*
Parameter:	X, y, z and a 1, 2, 3, ... ∞ [m/s³] (Natural numbers)
Description:	!calibrmjerk 4 1 ➔ Jerk 4 m/s³ at X and 1 m/s³ will be programmed at Y
	!calibrmjerk z 10 ➔ Jerk 10 m/s³ will be programmed at Z.
	?calibrmjerk ➔ All configurated values are displayed.
	?calibrmjerk a ➔ Jerk adjustment of A-axis is displayed.
Feedback:	Jerk adjustment during calibration
Error code:	--
Example:	!calibrmjerk 10 (A jerk of 10m/s³ will be adjusted at X) ?calibrmjerk
Activation:	immediately

Speed During Calibration			
Command:	!calibrmvel or ?calibrmvel		
Description of function:	*This command enables the configuration of the speed during calibration.*		
Parameter:	X, y, z and a 0 – maximum speed		
Description:	!calibrmvel 1.0 15	➔	At X and Y-axis the speed values (x=1.0, y=15 [R/s]) are described, the other axes remain unchanged.
	!calibrmvel z 0.1	➔	At the Z-axis the speed will be adjusted to 0.1 [R/s].
	?calibrmvel	➔	All adjusted speeds are displayed.
	?calibrmvel x	➔	Display of the adjusted speed of axis x.
Feedback:	Adjusted speed		
Error code:	--		
Example:	!calibrmvel 10 (The X-axis operates with max. 10 r/s) ?calibrmvel		
Activation:	immediately		

4.8 Travel instructions and their control functions

For all positioning commands, linear interpolation takes place for all axes started simultaneously with a command, i.e. all axes reach the specified position at the same time. The axis where the motor must traverse the most revolutions is deemed the lead axis and thus travels at the preset speed and acceleration. The x-axis is the lead axis, if all axes haves to travel the same stretch. The set speed and acceleration of the other axes are, however, monitored and are not exceeded.
If the axes have totally different dynamic behaviour, they could be started individually. Also an assynchrones movement is possible. Herewith is to be noted that in the setting auto status 1 the acknowledgement first comes, if all axes came to standstill.
If you want to start another axis while one is already moving the auto status = 0 and poll with ?statusaxis.

Position absolut	
Command:	!moa or moa
Parameter:	X, y, z or a +- Moving range
Note:	The input depends on the dimension.
Description:	Moa 10 0 20 ➔ The X-, Y- and Z-axes are positioned at the position values which were input.
	moa y 333 ➔ As above, however Y-axis only.
Feedback:	For each positioned axis a ‚@'
Error code:	--
Example:	Moa x 10 (The X-axis is positioned at the position which was input)

Relative Position	
Command:	!mor or mor
Parameter:	X, y, z or a +- Moving range
Note:	The input depends on the dimension.
Description:	Mor 100 0 39 ➔ The X- and Z-axes are travelled the distances which were input.
	Mor a 298 ➔ The A-axis is travelled the distance which was input.
Feedback:	A,@' for every axis which is travelled
Error code:	--
Example:	!mor 0 0 0 100 (Only the A-axis is travelled)

Position Relative (short command)	
Command:	!m or m
Parameter:	
Note:	This instruction is used when the same distance is to be travelled again and again at short intervals. The distance to be travelled must first be set with !distance or more instructions. The position is not updated, until after the next Move-command.
Description:	m ➔ Start travel of all enabled axes.
Feedback:	Depends on the autostatus setting.
Error code:	--
Example:	!mor 0 0 0 100 (Only the A-axis is travelled) m (A-axis is travelled by 100 again)

Distance		
Command:	!distance or ?distance	
Parameter:	X,Y,Z and A Min./max. range of travel	
Note:	Input and output depend on the dimension.	
Description:	!distance 1 2 3	➔ The distances for the X-, Y-, and Z-axes are set.
	!distance y 20	➔ The Y-axis distance is set.
	?distance	➔ Inquire present distances for all axes.
	?distance z	➔ Inquire present distance for Z-axis.
Feedback:	Distances	
Error code:	--	
Example:	!distance 10 20 (Set X- and Y-axis distances) ?distance (Inquire distances of all axes)	
Activation:	immediately	

SpeedPoti	
Command:	!pot or ?pot
Parameter:	0 or 1
Note:	
Description:	0 ➔ Travelling is done at the preset speed (vel).
	1 ➔ Travelling is done at a percentage of the preset speed (vel), depending on the setting of the potentiometer.
Feedback:	--
Error code:	--
Example:	!pot 1 ?pot
Activation:	immediately

Position		
Command:	!pos or ?pos	
Parameter:	X,Y,Z and A Min./max. range of travel	
Note:	Input and output depend on the dimension.	
Description:	!pos 1000 2000 3000	➔ The position values for the X-, Y-. and Z-axes are set.
	!pos y 2000	➔ The position of the Y-axis is set.
	?pos	➔ Inquire present position of all axes.
	?pos z	➔ Inquire present position of Z-axis.
Feedback:	Position values	
Error code:	--	
Example:	!pos100 200 (Set positions of X- and Y-axes) ?pos (Inquire positions of all axes)	
Activation:	immediately	

Abort	
Command:	!a or a
Parameter:	none
Description:	All travels are stopped.
Feedback:	A @ for every axis
Error code:	--
Example:	!a

Set table position		
Command:	(?)!tpos (Set Table Pos)	
Parameter:	1 – 1,000 Positions for x,y,z,a within the travel range limits	
Note:	Up to 1,000 positions can be stored in the table. The input depends on the dimension.	
Description:	!tpos x 5 10	➔ Position 10 is stored as the table position 5 of the X-axis
	!tpos 5 10 10 10 10	➔ As above, however for all axes
	?tpos 5	➔ Query the table position 5 for all axes
Feedback:	--	
Error code:	--	
Example:	Moa x 10 (The X-axis is positioned at the position which was input)	

Move to table position	
Command:	!mtpa (Move Table Pos Absolut)
Parameter:	X, y, z or a 1 – 1.000
Note:	
Description:	!mtpa 5 5 5 5 ➔ Moves all axes to the positions stored at position 5 of the table
	!mtpa y 5 ➔ As above, however Y-axis only.
Feedback:	For each positioned axis a ‚@'
Error code:	--
Example:	!mtpa x 10 (The x-axis is positioned at position 10 of the table)

Index Table Divider	
Command:	(?)!itd (Index Table Driver)
Parameter:	X, y, z or a 1 – 1,000,000
Note:	A rotary axis, for example, can be divided up into a number of equal sized angles with this command. An application example is a tool changer which has n-tools arranged equidistantly on a rotary disc.
Description:	!itd a 50 ➔ For a rotary unit that is connected at the a-axis, one revolution is divided up into 50 sections, equating to 7.2° per section.
	?itd a ➔ Readout of the set index of the a-axis
Feedback:	--
Error code:	--
Example:	!itd x 360 (the index is set to 360 for the x-axis)

Move Index Table	
Command:	!mita (Move Index Table)
Parameter:	X, y, z or a 1 – 1,000,000
Note:	
Description:	!mita a 10 ➔ For a setting of itd=50, the a-axis is positioned at 72°
	!mita a 1 ➔ As above, however at 7.2°
Feedback:	For each positioned axis a ‚@'
Error code:	--
Example:	!mita x 10 (The x-axis is positioned at index 10)

4.9 Joystick, Jog Mode and Trackball Commands

Information:	1. Behaviour similar to a joystick can be achieved per software with the "speed" command. 2. The joystick can be switched on by means of a command and move commands are possible even with the joystick is active. 3. If the Save command is sent while the joystick is switched on, the joystick is already active after the controller or the PC is switched on. 4. The inputs for jog mode, trackball and encoder inputs Quep1 are mutually exclusive; only one function is possible.

Digital Joystick (Speed)		
Command:	!speed or ?speed	
Parameter:	x, y, z or a +- Maximum speed (vel)	
Description:	With this command, single axes can be operated with a constant speed.	
	!speed 0	➔ All axes at speed 0 and joystick mode "OFF"
	!speed 10	➔ All axes at speed 10.0 [r/s] and joystick mode "ON".
	!speed 10 10 0 10	➔ X-, Y-, and A-axes at speed 10.0 [r/s], Z-axis speed 0 and joystick mode "ON".
	!speed y 25	➔ Y-axis speed 25 and joystick mode "ON".
	!speed y -25	➔ Axis y speed 25 in negative direction joystick-operation „ON„.
	?speed	➔ Read the preset speeds.
	?speed y	➔ Read the preset speed of the Y-axis.
Feedback:	Present speeds	
Error code:	--	
Example:	!speed 33 11 (X-axis speed 33.0 [r/s], Y-axis speed 11.0 [r/s] and joystick mode "ON") ?speed	
Note:	In order to position absolute or relative after carrying out the speed command, the digital joystick must be switched off with !speed 0 and the speed needs to be set new.	

Joystick direction	
Command:	!joydir or ?joydir
Parameter:	x y z a 0 or 1
Description:	With the input of joydir, the direction of rotation of the motors is reversed when the joystick is deflected. !joydir x 1 (The direction of rotation of the X-axis was reversed.)
Feedback:	Set joystick directions.
Error code:	--
Example:	!joydir 0 1 0 0 ?joydir
Activation:	!joy 1

Joystick Speed	
Command:	!joyvel or ? joyvel
Description of function:	*With this command the max. positioning speeds in Joystick operation can be adjusted.*
Parameter:	x, y, z, a 0 – 70 revs./sec.
Description	!joyvel x 40 (when the joystick is fully deflected, the X-axis travels at 40 revs./sec.)
Feedback:	Adjusted speed
Error code:	--
Example:	!joyvel 10 10 10 10 (all axes are operated at max. 10 r/s) ?joyvel
Activation:	!joy 1

Joystick filter time constant	
Command:	!joyoutpass or ?joyoutpass
Description of function:	*With this command, the input values can be filtered to prevent jerky changes in speed. (ramp function)*
Parameter:	x, y, z, a 0 – 500000µs
Description	!joyoutpass x 40 (the filter time constant for the X-axis is set to 40µs
Feedback:	Set filter time constant
Error code:	--
Example:	! joyoutpass 10 10 10 10 (Set filter time constant to 10µs for all axes) ? joyoutpass
Activation:	!joy 1

Joystick with current reduction	
Command:	!joyredcur or ?joyredcur
Parameter:	X, y, z, a 0 or 1
Description:	The current reduction in joystick mode can be switched on and off with the input of joyredcu. When the current reduction is active, the motor current is reduced to the value set with "reduction" when the joystick is not deflected and is in the middle position. !joyredcur x 1 (The current reduction for the X-axis is active.)
Feedback:	Settings:
Error code:	--
Example:	!joyredcur 1 1 1 0 ?joyredcur
Activation:	!joy 1

Joystick Window (joywindow)	
Command:	!?joywindow
Parameter:	0 – 100
Description:	With joywindow the analogue range is determined in which the axes move. Applies to all axes.
	joywindow 10 => the axes move only, if the excursion of the joystick is greater than 10 (points).
	?joywindow => Read out the joystick – window.
	Example: zero position joystick = 512 (analogue value) Joywindow = 10 i.e., that the axes move within the values < 502 and > 522.
Feedback:	Preset window
Error code:	--
Example:	?joywindow (reading window size)
Activation:	!joy 1

Enable joystick	
Command:	!joyenable or ?joyenable
Parameter:	0 or 1
Description:	Connected joystick axes can be enabled or disabled with the input of joyenable. !joyenable x 1 (The joystick for the X-axis is enabled)
Feedback:	Settings:
Error code:	--
Example:	!joyenable 1 1 1 0 (Enabling for X Y Z) ?joyenable
Activation:	!joy 1

Joystick axis assignment	
Command:	!joytoaxis or ?joytoaxis
Parameter:	X y z a 0 through 4
Description:	The joystick axes can be assigned at will. As there are generally only 2 and 3 axes joysticks, you can use the Z-joystick to move the 4th axis, for example. !joytoaxis a 3 Assignment of one joystick input to two axes simultaneously is not permitted.
Feedback:	Preset allocating.
Error code:	--
Example:	!joytoaxis 1 2 3 4 (X=1 Y=2 Z=3 A=4) ?joytoaxis
Activation:	!joy 1

Joystick	
Command:	!joy or ?joy
Parameter:	0 or 1
Description	The joystick can be switched on and off with this command !joy 1 (Joystick On) / !joy 0 (Joystick Off)
Feedback:	@@@@
Error code:	--
Example:	!joy 1 ?joy

Jog mode speed	
Command:	!tippvel or ?tippvel
Description of function:	*The speeds of travel in jog mode can be set with this command.*
Parameter:	x, y, z, a 0 – 70 R/sec.
Description	!tippvel x 40 (The X-axis travels in jog mode at 40 revs./sec.)
Feedback:	Adjusted speed
Error code:	--
Example:	!tippvel 10 10 10 10 (all axes are operated at max. 10 r/s) ?tippvel
Activation:	!tipp 1

Jog mode filter time constant	
Command:	!tippoutpass or ?tippoutpass
Description of function:	*With this command, the input values can be filtered to prevent jerky changes in speed. (ramp function)*
Parameter:	x, y, z, a 0 – 500000µs
Description	!tippoutpass x 40 (The filter time constant for the X-axis is set to 40µs)
Feedback:	Set filter time constant
Error code:	--
Example:	!tippoutpass 10 10 10 10 (The filter time constant is set at 10µs for all axes) ?tippoutpass
Activation:	!tipp 1

Jog mode direction	
Command:	!tippdir or ?tippdir
Parameter:	x y z a 0 or 1
Description:	The direction of rotation of the motors is changed for the direction keys with the input of tippdir. !tippdir x 1 (The direction of rotation for the X-axis has been reversed.)
Feedback:	Set jog mode directions.
Error code:	--
Example:	!tippdir 0 1 0 0 ?tippdir
Activation:	!tipp 1

Enable jog mode	
Command:	!tippenable or ?tippenable
Parameter:	0 or 1
Description:	The connected axes can be enabled or disabled for jog mode with the input of tippenable. !tippenable x 1 (The X-axis is enabled for jog mode)
Feedback:	Settings:
Error code:	--
Example:	!tippenable 1 1 1 0 (Enabling for X Y Z) ?tippenable
Activation:	!tipp 1

Jog mode with current reduction	
Command:	!tippredcur or ?tippredcur
Parameter:	X, y, z, a 0 or 1
Description:	The current reduction in jog mode can be switched on and off with the input of tippredcur. When the current reduction is active, the motor current is reduced to the value set with "reduction" when all axes are at a standstill. !tippredcur x 1 (The current reduction for the X-axis is active.)
Feedback:	Settings:
Error code:	--
Example:	!tippredcur 1 1 1 0 ?tippredcur
Activation:	!tipp 1

Switch jog mode on or off	
Command:	!tipp or ?tipp
Parameter:	0 or 1
Description:	Jog mode can be switched on with !tipp 1 The trackball can be switched off with !tipp 0
Feedback:	Settings:
Error code:	--
Example:	!tipp 1 (jog mode is switched on) ?tipp (Query whether jog mode is switched on or off)
Activation:	immediately

Trackball speed	
Command:	!tbvel or ?tbvel
Description of function:	*The speeds of travel in trackball mode can be set with this command.*
Parameter:	x, y, z, a 0 – 70 R/sec.
Description	!tbvel x 40 (the X-axis travels in trackball mode at max. 40 revs./sec.)
Feedback:	Adjusted speed
Error code:	--
Example:	!tbvel 10 10 0 0 (all axes are operated at max. 10 r/s) ?tbvel
Activation:	!tb 1

Trackball speed	
Command:	!tboutpass or ?tboutpass
Description of function:	*With this command, the input values can be filtered to prevent jerky changes in speed. (ramp function)*
Parameter:	x, y, z, a 0 – 500000µs
Description	!tboutpass x 40 (the filter time constant of the X-axis is set to 40µs)
Feedback:	Set filter time constant
Error code:	--
Example:	!tboutpass 10 10 0 0 (the filter time constants of all axes is set to 10µs) ?tboutpass
Activation:	!tb 1

Trackball direction	
Command:	!tbdir or ?tbdir
Parameter:	x y z a 0 or 1
Description:	With the input of tbdir, the direction of rotation of the motors are reversed for the trackball !tbdir x 1 (The direction of rotation for the X-axis was changed.)
Feedback:	Set trackball directions.
Error code:	--
Example:	!tbdir 0 1 0 0 ?tbdir
Activation:	!tb 1

Enable trackball	
Command:	!tbenable or ?tbenable
Parameter:	0 or 1
Description:	The connected axes can be enabled or disabled for trackball mode with the input of tbenable. !tbenable x 1 (The X-axis is enabled for trackball mode)
Feedback:	Settings:
Error code:	--
Example:	!tbenable 1 1 0 0 (Enabling for X Y) ?tbenable
Activation:	!tb 1

Trackball with current reduction	
Command:	!tbredcur or ?tbredcur
Parameter:	X, y, z, a 0 or 1
Description:	The current reduction in trackball mode can be switched on and off with the input of tbredcur. When the current reduction is active, the motor current is reduced to the value set with "reduction" when all axes are at a standstill. !tbredcur x 1 (The current reduction for the X-axis is active.)
Feedback:	Settings:
Error code:	--
Example:	!tbredcur 1 1 1 0 ?tbredcur
Activation:	!tb 1

Trackball axis assignment	
Command:	!tbtoaxis or ?tbtoaxis
Parameter:	X y z a 0 = no allocation of axes 1 = Trackball horizontal 2 = Trackball vertical
Description:	The trackball axes can be assigned at will with the input of tbenable. As a trackball only has 2 axes, any axes can be assigned to it. !tbtoaxis z 1 (The Z-axis is moved with the horizontal trackball direction.) Assignment of one trackball input to two axes simultaneously is not permitted.
Feedback:	Preset allocating.
Error code:	--
Example:	!tbtoaxis 1 2 0 0 (X=1 Y=2 Z=0 A=0) ?tbtoaxis
Activation:	!tb 1

Switch trackball on or off	
Command:	!tb or ?tb
Parameter:	0 or 1
Description:	The trackball can be switched on with !tb 1 The trackball can be switched off with !tb 0
Feedback:	Settings:
Error code:	--
Example:	!tb 1 (Trackball is switched on) ?tb (Query whether the trackball is switched on or off)
Activation:	immediately

4.10 In/Outputs

The LSTEPexpress or PCIexpress can be equipped with 16 digital inputs and outputs, at option. To use these inputs and outputs, you must order the appropriate control model.

Digital inputs	
Command:	?digin
Parameter:	0 through 15
Description:	?digin ➔ Read all input pins
	?digin 8 ➔ Read input pin 8
Feedback:	Status of the input pins
Error code:	--
Example:	?digin (Read all input pins)

Dgital outputs	
Command:	!digout or ?digout
Parameter:	0 through 15
Description:	!digout 11110000 ➔ Output pins 0,1,2,3 are set to "1" and output pins 4,5,6,7 are set to"0".
	!digout 5 1 ➔ Output pin 5 is set to "1".
	?digout ➔ Read the current status of all output pins.
	?digout 8 ➔ Read the current status of output pin 8
Feedback:	Status of the output pins
Error code:	--
Example:	!digout 7 0 (Set output pin 7 to "0") ?digout (Read all output pins)

Analogue outputs	
Command:	!anaout or ?anaout
Parameter:	0 through 100 % 0 and 1 (analogue channels) c (c = channel)
Note:	Channels 0 and 1 are on the controller's multifunction port.
Description:	!anaout 100 50 ➔ The first analogue channel is set to 100% (full power) and the second to 50% (half power).
	!anaout c 1 25 ➔ Analogue channel 1 is set to 25%.
	?anaout ➔ Read the current status of all analogue channels.
	?anaout c 0 ➔ Read the current status of analogue channel 0.
Feedback:	Status of the modulation in percent of the analogue channels.
Error code:	--
Example:	!anaout c 1 0 (Set analogue channel 1 to "0") ?anaout (Read all analogue channels)

Analogue inputs	
Command:	?anain
Parameter:	0 through 13 (analogue channel) c (c = channel)
Description:	?anain c 2 => Read the current status of analogue channel 2
Feedback:	Status depending on the analogue channel
Error code:	--
Example:	?anain (Read the current status of all analogue channels)

Channel / Analogue inputs			
Channel	**MFP**	**50pol (25pol)**	**Function**
0	MFP	Pin 1 (24)	Joystick Axis 1
1	MFP	Pin 2 (12)	Joystick Axis 2
2	MFP	Pin 3 (25)	Joystick Axis 3
3	MFP	Pin 4 (n.c.)	Joystick Axis 4
4	MFP	Pin 11 (n.c.)	Potentiometer 1
5			Motor voltage
6	MFP	Pin 5 (8)	analogue Input / 0…5V
7	MFP	Pin 6 (20)	analogue Input / 0…5V
8	MFP	Pin 7 (7)	analogue Input / 0…5V
9	MFP	Pin 8 (19)	analogue Input / 0…5V
10	MFP	Pin 9 + 10	Analogue Input / PT 100
11			Driver voltage for power amplifiers
12	MFP	Pin 12 (n.c.)	Potentiometer 2
13	MFP	Pin 13 (n.c.)	Potentiometer 3

4.11 Interpretation of Clock Pulse and Direction of Rotation Settings

As an option, the axes can also be moved forward and backward using clock signals dependent on the direction of rotation signal instead of using vector commands or manual input (joystick, trackball, jog mode, handwheel). This mode is also possible asynchronous to travels that have been initiated by means of travel commands. The multifunction port MFP is available for this purpose.

4.11.1 Travel Range Monitoring

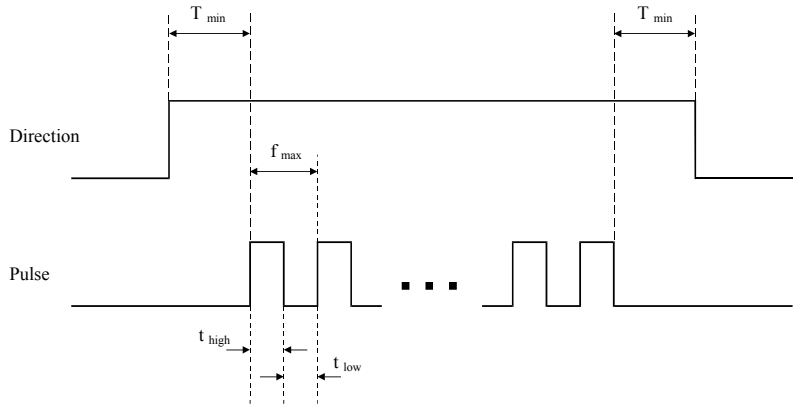
In CFB mode, monitoring is done to ensure that the permitted travel limits are not exceeded. The travel limits may have been determined using the combination ‚*Calibrate*' and ‚*Measure range*'. Another option is to set the travel limits by means of commands.
If the controller determines that a travel limit would be exceeded due to the cumulative counter pulses, all other movement of the axis in this direction are suppressed. Movement in the opposite direction is, however, still possible. No message is transmitted to the PC.

Information:	The application program is responsible for ensuring that the maximum Start/Stop frequencies of the drive are not exceeded and that the respective axis is not overloaded acceleration-wise.

4.11.2 Time-Related Boundary Conditions for the Signals

The timing of the flanks of the clock and direction of rotation signals of an axis is subject to the following boundary conditions

- The next clock pulse may be applied T_{min} after each polarity reversal at the earliest.
- The clock pulses must be stopped T_{min} before each polarity reversal of the direction signal at the latest
- T_{min} is presently 50µs.
- The maximum frequency of the clock pulses must not exceed f_{max} = 833 kHz. The minimum times T_{low} = 600ns and T_{high} = 600ns must be adhered to.
- To protect the controller inputs, input filters with 470Ω and 220pF are used. It is therefore important to ensure that the clock source has enough driver power.



Clock forward/ backward axis assignment	
Command:	!tvrtoaxis or ?tvrtoaxis
Parameter:	x y z a 0 = no allocation of axes 1 = QEP 1 2 = QEP 2
Description:	With the input of tvrtoaxis, the QEP inputs can be assigned at will. As there are only two QEP inputs available on the MFP, these can be assigned to any axes. !tvrtoaxis z 1 (The Z-axis is travelled by means of the QEP1 inputs.)
Feedback:	Preset allocating.
Error code:	--
Example:	!tvrtoaxis 1 2 0 0 (X=1 Y=2 Z=0 A=0) ?tvrtoaxis
Activation:	immediately

Clock pulse Forward/Back On/Off	
Command:	!?tvr
Parameter:	x, y, z, a 0 or 1
Description:	!tvr x 0 Clock forward/backward Off for x-axis !tvr x 1 Clock forward/backward On for x-axis ?tvr Query clock forward/backward status
Feedback:	0 or 1
Error code:	--
Example:	!tvr 0 1 1 0 (C-F/B active for the axes Y + Z)
Activation:	immediately

Clock foward/backward mode	
Command:	!tvrm or ?tvrm
Parameter:	x, y, z and a 0, 1, 2, 3, 4
Description:	**Functions:**
	0 ➔ Clock pulse Forward /Back "OFF"
	1 ➔ Normal clock pulse Forward/Back processing.
	2 ➔ Processing of clock pulse Forward/Back with a factor.
	3 ➔ Clock forward/backward processing requires external enabling by means of Start/Stop inputs.
	4 ➔ Combination of 2 & 3.
	commands:
	!tvrm 1 ➔ Normal C-F/B processing (with each clock pulse, the selected axis is moved one microstep) for the X-axis
	!tvrm 2 ➔ C-F/B processing with factor (with each clock pulse the value of the factor tims 1 microstep is travelled) for the X-axis
	!tvrm 3 ➔ The axis is only travelled with it is enabled by means of the Start/Stop input. (The axes can thus still be disabled separately, or a fixed clock frequency can be applied to the clock input.) for the X-axis
	?tvrm ➔ The set mode is displayed
Feedback:	0 through 4
Error code:	--
Example:	!tvrm 4 4 4 4 (C-F/B mode 4 is set for all axes) ?tvrm
Activation:	!tvr 1

Factor for clock pulse Forward/ Back		
Command:	!tvrf or ?tvrf	
Parameter:	x, y, z and a -10000 through + 10000 As an alternative to the forward/backward signal, the direction of rotation can be reversed with a negative factor	
Description:	!tvrf 1 1 0 0	➔ Clock pulse Forward/Back is to work with the factor 1 for the X- and Y-axis i.e. one clock pulse = one motor increment (MI)].
	!tvrf a 10	➔ For axis a, clock forward/backward is to work with the factor 10 (i.e. one clock pulse = 10 MI)
	?tvrf	➔ All preset factors are displayed.
	?tvrf z	➔ The actual z-axis factor is displayed
Feedback:	Factor values	
Error code:	--	
Example:	!tvrf 10 ?tvrf	(Factor = 10.00 for the x-axis) (One pulse = ten Motor increments)
Activation:	immediately	

4.12 Interpretation of Incremental Measuring Systems

Axes with and without encoders can be operated simultaneously on the controller. During the calibration process or the AutoCommutation, the controller checks whether encoders are connected, provided that the latter have been enabled with *enctoaxis*. To instruction *?enc* can be used to see the results of these checks. The controller does not however distinguish between incorrectly connected encoders and missing encoders.
Also while calibration to reference mark, the axis drives in negative direction into the zero-proximity switch, does a inversion of direction and drives with the speed which was set via calbspeed to the reference mark. If a system does not have a proximity switch (i.e. a turn axis) it moves directly to the reference mark, if all proximity switches were deactivated prior.

Positioning of transmitter input and axis	
Command:	enctoaxis
Parameter:	1…6
Note:	The LSTEP-PCIexpress has 6 inputs for QEP-Encoder, which can be assigned to an axis as desired. The inputs 1+2 can be connected oer the MFP The inputs 3…6 over the encoder inputs
Description:	!enctoaxis 3 4 5 6 ➔ Encoder X on input 3
	➔ Encoder Y on input 4
	➔ Encoder Z on input 5
	➔ Encoder A on input 6
Feedback:	Transmitter positioning
Error code:	--
Example:	!enctoaxis 1 2 (enctoaxis 1 2 (Encoder X on input 1, and Y on input 2) ?enctoaxis
Activation:	immediately

Encoder type	
Command:	!enctype or ? enctype
Parameter:	x, y, z and a 0 through 8
Note:	The encoders must be properly set prior to being put into operation
Description	!enctype 0 to 8 Rotary encoder: 1=1Vss, 2=11µA, 3=MR, 4=TTL Linear encoder: 5=1Vss, 6=11µA, 7=MR, 8=TTL
Feedback:	Encodertype
Error code:	--
Example:	?enctype
Activation:	!validconfig resp. !validpar

Encoder count direction		
Command:	?encdir	
Parameter:	x, y, z and a 0 or 1 (1 = reverse count direction)	
Note:	An incorrect count direction can be reverse by means of a command, without having to turn the encoder or a phase	
Description:	!encdir 0 1 0 0	➔ Count direction reversed for the Y-axis.
	!encdir x 1	➔ Count direction reversed for the X-axis.
Feedback:	0 or 1	
Error code:	--	
Example:	?encdir	
Activation:	!validconfig resp. !validpar	

Encoder Mask For Detected Encoders		
Command:	?enc	
Parameter:	x, y, z and a 0 or 1 (On,Off)	
Note:	A 1 comes for all detected encoders	
Description:	?enc	➔ All encoder statuses are displayed.
	?enc x	➔ Display of the encoder mask for the X-axis.
Feedback:	Encoder status	
Error code:	--	
Example:	?enc	

Signal Periods/ Linear Encoder		
Command:	!encperiod or ?encperiod	
Parameter:	x, y, z and a Period length of the set dimension	
Description:	!encperiod 0.5 0.020	➔ The length of the encoder signal period is 500µm for the X-axis and 20µm for the Y-axis.
	?encperiod	➔ All encoder period lengths are displayed.
	?encperiod x	➔ Display of the length of the encoder period length the X-axis.
Feedback:	Encoder period length in the setting of the dimension	
Error code:	--	
Example:	!encperiod x 0.1 (Length of encoder period for the X-axis is 0.1mm) ?encperiod	
Activation:	!validconfig resp. !validpar	

Encoder pole pairs / rotary encoder	
Command:	!encpolepairs or ?encpolepairs
Parameter:	x, y, z, a 1 to 50,000
Description:	Shows the amount of encoder signal periods per motor revolution. The ratio of the periods to the unit factor should result to a whole number (integer), if the encoder is mounted behind a gear.
Feedback:	-
Error code:	--
Example:	!encpolepairs 500 500 500 ?encpolepairs
Activation:	!validconfig resp. !validpar

Encoder Reference Signal		
Command:	!encref or ?encref	
Parameter:	x, y, z or a 0 or 1	
Description:	!encref 1 1 0	➔ The reference signal of the X-and Y-axis encoders is interpreted when calibration is done.
	!encref z 1	➔ The reference signal of the Z-axis encoder is interpreted when calibration is done.
	?encref	➔ The present setting is displayed.
	?encref y	➔ The present setting for the Y-axis is displayed.
Feedback:	0 or 1	
Error code:	--	
Example:	!encref x 0 (No reference signal interpretation for the X-axis) ?encref	
Activation:	immediately	

Polarity of reference marks of QEP-Encoder	
Command:	!encrefpol or?encrefpol
Parameter:	x, y, z, a 0 or 1 (0 = negative, 1 = positive reference impulse)
Note:	With assistance of this command the polarity of the reference marks of the QEP-Encoder can be adjusted.
Description:	!encrefpol 1 1 0 0 ➔ Encoder X+Y positive reference impulse Encoder Z+A negative reference impulse
	?encrefpol ➔ All adjustments are displayed.
	?encrefpol x ➔ Adjustment of X-axis is displayed.
Feedback:	Adjustments (for ex.: 1 1 0 0)
Error code:	--
Example:	!encrefpol 1 1 (Encoder X+Y positive reference impulse) ?encpos
Activation:	immediately

Encoder Position	
Command:	!encpos or ?encpos
Parameter:	X, y, z, a 0 or 1
Description:	!encpos 1 1 ➔ When the positions are queried, the encoder values of the detected encoders for the X and Y axes are displayed.
	?encpos ➔ The actual settings for all axes are displayed.
Feedback:	0 or 1
Error code:	--
Example:	!encpos 0 0 0 0 (encoder position display for all axes "OFF") ?encpos
Activation:	immediately

Encoder Error	
Command:	!encerr or ?encerr
Parameter:	X, y, z, a 0 or e
Description:	!encerr 0 0 0 ➔ Clear encoder error messages from X-, Y-and Z-axes.
	!encerr a 0 ➔ Clear encoder error message from A-axis.
	?encerr ➔ The present encoder error messages for all axes are displayed.
	?encerr z ➔ The present encoder error message for the Z-axis is displayed.
Feedback:	0 or e
Error code:	--
Example:	!encerr 0 (Clear encoder error message from A-axis) ?encerr
Activation:	immediately

4.13 Controller Settings For LSTEP

Position controller kp-band	
Command:	?poskonkp or !posconkp
Parameter:	0 – 4000 %
Note:	
Description:	!posconkp x 100 (ku-band of the x-axis is set to 100%)
Feedback:	Set value
Error code:	
Example:	!posconkp a 100 (ku-band of the a-axis is set to 100%) ?posconkp z (Query the kp-band of the z-axis)
Activation:	!validconfig resp. !validpar

Position controller filter time constant	
Command:	!posconoutpass or ?posconoutpass
Parameter:	0 – 100000µs
Note:	
Description:	!posconoutpass x 320 (time constant of the position controller output filter is set to 320µs)
Feedback:	Set value
Error code:	
Example:	
Activation:	!validconfig resp. !validpar

Position controller axis enable	
Command:	!posconenable or ?posconenable
Parameter:	0 or 1
Note:	Enabling or disabling of the position controller
Description:	posconenable x 1 (Enable for the X-axis) !posconenable 1 1 1 1 (Enable for all axes) ?posconenable (Query the settings)
Feedback:	0 or 1
Error code:	
Example:	!posconenable 0 1 0 (Enable for the Y-axis)
Activation:	!validconfig resp. !validpar

Switch position controller on and off	
Command:	!poscon or ?poscon
Parameter:	0 or 1
Note:	Switches the position controller on or off
Description:	!poscon 1 (Switch-on for all enabled axes) !poscon 0 (Switch off all position controllers)
Feedback:	0 or 1
Error code:	
Example:	?poscon (Position controller query)
Activation:	immediately

Deviation check - range	
Command:	?deviationrange or !deviationrange
Parameter:	depends on the dimension
Note:	The deviation check range that must be exceeded to trigger the monitor is set with this command.
Description:	!deviationrange x
Feedback:	Set value depending on dimension
Error code:	5(?err) 1108(error list) 12(?sysstat)
Example:	?deviationrange (read out the set ranges for all axes)
Activation:	!validconfig resp. !validpar

Deviation check - time frame	
Command:	?deviationtime or ! deviationtime
Parameter:	0 through 10000 ms
Note:	The time after which the position controller is switched off if the deviation range is exceeded can be set with this command.
Description:	!deviationtime x 1000 (if the deviation range is exceeded longer than 1000ms, the controller is switched off.)
Feedback:	Value in ms
Error code:	5(?err) 1108(error list) 12(?sysstat)
Example:	?deviationtime (read the preset time)
Activation:	!validconfig resp. !validpar

Switch deviation check on and off	
Command:	?deviationcheck or ! deviationcheck
Parameter:	0 or 1
Description:	!deviationcheck x 1 (the deviation check is active for the X-axis)
Feedback:	0 or 1
Error code:	
Example:	?deviationcheck (readout of the setting)
Activation:	!validconfig resp. !validpar

Target Window		
Command:	!twi or ?twi	
Parameter:	X, y, z and a 1 to 25000 (motor increments) 0.1 to spindle pitch/2 (µm) 0.0001 to spindle pitch/2 (mm)	
Note:	The input and output values depend on the dimension.	
Description:	!twi 1.0 0.002	➔ The target window is 1 mm for the X-axis and 2µm for the Y-axis (when Dim = 2). The other axes remain unchanged.
	!twi z 0.1	➔ The target window is set to 0.1µm for the Z-axis (when Dim = 1).
	?twi	➔ All preset target windows are displayed.
	?twi x	➔ The preset target window for the X-axis is displayed.
Feedback:	Target window which has actually been set (rounding errors are displayed)	
Error code:	--	
Example:	!twi 10 (The X-axis has a target window of 10 motor increments (when Dim = 0)). ?twi	
Activation:	!validconfig resp. !validpar	

Target Window	
Command:	!poswindowrange or ? poswindowrange Alternative to "twi"
Parameter:	X, y, z and a 1 to 25000 (motor increments) 0.1 to spindle pitch/2 (µm) 0.0001 to spindle pitch/2 (mm)
Note:	The input and output values depend on the dimension.
Description:	!poswindowrange 1.0 0.002 ➔ The target window is 1 mm for the X-axis and 2µm for the Y-axis (when Dim = 2). The other axes remain unchanged.
	!poswindowrange z 0.1 ➔ The target window is set to 0.1µm for the Z-axis (when Dim = 1).
	?poswindowrange ➔ All preset target windows are displayed.
	?poswindowrange x ➔ The preset target window for the X-axis is displayed.
Feedback:	Target window which has actually been set (rounding errors are displayed)
Error code:	--
Example:	!poswindowrange 10 (The X-axis has a target window of 10 motor increments (when Dim = 0)). ?twi
Activation:	!validconfig resp. !validpar

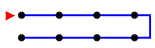
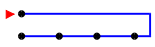
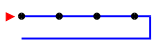
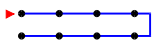
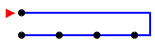
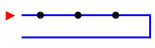
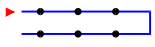
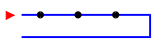
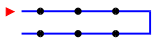
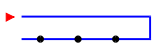
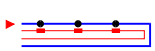
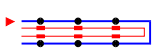
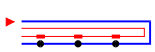
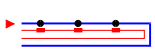
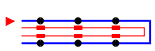
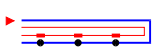
4.14 Configuration of the Trigger- Output signal

These Commands synchronizes an external unit i.e. video camera or laser. The signals are sent via a multi function port, which is available.

Trigger	
Command:	?trig or !trig
Parameter:	0 or 1 (OFF / ON)
Description:	!trig 1 ➔ Trigger „ON“
	?trig ➔ Shows the current status of the trigger processing
Important!	Switch on the trigger, only after all settings are transferred (exception with Triggermode 99).
Feedback:	ON or OFF
Error code:	--
Example:	!trig 0 (Trigger operation „OFF“) ?trig

Trigger Axis	
Command:	?triga or !triga
Parameter:	X, y, z or a
Description:	!triga y ➔ Trigger in reference to the x-axis
	?triga ➔ Shows the current reference axis
Feedback:	X, y, z or a
Error code:	--
Example:	!triga x (Trigger referring to the x-Axis) ?triga
Activation:	!trig 1

Trigger Mode	
Command:	?trigm or !trigm
Parameter:	0 – 17 or 99

Trigger Mode		
Description:	!trigm 0 → 	 high active
	!trigm 1 → 	 high active
	!trigm 2 → 	 high active
	!trigm 3 → 	 low active
	!trigm 4 → 	 low active
	!trigm 5 → 	 low active
The trigger modes 6-11 generate their first trigger pulse after half of the set trigger path and then, dependent on the trigger path, more pulses are generated.	!trigm 6 → 	 high active
	!trigm 7 → 	 high active
	!trigm 8 → 	 high active
	!trigm 9 → 	 low active
	!trigm 10 → 	 low active
	!trigm 11 → 	 low active
	!trigm 12 → 	 high active
	!trigm 13 → 	 high active
	!trigm 14 → 	 high active
	!trigm 15 → 	 low active
	!trigm 16 → 	 low active
	!trigm 17 → 	 low active
	!trigm 99	
With the trigger mode 99, at the beginning and at the end of the uniform motion, a trigger impulse is generated. A certain sequence has to be observed during the execution of the function. The command !trigm 99 needs to be send as the last command after the common settings, because it would be deleted with another mode setting.		
Feedback:	0 – 23 (Mode)	
Error code:	--	
Example:	!trigm 3 (Trigger Mode 3) ?trigm	

Legend				
▸	• • • • • •			
Starting point	Trigger point	Track	External Trigger signal	low active
Activation:	!trig 1			

Trigger Signal	
Command:	?trigs or !trigs
Parameter:	3 – 120 (µs)
Description:	!trigs 4 ➔ Trigger-Signal length 4 µs
	?trigs ➔ Shows the current status of the set Trigger-Signal length.
Feedback:	3 – 120 (µs)
Error code:	--
Example:	!trigs 3 (Trigger-length of signal = 3µs) ?trigs
Activation:	!trig 1

Trigger Distance	
Command:	?trigd or !trigd
Parameter:	Depends on the dimension
Description:	!trigd 1 ➔ Trigger-Distance 1mm (bei Dim 2)
	?trigd ➔ Shows the current Trigger distance.
Feedback:	Distance
Error code:	--
Example:	!trigd 1000 (1000 MI Trigger distance for Dim 0) ?trigd
Activation:	!trig 1

Trigger Counter	
Command:	?!trigcount;
Parameter:	0 through 2147483647
Note:	Es werden alle OFFgegebenen Trigger gezählt
Description:	
Feedback:	Amount of executed Trigger
Error code:	--
Example:	?trigcount ; (reads counter setting)

4.15 Configuration Of The Snapshot Input

The current positions can be saved in the controller whilst travelling is in progress with these instructions. These values can then subsequently be read out or the positions approached. This signal is set via the multi-function port, which is available as an option.

Snapshot	
Command:	?sns or !sns
Parameter:	0 or 1
Description:	!sns 1 ➔ Snapshot "ON"
	?sns ➔ Gives the present snapshot status.
Feedback:	Snapshot status
Error code:	--
Example:	!sns 0 (Snapshot "OFF") ?sns

Snapshot-Level (Polarity)	
Command:	?snsl or !snsl
Parameter:	0 or 1
Description:	!snsl 1 ➔ Snapshot is high-active. 
	?snsl ➔ Gives the current polarity
Feedback:	Current polarity
Error code:	--
Example:	!snsl 0 (Snapshot is low-active)  ?snsl
Activation:	!sns 1

Snapshot Filter	
Command:	?snsf or !snsf
Parameter:	0 – 100 ms
Note:	Serves as input filter for rebounding switches
Description:	!snsf 10 => 10 ms input filter ?snsf => Gives the current status
Feedback:	Current filter time
Error code:	--
Example:	!snsf 0 (no input filter) ?snsf
Activation:	!sns 1

Snapshot-Mode	
Command:	?snsm or !snsm
Parameter:	0 or 1
Description:	!snsm 1 ➔ Snapshot "Automatic". The position is automatically approached after the first impulse.
	?snsm ➔ Gives the present mode
Feedback:	Snapshot mode
Error code:	--
Example:	!snsm 0 (Normal snapshot) ?snsm
Activation:	!sns 1

Snapshot Counter	
Command:	?snsc
Parameter:	–
Description:	Contents are deleted after every "read".
	?snsc ➔ Gives the number of initiated snapshots.
Feedback:	Number of initiated snapshots
Error code:	--
Example:	?snsc

Snapshot Position		
Command:	!snsp or ?snsp	
Parameter:	X,Y,Z and A Min./max. range of travel	
Note:	Input and output depend on the dimension.	
Description:	!snsp 1000 2000 3000	➔ Position values are set for the X-, Y-, and Z-axes.
	!snsp y 2000	➔ Position of the Y-axis is set.
	?snsp	➔ Inquire present snapshot position of all axes.
	?snsp z	➔ Inquire present snapshot position of Z-axis.
Feedback:	Position values	
Error code:	--	
Example:	!snsp 100 200 (Set the X-and Z-axis positions) ?snsp (Inquire the snapshot positions of all axes)	

Snapshot Position Array		
Command:	?snsa	
Parameter:	X,Y,Z and A 1 – 200 (Positions)	
Note:	Input and output depend on the dimension.	
Description:	?snsa 33	➔ Inquire the snapshot position 33 of all axes
	?snsa z 99	➔ Inquire the snapshot position 99 of z-axis.
Feedback:	Position values	
Error code:	--	
Example:	?snsa 1 (Inquire the snapshot position 1 of all axes)	

5 Plug Assignment and Hardware

5.1 Multi-Function Port Pin Assignment (ST14, 50pol pin header on 25pol, or 50pol D-Sub-port)

The multi function port exists in two stages of expansion, as 25pol and 50pol Dsub port.
Due to the function variety the pins of the multi function port (MFP) are partially multiple used. Depending on how the controller is equipped, this means that only one signal output or input is present on a pin of the MPF.
The required functionality has to been defined with the order.
Standard: Trigger, Snapshot and Stop-Input.

50 pol Pin	25 pol Pin	Assignment	Signal	Remarks
1	24	1	AIN-0	Analogue Input/Joystick axis 1
2	12	1	AIN-1	Analogue Input/Joystick axis 2
3	25	1	AIN-2	Analogue Input/Joystick axis 3
4	-	1	AIN-3	Analogue Input/Joystick axis 4
5	8	1	AIN-6	Analogue Input/0…5V
6	20	1	AIN-7	Analogue Input/0…5V
7	7	1	AIN-8	Analogue Input/0…5V
8	19	1	AIN-9	Analogue Input/0…5V
9	6	1	AIN-10	Analogue Input/PT100
10	-	1	AIN-10	Analogue Input/PT100
11	-	1	AIN-4	Analogue Input/Potentiometer 1
12	-	1	AIN-12	Analogue Input/Potentiometer 2
13	-	1	AIN-13	Analogue Input/Potentiometer 3
14	18	1	AOUT-0	Analogue Output/0…10V or± 10V
15	-	1	AOUT-1	Analogue Output/0…10V or± 10V
16	-	1 2	Cycle-Output A TTL-Output	
17	-	1 2	V/R-Output A TTL-Output	
18	-	1 2	Cycle-Output B TTL-Output	
19	-	1 2	V/R-Output B TTL-Output	
20	1	1 2 3	Cycle-Input A Transmitter A/track A TTL-Input	For: Hand wheel/Trackball/Transmitter
21	2	1 2 3	V/R-Input A Transmitter A/track B TTL-Input	For: Hand wheel/Trackball/Transmitter
22	5	1	Transmitter A/Index	
23	3	1 2	Cycle-Input B Transmitter	For: Hand wheel/Trackball/Transmitter

50 pol Pin	25 pol Pin	Assignment	Signal	Remarks
		3	B/track A TTL-Input	
24	4	1 2 3	V/R-Input B Transmitter B/track B TTL-Input	For: Hand wheel/Trackball/Transmitter
25	14	1	Transmitter B/Index	
26	15		Triggerout 1	
27	-		Triggerout 2	
28	22		Snapshot-Input	
29	23		Stopp-Input	
30	10		Joystick On	
31	-		n.c.	
32	-		n.c.	
33	-		Driver voltage	For shut down of the power amplifier driver
34	-		Driver relais COM	Feedback joint contact
35	-		Driver relais NO	Feedback power amplifier active
36	-		Driver relais NC	Feedback power amplifier shut down
37	-		Motor brake A+	
38	-		Motor brake A-	
39	-		Motor brake B+	
40	-		Motor brake B-	
41	13		+3,0Vref	
42	-		+3.3V	
43	17		+5.0V	
44	-		+5,0Vanalogue	
45	21		+12V	
46	9		-12V	
47	11/16		GND	
48	11/16		GND	
49	11/16		GND	
50	11/16		GND	

5.2 MFP Adapter for two LSTEP-PCIexpress

25pol / ST1	Function	50pol / ST2 card 1	50pol / ST3 card 2
1	Stopp Input card 1	29	
14	Stopp Input card 2		29
2	Driver voltage card 1	33	
15	Driver voltage card 2		33
3	Driver relais COM card 1	34	
4	Driver relais NO card 1	35	
5	Driver relais NC card 1	36	
16	Driver relais COM card 2		34
17	Driver relais NO card 2		35
18	Driver relais NC card 2		36
6	GND card 1	47…50	
19	GND card 2		47…50
7	+5V card 1	43	
20	+5V card 2		43
8	+12V card 1	45	
21	+12V card 2		45
9	Triggerout 1, card 1	26	
22	Triggerout 1, card 2		26
10	Snapshot card 1	28	
23	Snapshot card 2		28

5.3 The Pin Assignment of RS232 Interface

Pin	Signal	Remarks
1	n.c.	
2	RxD	LSTEP receive line
3	TxD	LSTEP transmit line
4	GND	
5	GND	Signal ground
6	+5V	
7	RTS	Request to send, from LSTEP
8	CTS	Clear to send, from PC
9	either n.c. +5V or +12V DC	

5.4 The RS 232 Interface Cable

LSTEP		PC		
9 Pole, Sub-D Plug	Assignment	9 Pol Sub-D	25 Pole, Sub-D	Assignment
1	n.c.	-	-	-
2	RxD	3	2	TxD
3	TxD	2	3	RxD
4	n.c.	-	-	-
5	GND	5	7	GND
6	n.c.	-	-	-
7	RTS	8	5	CTS
8	CTS	7	4	RTS
9	n.c.	-	-	-

5.5 The Pin Assignment of USB Interface (plug type B)

Pin	Signal	Remarks
1	Vin (+5V)	
2	USB-Slave D-	
3	USB-Slave D+	
4	GND	

5.6 The CAN Interface (ST4, 10pol pin header on 9pol D-Sub)

Attention! Presently no CAN-protocol is supported.

Pin	Assignment	Pin-No.	Assignment
1	n.c.	6	CAN GND
2	CAN L	7	CAN H
3	CAN GND	8	n.c.
4	n.c.	9	CAN V+ (J2 plugged: +12V)
5	CAN screen (GND)	10	n.c.

5.7 Power Supply 12V (ST10, 4pol PC-Power Supply Plug)

Pin	Signal	Remarks
1	+12Vin	
2	GND	
3	GND	
4	+5Vin	

5.8 Motor Voltage Supply up to 48V (ST17, 6pol Tyco Print plug)

Pin	Signal	Remarks
1	+ UMOT	Motor voltage >24V to 48V
2	+ UMOT	Motor voltage >24V to 48V
3	+ 24V	For digital I/O
4	- UMOT	GND / Motor voltage >24V to 48V
5	- UMOT	GND / Motor voltage >24V to 48V
6	0V (+24V)	GND for digital I/O

5.9 Joystick Connection (ST1, 9pol D-Sub-plug)

Pin	Signal	Remarks
1	GND	Ground
2	Axis 4	Analogue Input for 4th axis
3	Axis 1	Analogue Input for 1st axis
4	Axis 2	Analogue Input for 2nd axis
5	Axis 3	Analogue Input for 3rd axis
6	Snap-Shot	Snap-Shot Input parallel to Pin22/25pol or Pin28/50pol of MFP
7	/Stop	Stop-Input parallel to Pin23/25pol, or Pin29/50pol of MFP
8	+5V analogue	5V analogue reference voltage
9	+5V analogue	5V analogue reference voltage

5.10 Limit Switch Inputs (ST5, 16pol Pin Header with 15pol D-Sub-connection, for separate connection of the limit switches)

Pin	Assignment	Pin	Assignment
1	Zero point switch axis 1	9	+5V
2	Final position switch axis 1	10	+5V
3	Zero point switch axis 2	11	+12V
4	Final position switch axis 2	12	+12V
5	Zero point switch axis 3	13	GND
6	Final position switch axis 3	14	GND
7	Zero point switch axis 4	15	GND
8	Final position switch axis 4	Housing	Screen

5.11 Analogue I/O: (ST 7, 10-pol Pin header with 9pol D-Sub-connection)

Pin	Assignment	Note
1	Analogue IN Channel 6	Analogue: 0...5V, 4.7kOhm against +5V, RC-Filter 10KOhm/100nF (=St11,Pin8)
2	Analogue IN Channel 7	Analogue: 0...5V, 4.7kOhm against +5V, RC-Filter 10KOhm/100nF (=St11,Pin20)
3	Analogue IN Channel 8	Analogue: 0...5V, 4,7kOhm against +5V, RC-Filter 10KOhm/100nF (=St11, Pin7)
4	Analogue IN Channel 9	Analogue: 0...5V, 4.7kOhm against +5V, RC-Filter 10KOhm/100nF (=St11,Pin19)
5	Analogue Out Channel 0	0...10V or +/-10V, R,Last >=1kOhm Ri= about 100 Ohm (=St11,Pin18)
6, 7	PT 100 Channel 10 Connection temperature sensor	Measuring current = 10 mA, LB 5 has to be closed, St11, Pin6 is not applicable)
8	GND	Ground
9	VAREF = +5V / 1A	Output
10	Analogue OUT Channel 1	0…10V or +/- 10V

5.12 TTL Transmitter Inputs: (St6, 16-pol Pin header D-Sub-method of counting)

Pin	Notation	Function
1	A 1	Incremental transmitter 1, track A
2	B 1	Incremental transmitter 1, track B
3	REF 1	Incremental transmitter 1, track Z (Reference signal)
4	A 2	Incremental transmitter 2, track A
5	B 2	Incremental transmitter 2, track B
6	REF 2	Incremental transmitter 2, track Z (Reference signal)
7	A 3	Incremental transmitter 3, track A
8	B 3	Incremental transmitter 3, track B
9	REF 3	Incremental transmitter 3, track Z (Reference signal)
10	A 4	Incremental transmitter 4, track A
11	B 4	Incremental transmitter 4, track B
12	REF 4	Incremental transmitter 4, track Z (Reference signal)
13	GND	Ground
14	+5V	
15	+12V	
16	nc	

5.13 Convertor for TTL-Transmitter Inputs (16pol pin header on 3(4) x 9pol port)

Pin	9pol port 1 - 4	Function / Comment
1	Axis 1, Pin 6	A1 / Incremental transmitter 1, track A
2	Axis 1, Pin 8	B1 / Incremental transmitter 1, track B
3	Axis 1, Pin 9	REF1 / Incremental transmitter 1, track Z (Reference signal)
4	Axis 2, Pin 6	A2 / Incremental transmitter 2, track A
5	Axis 2, Pin 8	B2 / Incremental transmitter 2, track B
6	Axis 2, Pin 9	REF2 / Incremental transmitter 2, track Z (Reference signal)
7	Axis 3, Pin 6	A3 / Incremental transmitter 3, track A
8	Axis 3, Pin 8	B3 / Incremental transmitter 3, track B
9	Axis 3, Pin 9	REF3 / Incremental transmitter 3, track Z (Reference signal)
10	Axis 4, Pin 6	A4 / Incremental transmitter 4, track A
11	Axis 4, Pin 8	B4 / Incremental transmitter 4, track B
12	Axis 4, Pin 9	REF4 / Incremental transmitter 4, track Z (Reference signal)
13	Axis 1 - 4, Pin 2	GND / Ground
14	Axis 1 - 4, Pin 7	+5V
15	Axis 1 - 4, Pin 4	+12V
16		n.c.

5.14 Motor Plug Axis 1 – 3 with Limit Switch (ST2 25pol Dsub port)

Pin	Assignment
1	Axis 1, Sinus +
2	Axis 1, Sinus -
3	Axis 1, Cos +
4	Axis 1, Cos -
5	Axis 2, Sinus +
6	Axis 2, Sinus -
7	Axis 2, Cos +
8	Axis 2, Cos -
9	Zero point switch axis 1
10	Final position switch axis 1
11	Motor brake A+(X)
12	Motor brake A- (Y)
13	Motor brake B+ (Z)
14	Axis 3, Sinus +
15	Axis 3, Sinus -
16	Axis 3, Cos +
17	Axis 3, Cos -
18	Zero point switch axis 2
19	Final position switch axis 2
20	Zero point switch axis 3
21	Final position switch axis 3
22	+5V
23	+12V
24	GND
25	GND
Housing	Screen

5.15 Motor Plug Axis 4 (ST1 or ST2 on the option card)

15pol DSub port ST 1	5pol DSub port ST2 (up to 10A)	Function
1	2	Sin +
2	1	Sin -
3	5	Cos +
4	4	Cos -
5		Final position switch
6		Zero point switch
7		+ 5V
8		GND
9		Sin +
10		Sin -
11		Cos +
12		Cos -
13		Motor brake B+ (Z)
14		Motor brake B - (A)
15		+ 12V

5.16 Digital I/O (ST11 40pol Pin Header Dsub-method of counting)

Pin	Assignment	Pin	Assignment
1	Output 1	20	Input 1
2	Output 2	21	Input 2
3	Output 3	22	Input 3
4	Output 4	23	Input 4
5	Output 5	24	Input 5
6	Output 6	25	Input 6
7	Output 7	26	Input 7
8	Output 8	27	Input 8
9	Output 9	28	Input 9
10	Output 10	29	Input 10
11	Output 11	30	Input 11
12	Output 12	31	Input 12
13	Output 13	32	Input 13
14	Output 14	33	Input 14
15	Output 15	34	Input 15
16	Output 16	35	Input 16
17	GND	36	GND
18	GND	37	+24V
19	GND	38 - 40	+24V

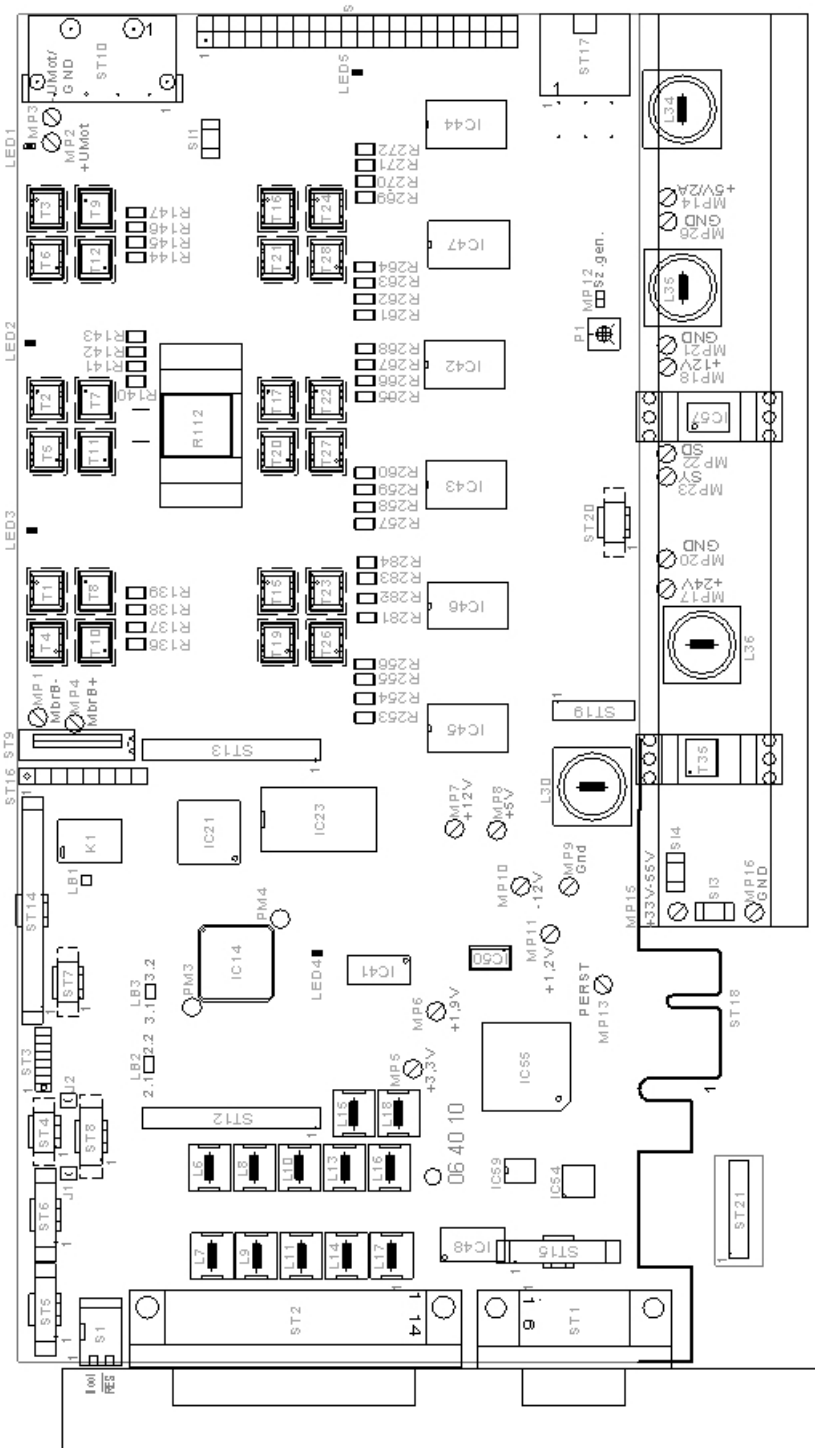
5.17 Technical Data

General	
Max. motor drive:	70 R/sec. at 200 motor steps
Max. motor current: axis 1…3 Max. motor current axis 4 (Mixed assembly is possible)	0.5A; 1.25A; 2.5A; 3.75A; 5A 0.5A; 1.25A; 2.5A; 3.75A; 5A; max. 10A
Step resolution	max. 1.600.000 steps/drive at 200 motor steps
Baud rate:	Up to 921.600 Kbd
Max. counting frequency for TTL-transmitter Inputs	16 Mflanks = 4 MHz for Quep 1 + 2 13 Mflanks = 3.25 MHz for Quep 3 to 6
Max. motor drive:	70 R/sec. at 200 motor steps

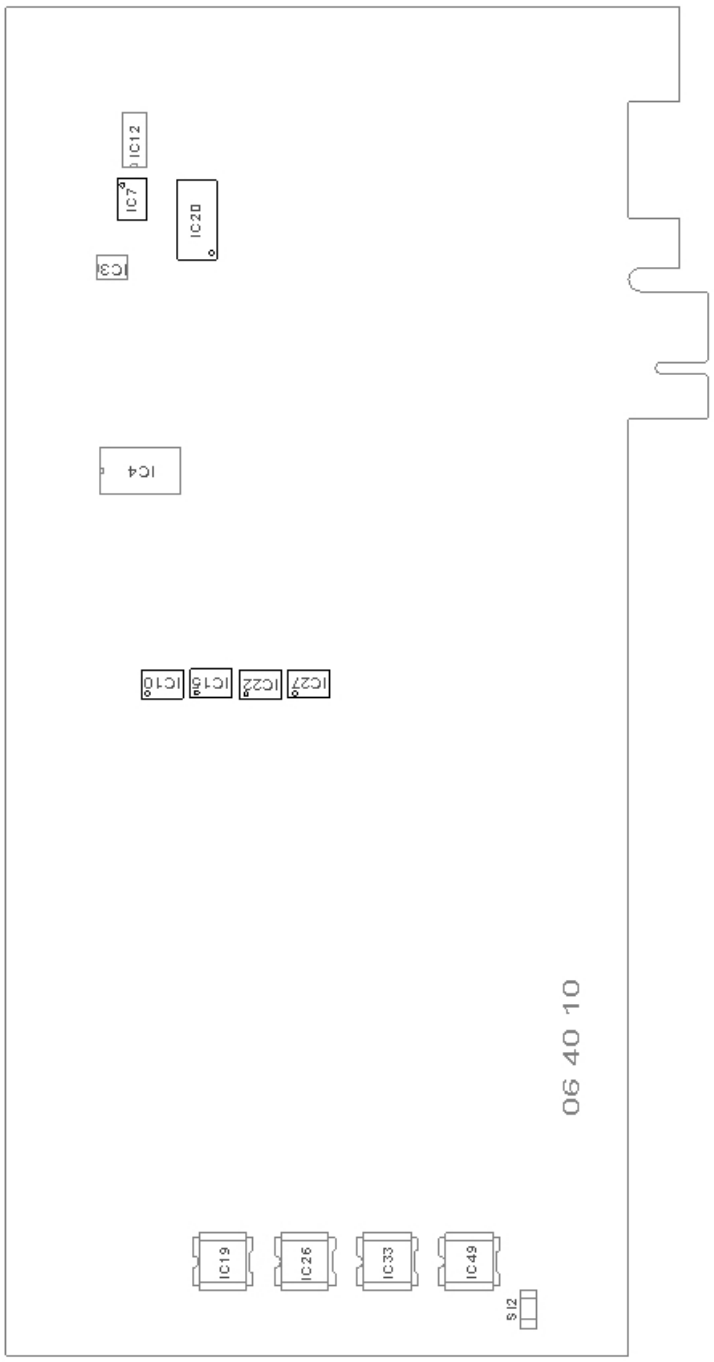
LSTEP-PCIexpress	
Power supply:	Logic voltage: PCIexpress-Slot from PC and 4pol power supply plug (12V) Motor voltage: 2V from PC-power supply pack or 24 …48V over ext. power supply pack
Motor voltage:	12V…48V
Dimensions L x H x W (1 Slot)	214mm x 107 x 15mm (1 Slot)

LSTEP-express	
Mains connection:	90…264V (intern 48V)
Fuses: - primary (in Euro port): - secondary (on the platine):	 • 2 A slow / 1 A slow LSTEP-1 and LSTEP-2 • 5 A slow / 2.25 A slow LSTEP-3 • Si1 LSTEP-1 and 2 ➔ 5 A slow / LSTEP-3 ➔10A slow
Max. power down time:	< 50ms at power failure (<0,77 * U_N) the LSTEP switches to Reset
Motor voltage:	48V
Ambient conditions: Air temperature at operation: Air temperature out of operation: Humidity at operation Humidity out of operation	 15 ... 40 degree C 0 ... 43 degree C 8 ... 80 % at 31° / Maximal 50% at 40° 0 ... 80 %
Dimensions L * W * H (without carry handle): - Standard housing: - 19“ table housing	 270 mm • 230 mm • 100 mm at LSTEP-1x 482.6 mm (19“) • 375.5 mm • 90 mm (2HE) at LSTEP-3x
Weight: - Standard housing: - 19“ table housing	 3.4 kg 5 kg

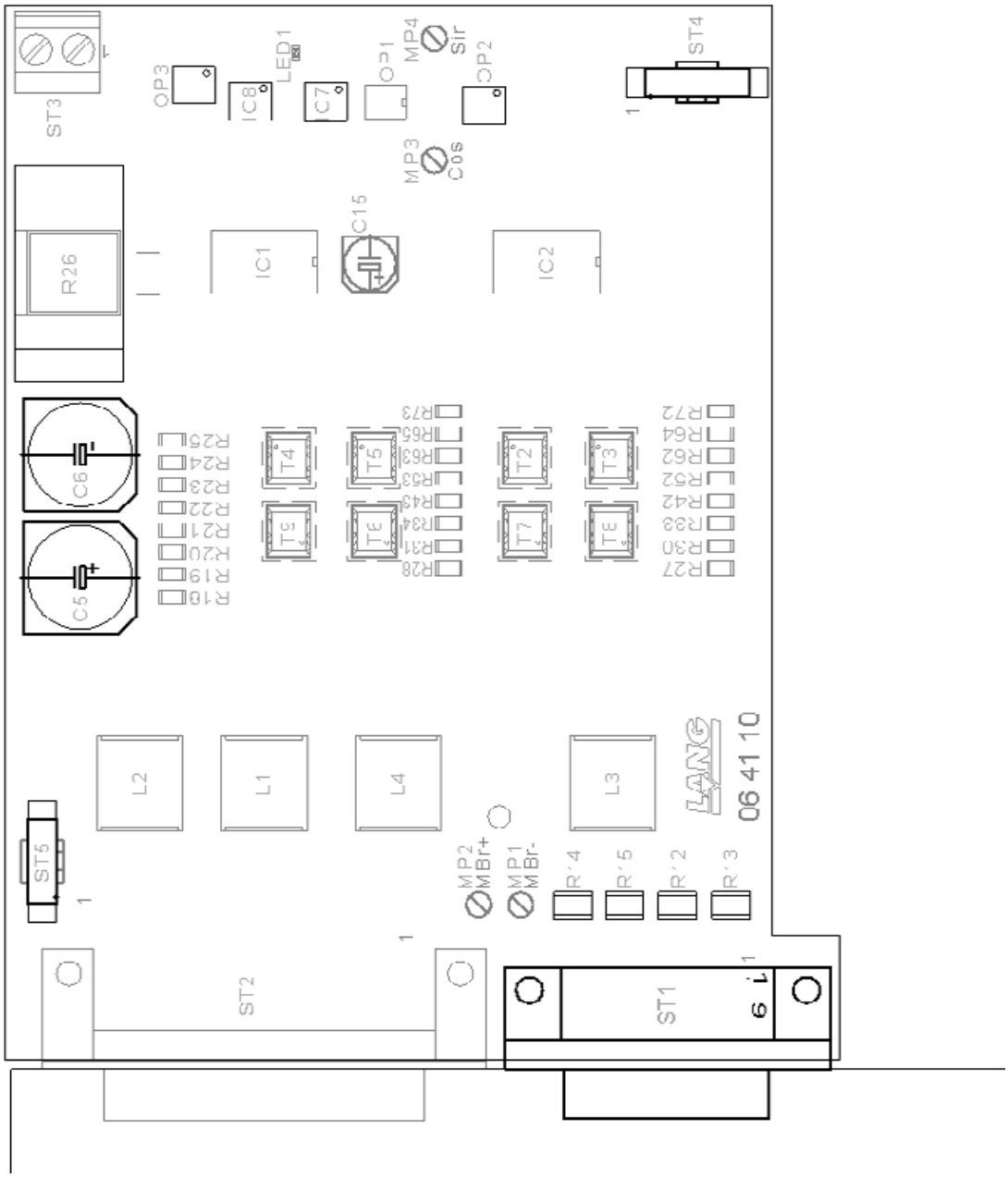
Assembly Diagram of LSTEPexpress / LSTEP-PCIexpress



Assembly Diagram of LSTEPexpress / LSTEP-PCIexpress (back side)



Assembly Diagram 4th Axis (Option card)



6 Appendix LSTEP-API

6.1 Introduction

The LSTEP-API (programming interface for the LStep precision positioning systems) is intended to assist all software developers to quickly and effectively develop applications with the controllers of the LSTEP family, without having to deal with hardware-related programming. It offers access to the complete command set of the LSTEP positioning systems.
For new developments, only Lstep4XDLL should now be used, as it permits parallel control of multiple LSTEPs. Lstep4DLL is no longer available for applications in LabVIEW.

6.1.1 IncludedFunctions

- Windows 32-bit DLL
- Support of the stepping motor controllers LSTEP xx, LSTEP xx/2, LSTEP-PC, ECO-STEP, LSTEP-44, LSTEP-PCI, LSTEP-PCIexpress, and LSTEPexpress
- Activation via RS232, USB, ISA, PCI (DPRAM), or PCIexpress
- Automatic recognition of the connected controller
- Configuration of the controller
- Execution of all commands supported by the controller
- Up to 4 axes
- Multithreading capable

6.1.2 System requirements

With LSTEP-API, as well as with LSTEP4X-API it is possible to develope applications with MS Windows 9x, Windows NT, Windows 2000, Windows Vista, Windows XP, Windows 7.

6.1.3 Supported Development Environments

The LSTEP-API and LSTEP4X-API has been tested with the following development and runtime environments

Borland/Inprise Delphi 3-5

Microsoft Visual C++ 6.0

National Instruments LabVIEW

It should be compatible with all other programming environments which can use DLLs.

(DLL = Dynamic Link Library; A DLL is an executable module which contains code and resources which are used by other applications or DLLs.)

6.2 DLL Interface

6.2.1 LSTEP-API

The main component of the LSTEP-APIs is the file LSTEP4.DLL. You use this DLL for developing your own programs, to configure the LSTEP, to transmit commands, to inquire position values, inputs/outputs, etc. **Please use only Lstep4XDLL for new developments.**

6.2.2 LSTEP4X-API

Main component of the LStep4X-APIs is the file LSTEP4X.DLL. Use this DLL for developing your programs, to configure one or multiple LSTEPS, to send commands, to query position values, inputs/outputs, etc.

6.2.3 General Information

6.2.3.1 LSTEP 4.DLL

The DLL LSTEP4.DLL implements the commands of the LSTEP-API. All functions are declared with a 32 bit integer as the return value. A return value of 0 indicates error-free execution of the function, if errors (e.g. timeouts) occur, the relevant error code (see table) is returned.

For functions such as LS_MoveAbs, values are always transmitted for 4 axes. If the controller has only 1-3 axes, the values for the non-existing axes are ignored and can be set to 0.

6.2.3.2 LSTEP4X.DLL

The DLL LSTEP4X.DLL implements the commands of the LSTEP4X-API. All functions are declared with a 32 bit integer as the return value. A return value of 0 indicates error-free execution of the function, if errors (e.g. timeouts) occur, the relevant error code (see table) is returned.

The first parameter send for all functions of the API is an integer value (between 1 and 32), which indicates the number of the LStep, where the command is supposed to be send to.

The function LSX_CreateLSID can be used, to send such an ID-value. With a call of LSX_FreeLSID an ID-value is set free again. (see Delphi-example)

For functions such as LSX_MoveAbs, values are always transmitted for 4 axes. If the controller has only 1-3 axes, the values for the non-existing axes are ignored and can be set to 0.

6.2.3.3 Difference in comparence with LSTEP4.DLL

The function circumference LSTEP4X.DLL has not changed in compare with LSTEP4.DLL, the function names are identical. The normal LStep API (LSTEP4.DLL) is continued, existing source code, which the LStep API uses must not be modified.

LSTEP4X API opens a protocol window for each LStep, and the Log-files are also written separately for each LStep.

Because the LSTEP4X supports API Multi-Threading, programs made by the customer can access the LSteps from several Threads through the API.

The parallel control of several LSTEP motor controls is possible.

The function names received different prefixes‘. For the LSTEP4X.DLL „LSX_“ is used instead „LS_“ like it is used for the LSTEP4.DLL.

A integer value is used as an additional parameter for all function calls of the LSTEP4X API which identifies the controller. The numbering of the LSTEPs starts with 1 to 32.

Information:	Under Windows NT the supplied driver GIVEIO must be installed so that the DPRAM interfaces of the LSTEP-PCs may be used.

6.2.4 Integration in Delphi

6.2.4.1 LSTEP4-API

All function names of the LSTEP4-API start with “LS_” for easier differentiation. To be able to use the functions of the LSTEP4 APIs, LSTEP4.pas must be present in the uses-clause of the unit it question and must be in one of the pre-set search paths.

Required files: LSTEP4.DLL and LSTEP4.pas

Delphi-example for the control of an LStep

```
...
var LStep1: Integer;
…
begin

LSX_ConnectSimple(1, 'COM1', 9600, True);

LSX_MoveAbs(10.0, 20.0, 30.0, 0.0, True);

LSX_Disconnect();

end;
```

6.2.4.2 LSTEP4X API

All function names of the LSTEP4X-API start with “LSX_” for easier differentiation. To be able to use the functions of the LSTEP4X APIs, LSTEP4X.pas must be present in the uses-clause of the unit it question and must be in one of the pre-set search paths.

Required files: LSTEP4X.DLL and LSTEP4X.pas

Delphi-example for the parallel control of 2 LSTEPs

```
...
var LStep1, LStep2: Integer;
…
begin
LSX_CreateLSID(LStep1);
LSX_CreateLSID(LStep2);

LSX_ConnectSimple(LStep1, 1, 'COM1', 9600, True);
LSX_ConnectSimple(LStep2, 1, 'COM2', 9600, True);

LSX_MoveAbs(LStep1, 10.0, 20.0, 30.0, 0.0, True);
LSX_MoveAbs(LStep2, 5.0, 10.0, 0.0, 0.0, True);

LSX_Disconnect(LStep1);
LSX_Disconnect(LStep2);

LSX_FreeLSID(LStep1);
LSX_FreeLSID(LStep2);

end;
```

6.2.5 Integration in Visual C++

6.2.5.1 LSTEP4-API

For Visual C++ , an encapsulation of the LSTEP4.DLL has been created. The class CLStep4 loads the DLL and all pointers in response to function calls dynamically. The methods of the LSTEP-object is not preceded by “LS_” .
(Example: LS.Calibrate() instead of LS_Calibrate)
Only one instance should be created by the class CLStep4 , since at present, no more than one LStep can be controlled simultaneously with the LSTEP .

Required files: LSTEP4.DLL, LSTEP4.h and LSTEP4.cpp

Visual C++- example for the control of a LStep

```
...
CLStep4 LS1;
…

LS1.ConnectSimple(1, "COM1", 9600, true);

LS1.MoveAbs(10.0, 20.0, 30.0, 0.0, true);

LS1.Disconnect();
delete LS1;
```

6.2.5.2 LSTEP4X-API

For Visual C++, an encapsulation of the LSTEP4X.DLL has been created. The class CLStep4X loads the DLL and all pointers in response to function calls dynamically. The methods of the LSTEP-object are not preceded by "LSX_".
(Example: LSX.Calibrate() instead of LSX_Calibrate)

With C++ the functions LSX_CreateLSID and LSX_FreeLSID must not be called for using the LSTEP4X.DLL, because the Wrapper-Klasse CLStep4X administers itself the integer value, which indicates the number of the LStep. The method of CLStep4X has no additional parameter for the number of the LStep.

Required files: LSTEP4X.DLL, LSTEP4X.h and LSTEP4X.cpp

Visual C++- example for the parallel control of 2 LSteps

```
...
CLStep4X* LS1,* LS2;
…

LS1 = new CLStep4X;
LS2 = new CLStep4X;

LS1->ConnectSimple(1, "COM1", 9600, true);
LS2->ConnectSimple(1, "COM2", 9600, true);

LS1->MoveAbs(10.0, 20.0, 30.0, 0.0, true);
LS2->MoveAbs(5.0, 10.0, 0.0, 0.0, true);

LS1->Disconnect();
delete LS1;
LS2->Disconnect();
delete LS2;
```

6.2.6 Integration in LabVIEW

NI LabVIEW is a developing environment based on the graphic programming language G. It enables programming with graphic symbols to be done quickly and easily. Complicated 32-bit programs can be created, thus ensuring that the required speed of execution for control, test and measuring applications is given.

All LabVIEW programs (so-called VIs, Virtual Instruments) have a front panel and a block diagram and can in turn be integrated into other programs as a sub-program (SubVI).

A VI library (LSTEP4.LLB resp. LSTEP4X.LLB) has been created for embedding LSTEP-API (LSTEP4.DLL and LSTEP4X.DLL) which contains approx. 110 VIs. These individual VIs (e.g. LS4 ConnectSimple.vi) encapsulate the respective LSTEP API functions. The LSTEP4.dll is used by means of the "Call Library Function" (calling ext. libraries).

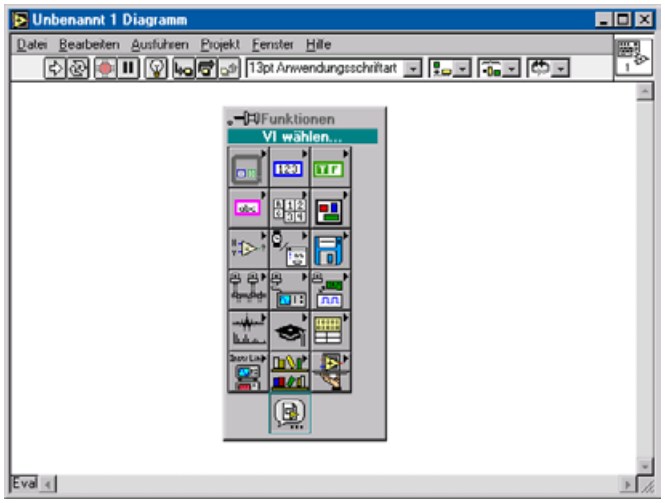
6.2.6.1 Differences between LSTEP4.LLB and LSTEP4X.LLB

In new software-projects with the LSTEP-API under LabView, you should use exclusively the VI-Library LSTEP4X.LLB. This has the following reason: Die LSTEP4.LLB only supports one, besides that all VI's have as a return value only a Boolesche variable, therefore no mistake code to be interpreted.

In the (newer) LSTEP4X.LLB the VI's have as a return value a 32-bit-Integer-value (which is named „error out"). If this equals 0, it signalizes that the LSTEP-API function was executed without errors. Otherwise -refer to the the table in this documentation for the meaning of the error code. Several LSteps can be parallel controlled via the LSTEP4X.DLL called in the VI's. The VI's for the LSTEP4X.DLL differ in the file name from the once of the LSTEP4.DLL through the shorthand symbol at the beginning it is „LS4X" instead „LS4". In LabView the VI's for the LSTEP4X.DLL purple, the once of the LSTEP4.DLL has the background colour blue. The designation of the VI's in the symbols is the same. In all VI's an additional connection comes is added. Which is a 32-bit-Integer-value and indicates the number of the LStep that the command, for example a moving command or the reading out of the current position, refers to („LStep Controller ID"). You can assign those numbers yourself (e.g. „0" for the LStep at the serial interfaces COM1, „1" for the LStep at COM2 etc.), or via the VI „LS4X CreateLSID". ID-numbers created with the VI „LS4X FreeLSID" can be released again. If you use only one LStep in your LabView-project, you can leave the connection „LStep Controller ID" open for all used VI's from the LSTEP4X.LLB, because it has the default-value 1 as a standard needed files in LabView: LSTEP4X.DLL and LSTEP4X.LLB resp. LSTEP4.DLL and LSTEP4.LLB

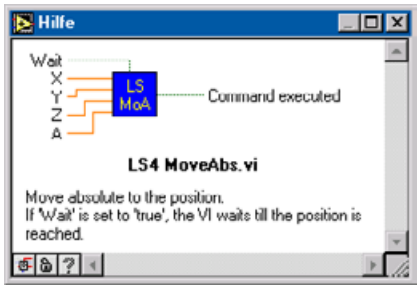
6.2.6.2 Procedure for using an LSTEP4 VIs

1. Create a new VI
2. Switch to the block diagram window (Ctrl+E)
3. Click on the diagram (right mouse button)

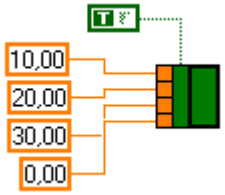


4. Select VI ...
5. Open the supplied VI Library LSTEP4.llb in the file dialog box, and then select the required command (e.g. LS4 MoveAbs.vi)
6. Place VI in the diagram

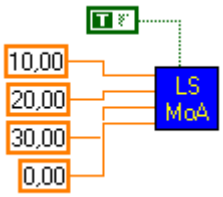
Press Ctrl+H to open a help window, which gives you information about the VI at which the mouse pointer is currently located



The transmission of parameters to SubVIs is done by terminals, which have to be "cabled". To display these terminals in the diagram, click the right mouse button on the VI and select "Display/Terminals". You can then allocate values/sources to the terminals. There are several ways of doing this, one of them is: Click the right mouse button on the required terminal then on the menu item "produce constants".



In this example and absolute travel command (X 10mm, Y 20mm, Z 30mm) is executed.

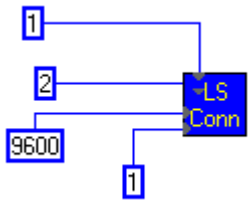


For details of the occupancy of the terminals of the Vis, please refer to the documentation for the API function in question. There, you will find a diagram, which also appears in the Help window of LabVIEW. The parameters are more or less the same as those of the DLL-function: There are only certain differences for functions to which bit masks are transmitted as the parameters (e.g. LS4 SetActiveAxes.vi)

The LSTEP4 VIs has a terminal called "Command executed". If this logical value is "true", the command was executed successfully. If an error has occurred, the value is set to "false".

Before travel commands can be executed or position values can be read out, etc., the connection to LSTEP must be opened. This is easiest with VI “LS4 ConnectSimple.vi”. It initialises the interface and detects the LSTEP, which is connected.

Example for RS232 (COM2 and 9600 Baud):

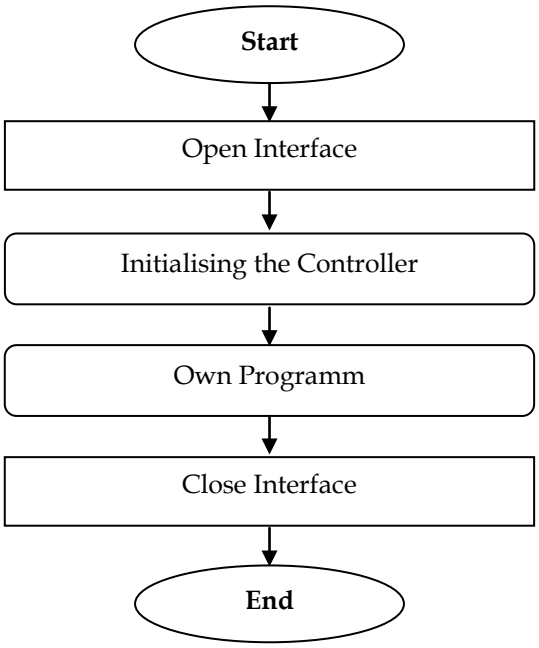


6.3 Notations to create own programs for programming the controller via the API

The following charts show the program flow diagram after that the programs for controlling positioning systems should be made. The used functions are listed in the LSTEP-API description and there they are described more detailed.

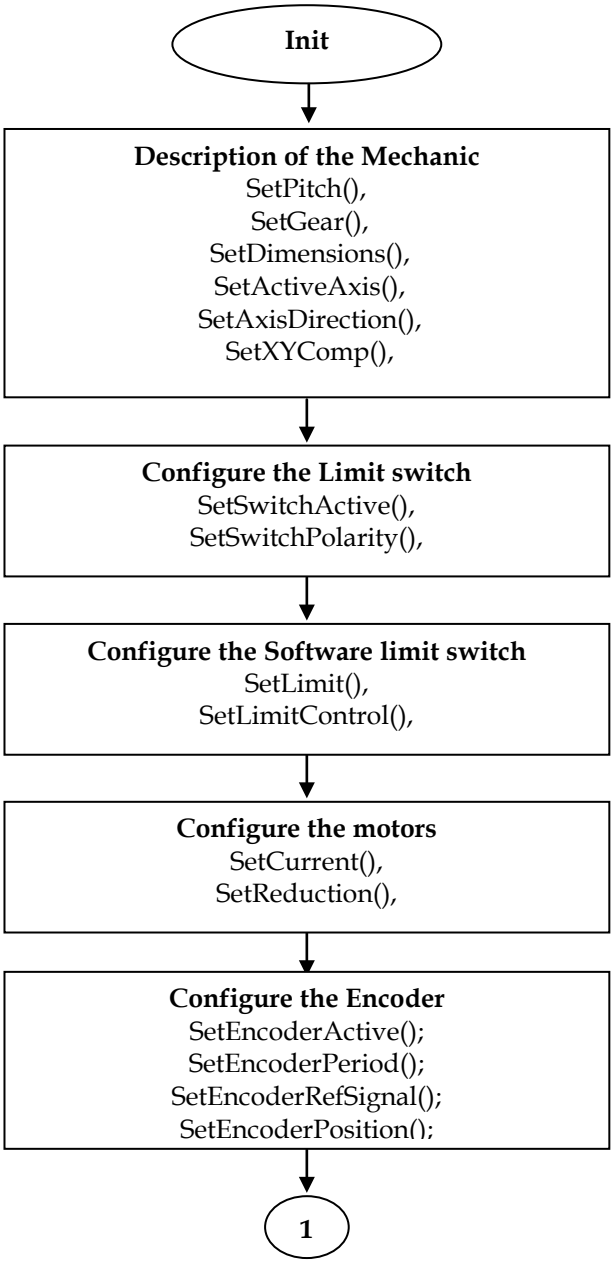
Because the pre-configured default-settings can not contain all data for every application, after the used interface was opened, follow the steps described under item „Initialising the Controller“.

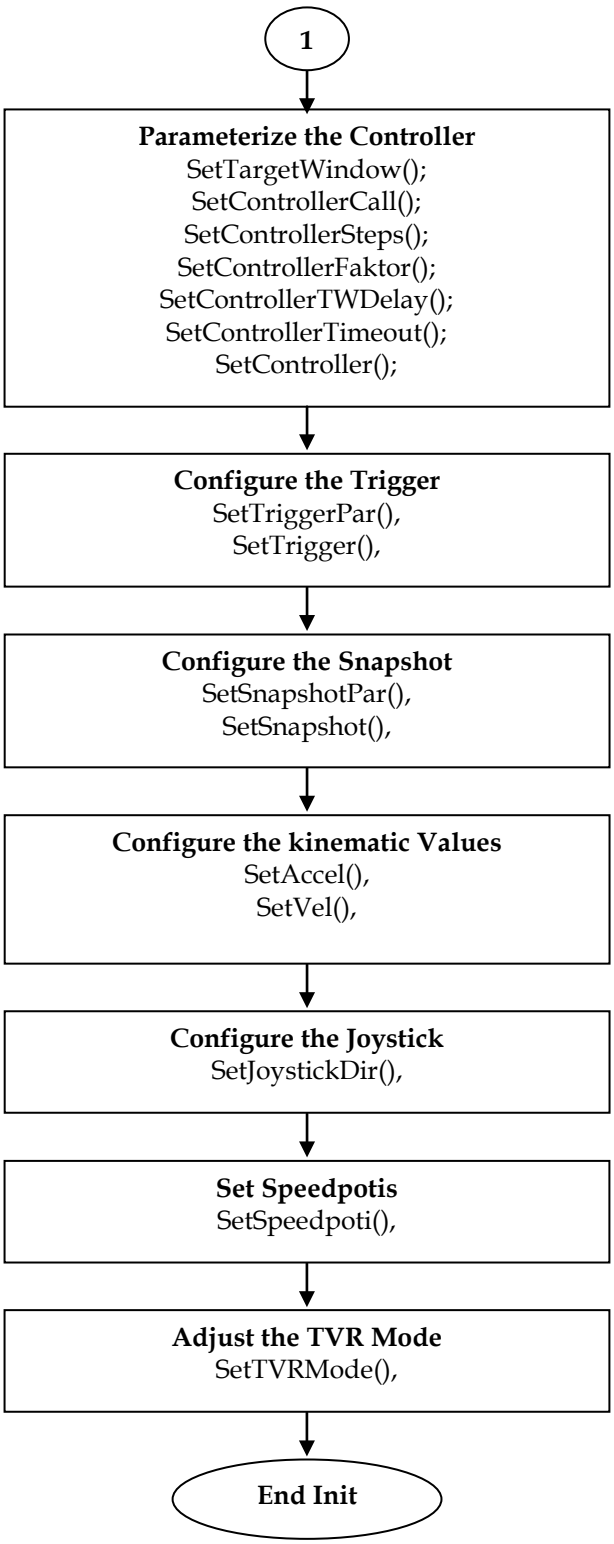
Subsequently under the use of LSTEP-API any user program can be written.



6.3.1 Initialising the Controller

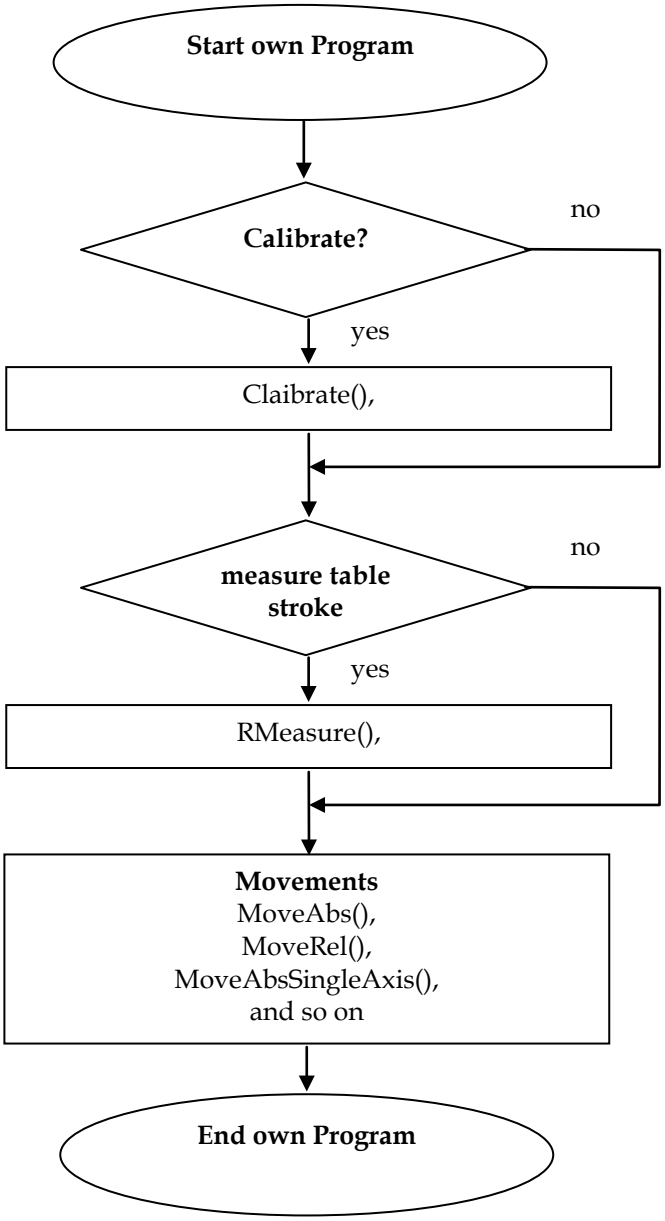
In order to have a fault free operation it is necessary to carry out the basic settings of the positioning controller prior to the start of the own program when initialising the used LSTEP.




1
Parameterize the Controller
SetTargetWindow();
SetControllerCall();
SetControllerSteps();
SetControllerFaktor();
SetControllerTWDelay();
SetControllerTimeout();
SetController();
Configure the Trigger
SetTriggerPar(),
SetTrigger(),
Configure the Snapshot
SetSnapshotPar(),
SetSnapshot(),
Configure the kinematic Values
SetAccel(),
SetVel(),
Configure the Joystick
SetJoystickDir(),
Set Speedpotis
SetSpeedpoti(),
Adjust the TVR Mode
SetTVRMode(),
End Init

6.3.2 Own Program part

In the own program part the user can program the desired functionality of the controller. Such as carrying out positioning movements in dependence of the I/O condition, as well as setting the trigger signal in dependence of the positions etc.



6.4 Functions

6.4.1 Index 1(Brief Description for API Commands)

Arrangement of the commands as follows:

Attention!

- **Only the DLL calls are described in this brief description.**
- **All new commands must be transmitted with the DLL call "SendString"**
- **You will find a complete list of all commands in Index 2 at the end of this chapter.**

API-Configuration/Interface

Controller-Info

Command	Short description	Page
GetSerialNr	Read serial number of the controller	36
GetVersionStr	Gives the current Firmware version number	36
GetVersionStrDet	Read out detailed version number of Firmware	36
GetVersionStrInfo	Gives detailed information about version number	37

Settings

Command	Short description	Page
ConfigMaxAxes	Set the number of axes used	38
GetAccel	Inquiry of acceleration	38
GetAccelJerk	Ask jerk during acceleration	39
GetActiveAxes	Delivers enable axes	40
GetAxisDirection	Inquiry of reverse-turning direction	41
GetCaliboffset	Inquiry of calibration-offset	42
GetCalibrateDir	Inquiry reverse preceding sign when calibrating	43
GetCalibRMAccel	Ask acceleration during calibration procedure	44
GetCalibRMBackSpeed	Reads the speed, with which the axis are moved back during calibration	45
GetCalibRMJerk	Ask jerk during calibration procedure	46
GetCalibRMVel	Ask positioning speed during calibration procedure	48
GetCurrentDelay	Indicates time delay for current reduction	48
GetDeceleration	Ask for delays	49
GetDecelJerk	Ask jerk during delay	50
GetDimensions	Inquiry dimensions of the axes	52
GetGearDenominator	Ask denominator of gear ratio	53
GetGearNumerator	Ask counter of gear ratio	54
GetJoystickFilter	Indicates, if the filtering and hysteresis is activated in joystick operation	55
GetMotorCurrent	Inquiry motor current	56
GetMotorFieldDir	Query rotating direction of the motor	57
GetMotorTablePatch	Indicates, if the correction table is activated.	58
GetPitch	delivers spindle pitch	59
GetPowerAmplifier	Power amplifier On/Off (LS44 only)	60
GetReduction	Inquiry of current reduction	61
GetRefSpeed	Reads the reverse speed, the axes move while searching the reference mark.	62
GetRMOffset	Inquiry RM-Offset	63
GetSpeedPoti	Indicates if the potentiometer On/Off	64
GetStopDecel	Stop input query deceleration	65
GetStopDecelJerk	Ask jerk during delay of system in case of Emergency Stop signal	66
GetStopPolarity	Read stop input polarity.	67
GetVel	Inquiry speed	68
GetVLevel	Delivers the speed limits of the indicated speed range	69
GetXYAxisComp	Inquiry XY-axis overlay	71
LstepSave	Save current configuration in LStep (EEPROM)	71
SetAccel	Set acceleration	39
SetAccelJerk	Adjust jerk during acceleration	40
SetAccelSingleAxis	Set acceleration	72
SetActiveAxes	Enable axes	41
SetAxisDirection	Reverse turning direction	42

Status request

Command	Short description	Page
GetError	shows the current error number	73
GetSecurityErr	Reads all statuses and results of the GAL-safety monitoring (only with LS44-controller)	74
GetSecurityStatus	Delivers the current statuses the safety monitoring (only with LS44-controller)	75
GetStatus	shows the current condition of the controller	76
GetStatusAxis	Gives the present status of the individual axes	76
GetStatusLimit	Delivers the current condition of the software-limits of each axis.	77
SetAutoStatus	AutoStatus On/Off	77

Moving commands and Position administration

Command	Short description	Page
Calibrate	Calibrate	78
CalibrateEx	Only the axes are calibrated, whose corresponding bit was set in the transmitted integer-value.	78
Clearpos	Sets the position values to 0 (for endless turning axes)	79
GetDelay	Reads the delay of the vector start.	79
GetDistance	Delivers the distance for LS_MoveRelShort	80
GetInputTrigMove	Supplies the configuration of Pin1 on the MFP	81
GetPos	Inquires the current positions of all axes	82
GetPosEx	Inquires the current encoder or position values of all axes	83
GetPosSingleAxis	Inquire the current position of an axis	83
MoveAbs	Move to absolute position	84
MoveAbsSingleAxis	Move individual axis to absolute position	84
MoveEx	extended moving command	85
MoveRel	Move relative vector	86
MoveRelShort	Move to relative position (short command)	86
MoveRelSingleAxis	Move individual axis relatively	87
RMeasure	Measure Table Stroke	87
RmeasureEx	Measure table stroke (The table stroke is only measured for axes for which the relevant bit has been set in the transmitted integer value).	88
SetDelay	The delay command is used to produce a vector start delay	80
SetDistance	Set distance (for LS_MoveRelShort)	81
SetInputTrigMove	Configurate the Pin 1 on the MFP	82
SetPos	Set positional values	88
StopAxes	Stop (all movements are stopped)	89
WaitForAxisStop	The function returns, as soon as the selected axes in the bit-mask AFlags reached its goal position.	89

Joystick and Handwheel

Command	Short description	Page
GetDigJoySpeed	Read out the set speeds	90
GetHandwheel	Read hand wheel condition	91
GetJoyChangeAxis	Read Joystick allocation of the axes	96
GetJoystick	Reads the delay of the vector start.	91
GetJoystickAxes	Shows, for which axes Joystick is switched on	92
GetJoystickDir	Joystick direction	93
GetJoyVel	Ask for the max. positioning speeds in Joystick operation	94
GetJoystickWindow	Read Joystick-window	95
JoyChangeAxis	sets allocation of axes of Joystick	96
SetDigJoySpeed	Read Digital joystick and speed .	90
SetDigJoyOff	Switches off digital Joystick	96
SetHandwheelOff	Handwheel Off	97
SetHandwheelOn	Handwheel On	97
SetJoystickAxes	Switches Joystick for specified axes	92
SetJoystickDir	Joystick direction	94
SetJoystickOff	Analogue joystick Off	97
SetJoystickOn	Analogue joystick On	98
SetJoyVel	Adjust the max. positioning speed in Joystick operation	95
SetJoystickWindow	Set joystick window	95

Control panel with Trackball and Joyspeed-keys

Command	Short description	Page
GetBPZ	Reads the condition of the additional control panel with track ball	99
GetBPZJoyspeed	Control panel joystick-speed	100
GetBPZTrackballBackLash	Read out control panel track ball-back lash	100
GetBPZTrackballFactor	Read out control panel trackball-factor	101
SetBPZ	Control panel On/ Off	99
SetBPZJoyspeed	Control panel joystick-speed	100
SetBPZTrackballBackLash	Control panel trackball-reverse backlash	101
SetBPZTrackballFactor	Control panel trackball-factor	101

Limit switch (Hardware a. Software)

Command	Short description	Page
GetAutoLimitAfterCalibRM	Indicates if the internal software limits will be set during calibration and table stroke measuring.	102
GetLimit	Set travel limits	103
GetLimitControl	Reads, if travel range monitoring is active	104
GetLimitControlMode	Supply the mode for monitoring of the Software limits.	105
GetSwitchActive	Read status of limit switch	106
GetSwitches	Reads the status of all limit switches	107
GetSwitchPolarity	Reads limit switch polarity	107
GetSwChange	Displays adjustments of limit switch	108
SetAutoLimitAfterCalibRM	Prevents that the internal software limits are set during calibration and table stroke measuring.	102
SetLimit	Set travel limits	103
SetLimitControl	Range monitoring	104
SetLimitControlMode	Place the mode for monitoring of the Software limits.	105
SetSwitchActive	Read status of limit switch On / Off	106
SetSwitchPolarity	Set limit switch polarity	108
SetSwChange	Change limit switch	109

Digital and analogue Input and Output

Command	Short description	Page
GetAnalogInput	Reads the current status of an analogue channel	110
GetAnalogInputs2	Reads the current status of the analogue channels PT100, MV and V24	110
GetDigitalInputs	Read all input pins	111
GetDigitalInputsE	read additional digital inputs (16-31)	111
SetAnalogOutput	Set analogue output	112
SetDigIO_Distance	Function of the digital inputs / outputs	112
SetDigIO_EmergencyStop	Function of the digital inputs / outputs Allocating of the Emergency Stop pin	113
SetDigIO_Off	"Off" function of the digital inputs/outputs	113
SetDigIO_Polarity	Set polarity	114
SetDigitalOutput	Set output pin	114
SetDigitalOutputs	Set digital outputs (0-15)	115
SetDigitalOutputsE	Set additional outputs (16-31)	115

Clock pulse Forward / Back

Command	Short description	Page
GetFactorTVR	Reads factor for clock pulse Forward/ Back	116
GetTVRMode	Read setup of clock pulse Forward /Back (= TVR Mode)	117
SetFactorTVR	Factor for clock pulse Forward/ Back	116
SetTVRMode	Set clock pulse Forward / Back	118

Clock pulse Forward /Back via Interface

Command	Short description	Page
SetTVRInPulse	Clock pulse Forward /Back via Interface	119

Clock pulse Forward / Back for the additional axes

Command	Short description	Page
GetAccelTVRO	Reads the set acceleration for the additional axes.	120
GetPosTVRO	Read position values of the additional axis	121
GetStatusTVRO	Delivers the current status of the additional axis	122
GetTVROutMode	Read settings of the additional axis	122
GetTVROutPitch	Reads the spindle pitch of the additional axis	123
GetTVROutResolution	Reads the resolution of the power amplifier which is to be controlled	124
GetVelTVRO	Reads the set speed of the additional axis	125
MoveAbsTVROSingleAxis	Move individual axis to absolute position	126
MoveAbsTVRO	Move to absolute position	127
MoveRelTVROSingleAxis	Move single axis absolute	127
MoveRelTVRO	Move relative vector	128
SetAccelSingleAxisTVRO	Acceleration of single additional axis	128
SetAccelTVRO	Set acceleration	120
SetPosTVRO	Set positional values	121
SetTVROutMode	Set clock pulse Forward / Back	123
SetTVROutPitch	Set spindle pitch	124
SetTVROutResolution	Sets the resolution of the power amplifier which is to be controlled	125
SetVelSingleAxisTVRO	set speed of the additional axis	129
SetVelTVRO	set speed of the additional axis	126

Encoder-Settings

Command	Short description	Page
ClearEncoder	Set encoder-counter to zero	130
GetEncoder	Reads all encoder positions	130
GetEncoderActive	Reads, which encoders are activated after the calibration.	131
GetEncoderMask	Read encoder statuses	132
GetEncoderPeriod	Read length of encoder period	133
GetEncoderPosition	Read encoder position setting	134
GetEncoderRefSignal	Reads if interpret reference signal from encoder when calibration is done	135
SetEncoderActive	This function is used to select which encoder is to be activated after calibration.	131
SetEncoderMask	(de-)activate encoder	132
SetEncoderPeriod	Set length of encoder period	133
SetEncoderPosition	Encoder position display On/Off	134
SetEncoderRefSignal	Interpret reference signal from encoder when calibration is done	135

Controller Setting

Trigger-Output

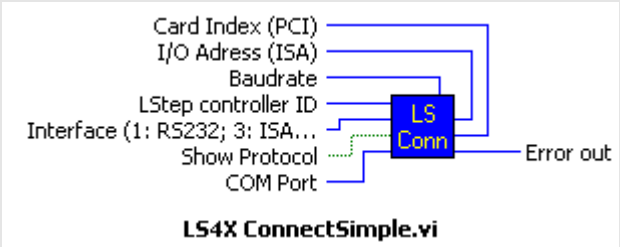
Snapshot-Input

6.4.2 Functions

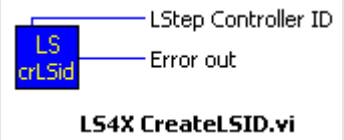
API-Configuration/Interface

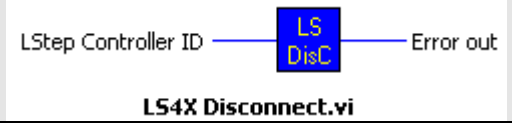
LS_Connect	
Description:	Connect with LSTEP Therefore the interfaces-parameters are used, which are loaded from the INI-file via LS_LoadConfig. (One of the functions LS_Connect, LS_ConnectSimple or LS_ConnectEx must be called for initialising of the interface, so that the communication with the LSTEP is possible.)
Delphi:	function LS_Connect: Integer; function LSX_Connect(LSID: Integer): Integer;
C++:	int Connect();
LabView:	
Parameter:	-
Example:	LS.LoadConfig("C:\LStepTest\LStep.INI"); LS.Connect();

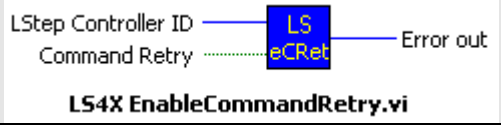
LS_ConnectEx	
Description:	Connect with LSTEP
	This function offers more possibilities; a pointer is transferred to a data structure that contains the interface parameter. In this record also informations about identified controller (version number...) are supplied (One of the functions LS_Connect, LS_ConnectSimple or LS_ConnectEx must be called for initialising of the interface, so that the communication with the LSTEP is possible.)
Delphi:	function LS_ConnectEx(var AControlInitPar: TLS_ControlInitPar): Integer; function LSX_ConnectEx(LSID: Integer; var AControlInitPar: TLS_ControlInitPar): Integer;
C++:	int ConnectEx (TLS_ControlInitPar *pAControlInitPar);
Parameter:	AControlInitPar: Pointer to a record of the type Typs TLS_ControlInitPar
Example:	LS.ConnectEx(&ControlInitPar1);

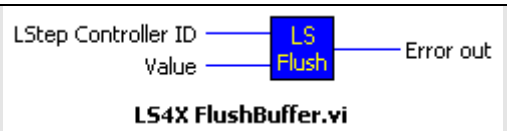
LS_ConnectSimple	
Description:	Connect with LSTEP
	The settings of the interface are delivered as a parameter. (One of the functions LS_Connect, LS_ConnectSimple or LS_ConnectEx must be called for initialising of the interface, so that the communication with the LSTEP is possible.)
Delphi:	function LS_ConnectSimple(AnInterfaceType: Integer; AComName: PChar; ABR: Integer; AShowProt: LongBool): Integer; function LSX_ConnectSimple(LSID: Integer; AnInterfaceType: Integer; AComName: PChar; ABaudRate: Integer; AShowProt: LongBool): Integer;
C++:	int Connect (int lAnInterfaceType, char *pcAComName, int lABR, BOOL AShowProt);
LabView:	
Parameter:	AnInterfaceType: Interface type 1 = RS232 2 = ArcNet 3 = DPRAM / ISA-Bus 4 = DPRAM / PCI-Bus 11= RS232 with RTS/CTS evaluation AComName: Name of the COM-interface, i.e. 'COM2', for ArcNet or DPRAM set to ZERO ABR: Meaning is dependent on the interface type RS232 → Baud rate, z. B. 9600 ArcNet: → 0 for Koax, 1 for Twisted Pair DPRAM / ISA-Bus: → Basis-I/O-Adress of the, for example 0x0340 DPRAM / PCI-Bus: → O=first card 1=second card AShowProt: determines whether the interfaces protocol is supposed to be indicated

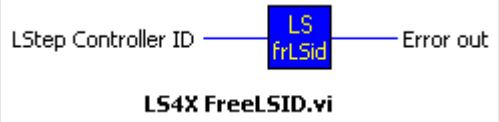
Example:	LS.ConnectSimple(1, "COM2", 9600, true);
	// RS232, 9600 Baud
	or
	LS_ConnectSimple(4, nil, 0, true);
	//LStep PCI card 0;

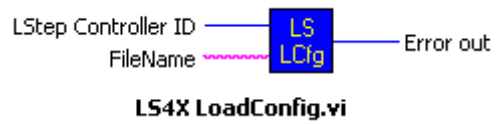
LSX_CreateLSID (only LSTEP4X-API)	
Description	Creates a LStep-ID-Number. This is used as an additional parameter in the LSTEP4X-API- commands, in order to select the LStep out of several connected LSteps, on which the command should refer to.
Delphi	function LSX_CreateLSID(var LSID: Integer): Integer;
C++	-
LabView:	
Parameter	LSID: contains after calling CreateLSID a new LStep-ID-Number, this number can be used for Connect-, moving commands and other commands
Example	var LStep1: Integer; ... LSX_CreateLSID(&LStep1);

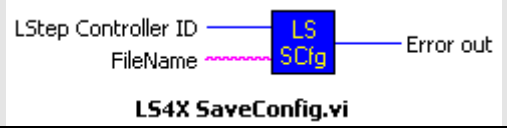
LS_Disconnect	
Description:	Disconnect LSTEP
	After calling this function, no more commands can be sent to the LSTEP. The function should be called shortly before termination of the program.
Delphi:	function LS_Disconnect: Integer; function LSX_Disconnect(LSID: Integer): Integer;
C++:	int Disconnect ();
LabView:	
Parameter:	-
Example:	LS.Disconnect();

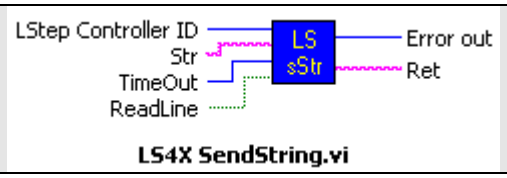
LS_EnableCommandRetry	
Description	With this function repeated sending of commands can be switched On/Off in case of a fault. (it is turned on as a standard)
Delphi	function LS_EnableCommandRetry(AValue: LongBool): Integer; function LSX_EnableCommandRetry(LSID: Integer; AValue: LongBool): Integer;
C++	int EnableCommandRetry (BOOL bAValue);
LabView:	
Parameter	AValue: true => if faults occur the LStep API repeats the sending of certain commands (especially with WaitForAxisStop) false => switch off repeated sending
Example	LS.EnableCommandRetry(false) ;

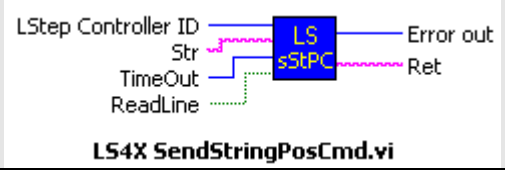
LS_FlushBuffer	
Description:	Delete communication input buffer (RS-232 und PCI) can be used in case of a fault, to erase acknowledgements from the input buffer that are no longer needed.
Delphi:	function LS_FlushBuffer(AValue: Integer): Integer; function LSX_FlushBuffer(LSID: Integer; AValue: Integer): Integer;
C++:	int FlushBuffer (int lAValue);
LabView:	
Parameter:	AValue: currently not used, can be set to =0
Example:	LS.FlushBuffer(0);

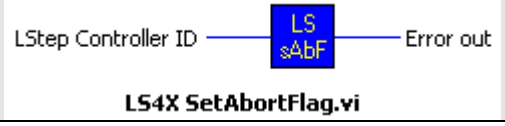
LSX_FreeLSID (only LSTEP4X-API)	
Description	Releases a created LStep-ID-number. This is used as an additional parameter in the LSTEP4X-API- commands, in order to select the LStep out of several connected LSteps, on which the command should refer to. FreeLSID should be called after Disconnect.
Delphi	function LSX_FreeLSID(LSID: Integer): Integer;
C++	-
LabView:	
Parameter	LSID: LStep-ID-number to be released; this may not used after FreeLSID
Example	var LStep1: Integer; ... LSX_CreateLSID(&LStep1); LSX_ConnectSimple(LStep1, ...); ... LSX_Disconnect(LStep1); LSX_FreeLSID(LStep1);

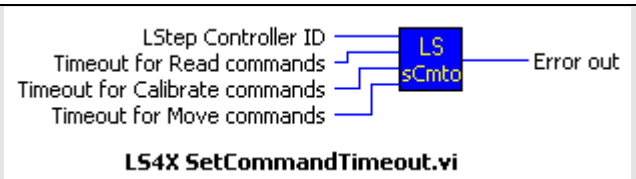
LS_LoadConfig	
Description:	Load LSTEP configuration (interface, axis settings, and controllers) from INI-file. The format of the INI-file is compatible with the Win-Commander-INI-file, i.e. the settings can be taken over from the Win-Commander (Wincom4.ini) The loaded configuration is used in the functions LS_Connect and LS_SetControlPars. Attention! For LSTEP-PCIexpress or LSTEPexpress you can save the settings with WinCommander 5 in the Controller /Export Configuration menu and can load them with LoadConfig.
Delphi:	function LS_LoadConfig(FileName: PChar): Integer; function LSX_LoadConfig(LSID: Integer; FileName: PChar): Integer;
C++:	int LoadConfig (char *pcFileName);
LabView:	
Parameter:	FileName: File name of the INI-file as a zero-terminated string
Example:	LS.LoadConfig("C:\LStepTest\LStep.INI");

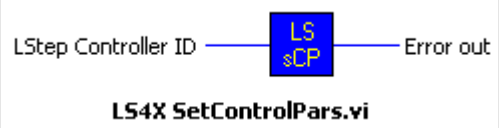
LS_SaveConfig	
Description:	Save LSTEP configuration (interface, axis settings, controllers) into INI-file. The format of the INI-file is compatible with the Win-Commander-INI-file.
Delphi:	function LS_SaveConfig(FileName: PChar): Integer; function LSX_SaveConfig(LSID: Integer; FileName: PChar): Integer;
C++:	int SaveConfig (char *pcFileName);
LabView:	
Parameter:	FileName: File name of the INI-file as a zero-terminated string
Example:	LS.SaveConfig("C:\LStepTest\LStep.INI");

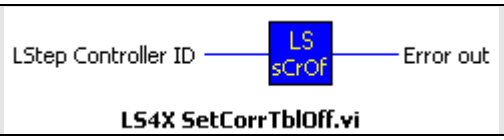
LS_SendString		
Description:	Send string to LSTEP	
Delphi:	function LS_SendString(Str, Ret: PChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer; function LSX_SendString(LSID: Integer; Str, Ret: PChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer;	
C++:	int SendString (char *pcStr,char *pcRet,int lMaxLen,BOOL ReadLine,int lTimeOut);	
LabView:		
Parameter:	Str	➔ Zero terminated String that is to be sent to the controller.
	Ret	➔ Buffer, that contains the acknowledgement of the LSTEP, if ReadLine = true
	MaxLen	➔ Maximum amount of characters that can be copied into the buffer.
	ReadLine	➔ Read acknowledgement of the LSTEP:
	TimeOut	➔ maximum waiting time for acknowledgement [ms]
Example:	LS.SendString("?ver\r", pcLStepVer, 256, true, 1000); // Read version number, Timeout 1s	

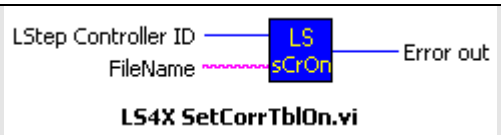
LS_SendStringPosCmd		
Description	Moving command, which awaits confirmation , send to LSTEP as a string	
Delphi	function LS_SendStringPosCmd(Str, Ret: PChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer; function LSX_SendStringPosCmd(LSID: Integer; Str, Ret: PChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer;	
C++	int SendStringPosCmd (char *pcStr, char *pcRet, int lMaxLen, BOOL bReadLine, int lTimeOut);	
LabView:		
Parameter:	Str	➔ Zero terminated String that is to be sent to the controller.
	Ret	➔ Buffer, that contains the acknowledgement of the LSTEP, if ReadLine = true
	MaxLen	➔ Maximum amount of characters that can be copied into the buffer.
	ReadLine	➔ Read acknowledgement of the LSTEP:
	TimeOut	➔ maximum waiting time for acknowledgement [ms]
Example	LS.SendStringPosCmd("!moa 1 2\r", pcLStepVer, 256, true, 100000);	

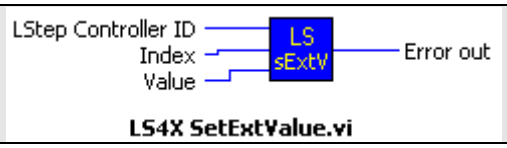
LS_SetAbortFlag	
Description:	Set flag to terminate the communication with the LSTEP
	A function which is still waiting for a feedback from the controller when LS_SetAbortFlag is called (e.g. travel commands), comes back with a fault message. This function is especially useful in programs with message handling routines or several threads, if e.g. a movement is to be aborted quickly.
Delphi:	function LS_SetAbortFlag: Integer; function LSX_SetAbortFlag(LSID: Integer): Integer;
C++:	int SetAbortFlag ();
LabView:	
Parameter:	-
Example:	LS.SetAbortFlag(); LS.StopAxes(); (Terminate communication with the LSTEP and send the command to stop all axes)

LS_SetCommandTimeout	
Description:	Sets the Timeouts for waiting for the feedback signal, of positioning and calibrating.
Delphi:	function LS_ SetCommandTimeout (AtoRead, AtoMove, AtoCalibrate: Integer): Integer; LSX_SetCommandTimeout(LSID: Integer; AtoRead, AtoMove, AtoCalibrate: Integer): Integer;
C++:	int SetCommandTimeout (int lAtoRead, int lAtoMove, int lAtoCalibrate) ;
LabView:	
Parameter:	AtoRead: Timeout for waiting fort he feedback signal [ms] AtoMove: Timeout for Positioning [ms] AtoCalibrate: Timeout for calibrating [ms]
Example:	LS. SetCommandTimeout (int lAtoRead, int lAtoMove, int lAtoCalibrate) ;

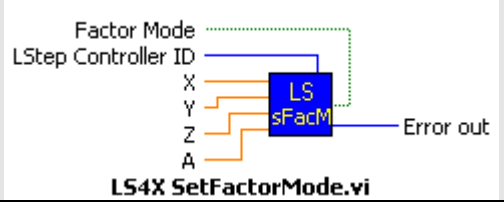
LS_SetControlPars	
Description:	Transmits the parameters, which were loaded with LS_LoadConfig to the LSTEP.
Delphi:	function LS_SetControlPars: Integer; function LSX_SetControlPars(LSID: Integer): Integer;
C++:	int SetControlPars ();
LabView:	
Parameter:	-
Example:	LS.SetControlPars();

LS_SetCorrTblOff	
Description	deactivate axis correction
Delphi	function LS_SetCorrTblOff: Integer; function LSX_SetCorrTblOff(LSID: Integer): Integer;
C++	int SetCorrTblOff ();
LabView:	
Parameter	-
Example	LS.SetCorrTblOff() ;

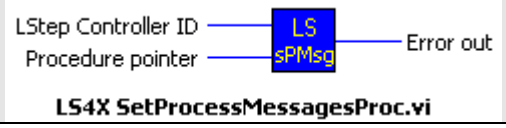
LS_SetCorrTblOn	
Description	Activate axes correction in x/y-matrix with linear interpolation
Delphi	function LS_SetCorrTblOn(AFileName: PChar): Integer; function LSX_SetCorrTblOn(LSID: Integer; AFileName: PChar): Integer;
C++	int SetCorrTblOn (char *pcAFileName);
LabView:	
Parameter	The correction table is entered manually in the Ini-file. The file name of this file is indicated by AFileName. Structure of a correction table: In the section [Options] the axes correction with linear interpolation is activated via the line „CorrectionXY=1“. XCount and YCount indicate the amount of correction values. The Parameter XDistance determines the distance of measuring points in a row (X-Axis), YDistance the distance of the rows (Y-Axis). The section [CorrTbl] contains the correction values. A corrected position is assigned to each desired value position (x/y-pair of varieties), the desired value position must be a point which is determined by the screen XCount, YCount, XDistance and YDistance. The allocations (desired value position=corrected position) can be performed in any desired sequence, important is, that the desired value position is always within the screen (Zero point of the correction table is (0|0)). Example of a correction table: [Options] CorrectionXY=1 XCount=3 YCount=3 XDistance=1.0 YDistance=1.0 [CorrTbl] 0.0 0.0=0.0 0.0 1.0 0.0=1.0 0.0 2.0 0.0=2.0 0.0 0.0 1.0=0.0 1.0 1.0 1.0=0.9 1.1 (desired value position x=1 y=1, corrected Position x=0.9 y=1.1) 2.0 1.0=2.0 1.0 0.0 2.0=0.0 2.0 1.0 2.0=1.0 2.0 2.0 2.0=2.0 2.0
Example	LS.SetCorrTblOn(„C:\...\corrtbl.ini“);

LS_SetExtValue	
Description:	Switches on the extension of the API, partly it concerns experimental Modes for Debugging purposes.
Delphi:	function LS_SetExtValue(AName: Integer; AValue: Integer): Integer; function LSX_SetExtValue(LSID: Integer; AName, AValue: Integer): Integer;
C++:	int SetExtValue (int lAName, int lAValue);
LabView:	
Parameter:	AName: Number of the extended function AValue: Parameter **AName=2 (IFSleepTime)** setup of the Polling-Interval for the DPRAM of the LStep-PCI AValue: Time-interval in [ms], standard is 10 **AName=3 (ProtMoveOnly)** switches on filter for Log-file, which only protocols Moves&Errors. AValue=1 → Filter on AValue=0 → Filter off **AName=4 (Max_LogLn)** limits the length of the Log-file, older Log-file will be renamed in .old AValue=Maximum number of line **AName=5 (ThreadPriority)** changes the priority of Threads of the LStep API. After Connect the Threads are always set to normal Priority, with SetExtValue(5, ...) they can be changed one after the other. AValue=Windows-API-constant for Thread-Priority like THREAD_PRIORITY_ABOVE_NORMAL

Example:	LS.SetExtValue(3, 1); // Filter for Move-commands on
	LS.SetExtValue(4, 10000); // maximum length of the Log-file = 10000 lines
	LS.SetExtValue(5, THREAD_PRIORITY_HIGHEST);

LS_SetFactorMode	
Description	Position value-Conversion for ‚krumme' spindle pitch
Delphi	function LS_SetFactorMode(AFactorMode: LongBool; X, Y, Z, A: Double): Integer; function LSX_SetFactorMode(LSID: Integer; AFactorMode: LongBool; X, Y, Z, A: Double): Integer;
C++	int SetFactorMode (BOOL bAFactorMode, double dX, double dY, double dZ, double dA);
LabView:	
Parameter	AFactorMode: Switch on factor-mode This command activates a API-internal conversion of the position values/spindle pitch, to avoid rounding errors with 'odd' spindle pitch X, Y, Z, R: Spindle pitch values, that are transferred to the LStep (if possible values like 1.0 or 4.0, so that a micro step corresponds with a non periodic decimal fraction) Only **after** SetFactorMode, SetPitch should be called with the actual physical spindle pitch. All moving commands use after calling SetFactorMode and SetPitch a factor-conversion, so that the LStep is positioned correctly. send to LStep Position vector = Position vector * send to LStep spindle pitch /physical spindle pitch
Example	LS.SetFactorMode(true, 1, 1, 1, 0); LS.SetPitch(1.234, 1.234, 2.345, 0); LS.MoveAbs(1.234, 2.468, 2.345, 0, true);

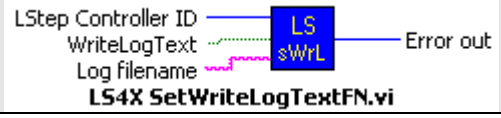
LS_SetLanguage	
Description:	Set language for LSTEP-API (log / messages)
Delphi:	function LS_SetLanguage(PLN: PChar): Integer; function LSX_SetLanguage(LSID: Integer; PLN: PChar): Integer;
C++:	int SetLanguage (char *pcPLN);
LabView:	
Parameter:	PLN: Language (Abbreviated, e.g. “DEU” or “ENG”) The appropriate text file (LSTEP4deu.txt or LSTEP4eng.txt) must be in the program directory
Example:	LS.SetLanguage(‘ENG’);

LS_SetProcessMessagesProc	
Description	Enables the replacement of the internal message-dispatching procedure of the LStep API. The LStep API processes during waiting for confirmation of the LStep in the main-thread messages. If you want to switch of the Message-Dispatching or replace with your own Code, you can use SetProcessMessagesProc for using a callback-procedure.
Delphi	function LS_SetProcessMessagesProc(Proc: Pointer): Integer; function LSX_SetProcessMessagesProc(LSID: Integer; Proc: Pointer): Integer;
C++	int SetProcessMessagesProc (void* pProc);
LabView:	
Parameter	pProc must be a pointer to a stdcall-procedure without a parameter : void MyProcessMessages () { .. }
Example	LS. SetProcessMessagesProc (&MyProcessMessages);

LS_SetShowCmdList	
Description:	LStep-API Command list On/Off
Delphi:	function LS_SetShowCmdList(ShowCmdList: LongBool): Integer; function LSX_SetShowCmdList(LSID: Integer; ShowCmdList: LongBool): Integer;
C++:	int SetShowCmdList (BOOL bShowCmdList);
LabView:	-
Parameter:	ShowProt: Indicates, if the window „LStep-API command list“ should be shown
Example:	LS.SetShowCmdList(true); // Show interface protocol if not already visible

LS_SetShowProt	
Description:	Interface protocol On/Off
Delphi:	function LS_SetShowProt(ShowProt: LongBool): Integer; function LSX_SetShowProt(LSID: Integer; ShowProt: LongBool): Integer;
C++:	int SetShowProt (BOOL ShowProt);
LabView:	
Parameter:	ShowProt: Specifies whether the window “Interface Protocol” is to be shown or not
Example:	LS.SetShowProt(true); // Show interface protocol if not already visible

LS_SetWriteLogText	
Description:	Switch on / switch off write log file LSTEP4.log (Writing in LSTEP4-log is normally switched off)
Delphi:	function LS_SetWriteLogText(AWriteLogText: LongBool): Integer; function LSX_SetWriteLogText(LSID: Integer; AWriteLogText: LongBool): Integer;
C++:	int SetWriteLogText (BOOL AWriteLogText);
LabView:	
Parameter:	-
Example:	LS.SetWriteLogText (true);

LS_SetWriteLogTextFN	
Description	Switch On/Off the writing a interface –protocol in certain file (Standard mode the writing is turned off.)
Delphi	function LS_SetWriteLogTextFN(AWriteLogText: LongBool; ALogFN: PChar): Integer; function LSX_SetWriteLogTextFN(LSID: Integer; AWriteLogText: LongBool; ALogFN: PChar): Integer;
C++	int SetWriteLogTextFN (BOOL bAWriteLogText, char *pcALogFN);
LabView:	
Parameter	AWriteLogText: true => write protocol file ALogFN: filename of the protocol file
Example	LS.SetWriteLogTextFN(true, „C:\Temp\prot.txt“);

Controller-Info

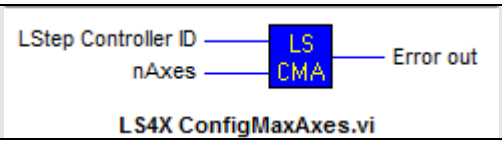
LS_GetSerialNr	
Description:	Read serial number of the controller
Delphi:	function LS_GetSerialNr(SerialNr: PChar; MaxLen: Integer): Integer; function LSX_GetSerialNr(LSID: Integer; SerialNr: PChar; MaxLen: Integer): Integer;
C++:	int GetSerialNr (char *pcSerialNr,int lMaxLen);
LabView:	
Parameter:	SerialNr: Pointer to a buffer in which the serial number is returned MaxLen: Maximum number of characters, that can be copied into the buffer
Example:	LS.GetSerialNr(pcSerialNr, 256);

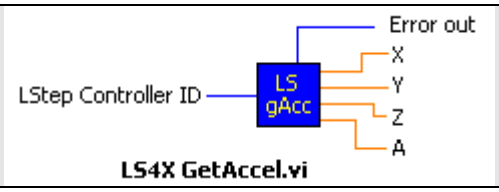
LS_GetVersionStr	
Description:	gives the current Firmware version number
Delphi:	function LS_GetVersionStr(Vers: PChar; MaxLen: Integer): Integer; function LSX_GetVersionStr(LSID: Integer; Vers: PChar; MaxLen: Integer): Integer;
C++:	int GetVersionStr (char *pcVers,int lMaxLen);
LabView:	
Parameter:	Stat: Pointer to a buffer in which the version string is returned MaxLen: Maximum number of characters, that can be copied into the buffer
Example:	LS.GetVersionStr(pcVers, 64); // Read version number

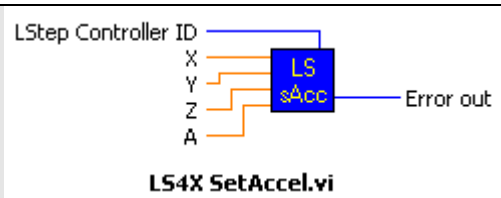
LS_GetVersionStrDet	
Description:	Read out detailed version number of Firmware
Delphi:	function LS_GetVersionStrDet(VersDet: PChar; MaxLen: Integer): Integer; function LSX_GetVersionStrDet(LSID: Integer; VersDet: PChar; MaxLen: Integer): Integer;
C++:	int GetVersionStrDet (char *pcVersDet, int lMaxLen);
LabView:	
Parameter:	VersDet: Points to buffer, where the detailed versio-string is returned to MaxLen: Maximum number of characters, that can be copied into the buffer
Example:	LS.GetVersionStrDet(pcVersDet, 64); // read out detailed version number

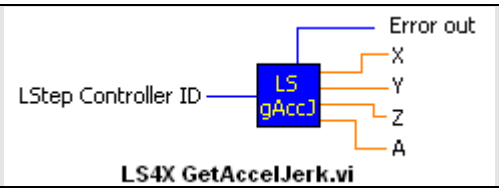
LS_GetVersionStrInfo	
Description:	Gives detailed information about version number
Delphi:	function LS_ GetVersionStrInfo (VersInfo: PChar; MaxLen: Integer): Integer; function LSX_ GetVersionStrInfo (LSID: Integer; VersInfo: PChar; MaxLen: Integer): Integer;
C++:	int GetVersionStrInfo (char *pcVersInfo, int lMaxLen);
LabView:	
Parameter:	VersInfo: Points to buffer, in which the day of the week, calendar week, year , consecutive number is returned to i. e.: T04.35.02-0004 MaxLen: Maximum number of characters, that can be copied into the buffer
Example:	LS.GetVersionStrInfo (pcVersInfo, 64);

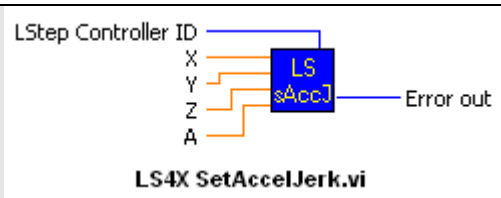
Settings

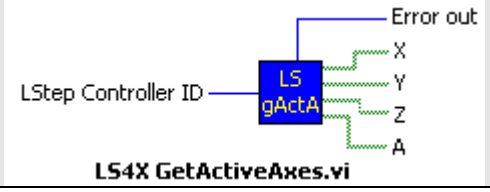
LS_ConfigMaxAxes	
Description:	Configures the number of axes
Delphi:	function LS_ ConfigMaxAxes s (nAxes: Integer): Integer; function LSX_ConfigMaxAxes(LSID: Integer; nAxes: Integer): Integer;
C++:	int ConfigMaxAxes (int nAxes);
LabView:	
Parameter:	nAxes: Number of axes, 1 - 4
Example:	LS. ConfigMaxAxes (2); // controller has two axes

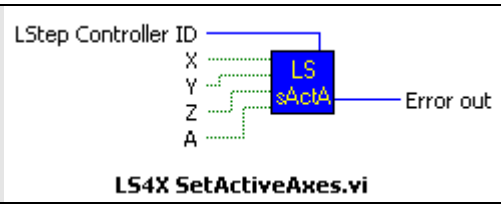
LS_GetAccel	
Description:	Inquiry of acceleration
Delphi:	function LS_GetAccel(var X, Y, Z, R: Double): Integer; function LSX_GetAccel(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetAccel (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Acceleration values [m/s²]
Example:	LS.GetAccel(&X, &Y, &Z, &A);

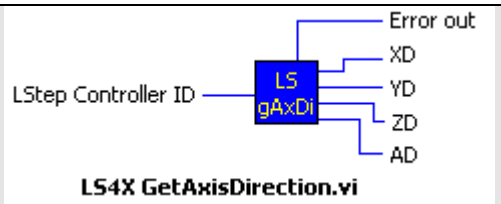
LS_SetAccel	
Description:	Set acceleration
Delphi:	function LS_SetAccel(X, Y, Z, R: Double): Integer; function LSX_SetAccel(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetAccel(double dX, double dY, double dZ, double dA);
LabView:	
Parameter:	X, Y, Z and A 0.01 – 10.00 [m/s²]
Example:	LS.SetAccel(1.0, 1.5, 0, 0);

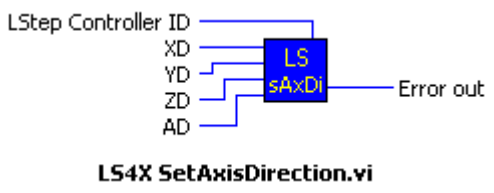
LS_GetAccelJerk	
Description:	Ask jerk during acceleration
Delphi:	function LS_GetAccelJerk(var XD, YD, ZD, AD: Double): Integer; function LSX_GetAccelJerk(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;
C++:	int GetAccelJerk(double *pdXD, double *pdYD, double *pdZD, double *pdAD);
LabView:	
Parameter:	XD, YD, ZD, AD: Jerk values [m/s³]
Example:	LS.GetAccelJerk(&XD, &YD, &ZD, &AD);

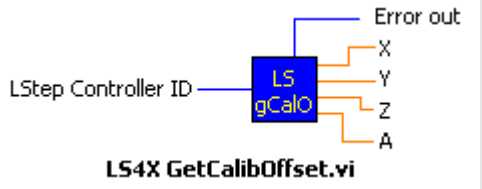
LS_SetAccelJerk	
Description:	Adjust jerk during acceleration
Delphi:	function LS_SetAccelJerk(XD, YD, ZD, AD: Double): Integer; function LSX_SetAccelJerk(LSID: Integer; XD, YD, ZD, AD: Double): Integer;
C++:	int SetAccelJerk(double dXD, double dYD, double dZD, double dAD);
LabView:	
Parameter:	X, Y, Z and A: 1, 2, 3, ... ∞ [m/s³] (Natural numbers)
Example:	LS.SetAccelJerk(1.0, 1.5, 0, 0);

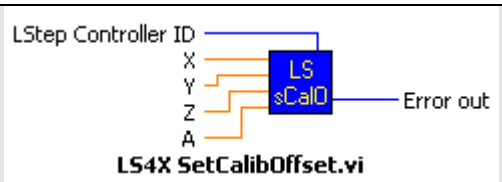
LS_GetActiveAxes	
Description:	Delivers enable axes
Delphi:	function LS_GetActiveAxes(var Flags: Integer): Integer; function LSX_GetActiveAxes(LSID: Integer; var Flags: Integer): Integer;
C++:	int GetActiveAxes (int *plFlags);
LabView:	
Parameter:	Flags: 32-bit-Integer, which contains after calling of the function in the Bits 0-4 the Bit-mask. Bit 0 = 1 → X-axis enabled, i.e. can be travelled Bit 2 = 0 → Z-axis not enabled
Example:	LS.GetActiveAxes(&Flags);

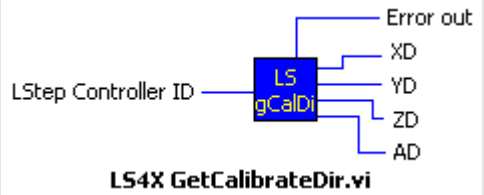
LS_SetActiveAxes	
Description:	Enable axes
Delphi:	function LS_SetActiveAxes(Flags: Integer): Integer; function LSX_SetActiveAxes(LSID: Integer; Flags: Integer): Integer;
C++:	int SetActiveAxes(int Flags);
LabView:	
Parameter:	Flags: Bit mask Bit 0 = 1 → X-axis enabled, i.e. can be travelled Bit 2 = 0 → Z-axis not enabled
Example:	LS.SetActiveAxes(3); /* Enable X- and Y-axes (Bits 0 and 1 set), do not enable Z-axis (Bit 2 = 0) */

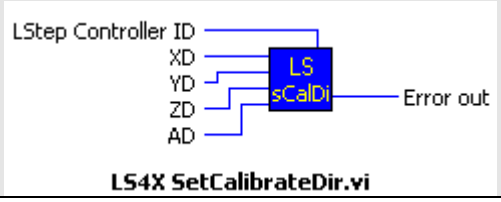
LS_GetAxisDirection	
Description	Inquiry of reverse-turning direction
Delphi	function LS_GetAxisDirection(var XD, YD, ZD, AD: Integer): Integer; function LSX_GetAxisDirection(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;
C++	int GetAxisDirection (int *plXD, int *plYD, int *plZD, int *plAD);
LabView:	
Parameter	XD, YD, ZD, AD: 32-bit-Integers 0 => normal rotation direction 1 => Reverse-turning direction
Example	LS.GetAxisDirection(&XD, &YD, &ZD, &AD);

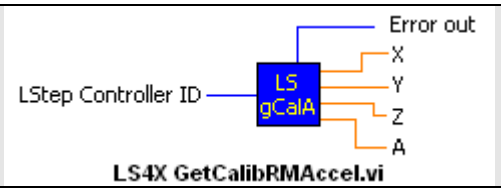
LS_SetAxisDirection	
Description	Reverse turning direction
Delphi	function LS_SetAxisDirection(XD, YD, ZD, AD: Integer): Integer; function LSX_SetAxisDirection(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;
C++	int SetAxisDirection (int lXD, int lYD, int lZD, int lAD);
LabView:	
Parameter	XY, YD, ZD, AD: 0 => normal rotation direction 1 => Reverse-turning direction
Example	LS.SetAxisDirection(1, 0, 0, 0); // Reverse turning direction of X-Axis

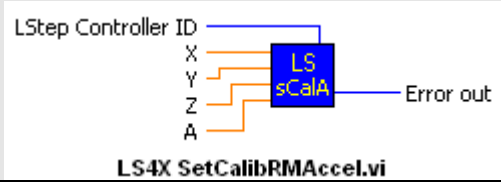
LS_GetCalibOffset	
Description:	Inquiry of calibration-offset
Delphi:	function LS_GetCalibOffset(var X, Y, Z, A: Double): Integer; function LSX_GetCalibOffset(LSID: Integer; var X, Y, Z, R: Double): Integer;
C++:	int GetCalibOffset (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z, A: calibration-offset dependent on dimension.
Example:	LS.GetCalibOffset(&X, &Y, &Z, &A);

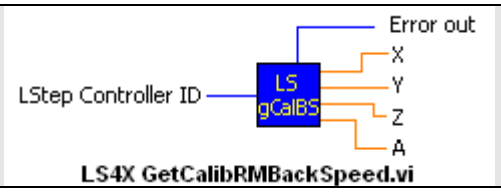
LS_SetCalibOffset	
Description:	Calibration Offset
Delphi:	function LS_SetCalibOffset(X, Y, Z, A: Double): Integer; function LSX_SetCalibOffset(LSID: Integer; X, Y, Z, R: Double): Integer;
C++:	int SetCalibOffset (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, y, z and a 0 – 32*50000 (32*spindle pitch)
Example:	LS.SetCalibOffset(1, 1, 1, 1); (When calibration is done, the X-, Y- and Z- axes are each moved 1 mm (for Dim. 2 2 2) away from the zero limit switch towards the center of the table and the zero position is then set (software limit).

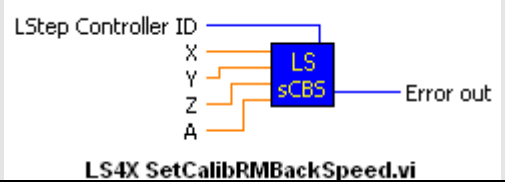
LS_GetCalibrateDir	
Description	Inquiry reverse preceding sign when calibrating
Delphi	function LS_GetCalibrateDir(var XD, YD, ZD, AD: Integer): Integer; function LSX_GetCalibrateDir(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;
C++	int GetCalibrateDir (int *plXD, int *plYD, int *plZD, int *plAD);
LabView:	
Parameter	XD, YD, ZD, AD: 32-bit-Integers 0 => no reversing of preceding sign 1 => reverse preceding sign
Example	LS.GetCalibrateDir(&XD, &YD, &ZD, &AD);

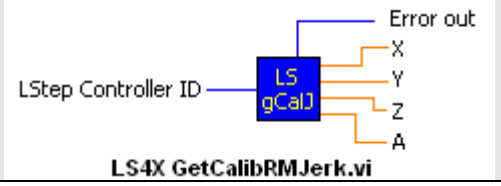
LS_SetCalibrateDir	
Description	Reverse preceding sign when calibrating
Delphi	function LS_SetCalibrateDir(XD, YD, ZD, AD: Integer): Integer; function LSX_SetCalibrateDir(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;
C++	int SetCalibrateDir (int lXD, int lYD, int lZD, int lAD);
LabView:	
Parameter	XD, YD, ZD, AD: 0 => no reversing of preceding sign 1 => reverse preceding sign
Example	LS.SetCalibrateDir(1, 1, 0, 0);

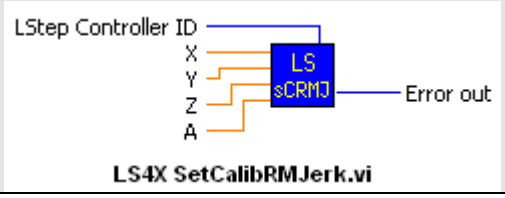
LS_GetCalibRMAccel	
Description:	Ask acceleration during calibration procedure
Delphi:	function LS_GetCalibRMAccel (var XD, YD, ZD, AD: Double): Integer; function LSX_GetCalibRMAccel (LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetCalibRMAccel (double *pdXD, double *pdYD, double *pdZD, double *pdAD);
LabView:	
Parameter:	XD, YD, ZD, AD: Acceleration values [m/s²]
Example:	LS.GetCalibRMAccel (&XD, &YD, &ZD, &AD);

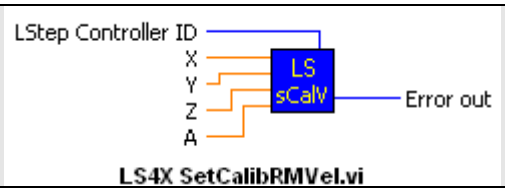
LS_SetCalibRMAccel	
Description:	Adjust acceleration during calibration procedure
Delphi:	function LS_SetCalibRMAccel (XD, YD, ZD, AD: Double): Integer; function LSX_SetCalibRMAccel (LSID: Integer; XD, YD, ZD, AD: Double): Integer;
C++:	int SetCalibRMAccel (double dXD, double dYD, double dZD, double dAD);
LabView:	
Parameter:	XD, YD, ZD and AD: Acceleration values 0.01 – 20.00 [m/s2]
Example:	LS.SetCalibRMAccel (1.0, 1.5, 0, 0);

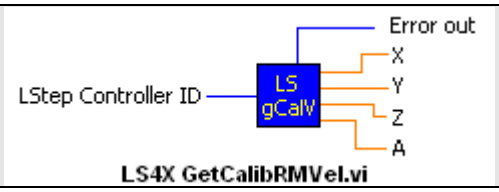
LS_GetCalibRMBackSpeed	
Description:	Ask positioning speeds for move out of the limit switch during calibration procedure
Delphi:	function LS_GetCalibRMBackSpeed (var XD, YD, ZD, AD: Double): Integer; function LSX_GetCalibRMBackSpeed (LSID: Integer; var XD, YD, ZD, AD: Double): Integer;
C++:	int GetCalibRMBackSpeed (double *pdXD, double *pdYD, double *pdZD, double *pdAD);
LabView:	
Parameter:	XD, YD, ZD, AD: Speed values [rp/s]
Example:	LS.GetCalibRMBackSpeed (&XD, &YD, &ZD, &AD);

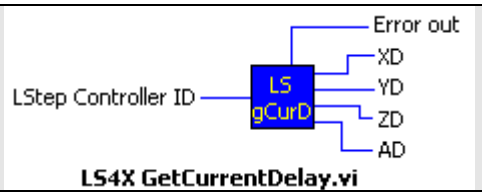
LS_SetCalibRMBackSpeed	
Description:	Ask positioning speeds for move out of the limit switches during calibration procedure
Delphi:	function LS_SetCalibRMBackSpeed (XD, YD, ZD, AD: Double): Integer; function LSX_SetCalibRMBackSpeed (LSID: Integer; XD, YD, ZD, AD: Double): Integer;
C++:	int SetCalibRMBackSpeed (double dXD, double dYD, double dZD, double dAD);
LabView:	
Parameter:	XD, YD, ZD and AD: Speed, 5..100 [U/s]
Example:	LS.SeCalibRMBackSpeed (1.0, 15.0, 0, 0);

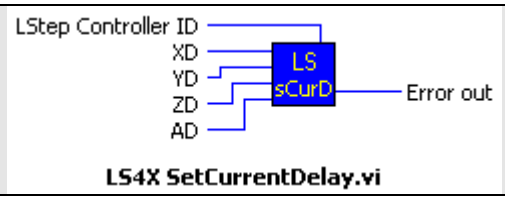
LS_GetCalibRMJerk	
Description:	Ask jerk during calibration procedure
Delphi:	function LS_GetCalibRMJerk(var XD, YD, ZD, AD: Double): Integer; function LSX_GetCalibRMJerk(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;
C++:	int GetCalibRMJerk (double *pdXD, double *pdYD, double *pdZD, double *pdAD);
LabView:	
Parameter:	XD, YD, ZD, AD: Jerk values [m/s³]
Example:	LS.GetCalibRMJerk(&XD, &YD, &ZD, &AD);

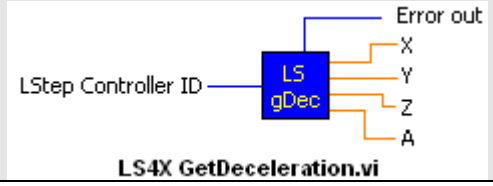
LS_SetCalibRMJerk	
Description:	Jerk during calibration
Delphi:	function LS_SetCalibRMJerk(XD, YD, ZD, AD: Double): Integer; function LSX_SetCalibRMJerk(LSID: Integer; XD, YD, ZD, AD: Double): Integer;
C++:	int SetCalibRMJerk(double dXD, double dYD, double dZD, double dAD);
LabView:	
Parameter:	X, Y, Z and A: 1, 2, 3, ... ∞ [m/s³] (Natural numbers)
Example:	LS.SetCalibRMJerk(1.0, 1.5, 0, 0);

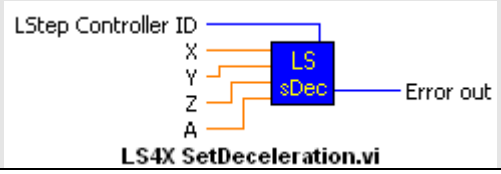
LS_SetCalibRMVel	
Description:	Set speeds of travel during the calibration process
Delphi:	function LS_SetCalibRMVel (XD, YD, ZD, AD: Double): Integer; function LSX_SetCalibRMVel (LSID: Integer; XD, YD, ZD, AD: Double): Integer;
C++:	int SetCalibRMVel (double dXD, double dYD, double dZD, double dAD);
LabView:	
Parameter:	XD, YD, ZD and AD: Speed [rp/s]
Example:	LS.SeCalibRMVel (1.0, 15.0, 0, 0);

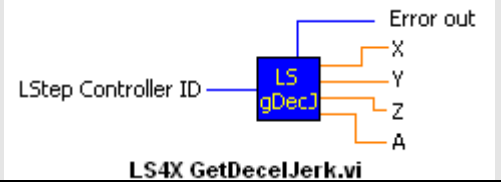
LS_GetCalibRMVel	
Description:	Query speeds of travel during the calibration process
Delphi:	function LS_GetCalibRMVel(var XD, YD, ZD, AD: Double): Integer; function LSX_GetCalibRMVel(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;
C++:	int GetCalibRMVel (double *pdXD, double *pdYD, double *pdZD, double *pdAD);
LabView:	
Parameter:	XD, YD, ZD, AD: Speed values [rp/s]
Example:	LS.GetCalibRMVel(&XD, &YD, &ZD, &AD);

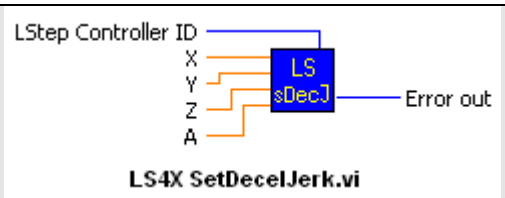
LS_GetCurrentDelay	
Description:	Indicates time delay for current reduction
Delphi:	function LS_GetCurrentDelay(var X, Y, Z, R: Integer): Integer; function LSX_GetCurrentDelay(LSID: Integer; var X, Y, Z, R: Integer): Integer;
C++:	int GetCurrentDelay (int *plX, int *plY, int *plZ, int *plR);
LabView:	
Parameter:	X, Y, Z, R: Time delay in ms
Example:	LS.SetCurrentDelay(&X, &Y, &Z, &A);

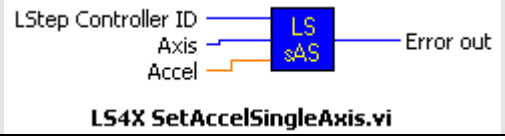
LS_SetCurrentDelay	
Description:	Time delay for current reduction
Delphi:	function LS_SetCurrentDelay(X, Y, Z, R: Integer): Integer; function LSX_SetCurrentDelay(LSID: Integer; X, Y, Z, R: Integer): Integer;
C++:	int SetCurrentDelay (int lX, int lY, int lZ, int lR);
LabView:	
Parameter:	X, Y, Z, R: 0-10000 [ms]
Example:	LS.SetCurrentDelay(100, 300, 1000, 0) ;

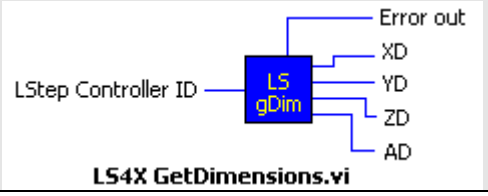
LS_GetDeceleration	
Description:	Ask for delays
Delphi:	function LS_GetDeceleration (var XD, YD, ZD, AD: Double): Integer; function LSX_GetDeceleration (LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetDeceleration (double *pdXD, double *pdYD, double *pdZD, double *pdAD);
LabView:	
Parameter:	XD, YD, ZD, AD: delay values [m/s²]
Example:	LS.GetDeceleration (&XD, &YD, &ZD, &AD);

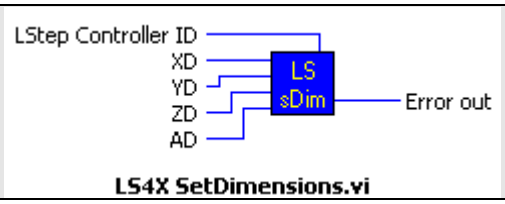
LS_SetDeceleration	
Description:	Adjust delay
Delphi:	function LS_SetDeceleration (XD, YD, ZD, AD: Double): Integer; function LSX_SetDeceleration (LSID: Integer; XD, YD, ZD, AD: Double): Integer;
C++:	int SetDeceleration (double dXD, double dYD, double dZD, double dAD);
LabView:	
Parameter:	XD, YD, ZD and AD: delay values 0.01 – 20.00 [m/s2]
Example:	LS.SetDeceleration (1.0, 1.5, 0, 0);

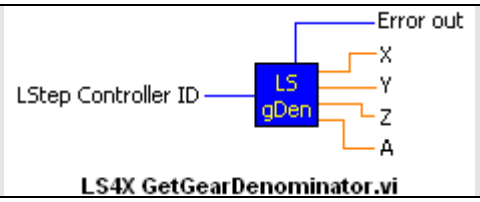
LS_GetDecelJerk	
Description:	Ask jerk during delay
Delphi:	function LS_GetDecelJerk(var XD, YD, ZD, AD: Double): Integer; function LSX_GetDecelJerk(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;
C++:	int GetDecelJerk(double *pdXD, double *pdYD, double *pdZD, double *pdAD);
LabView:	
Parameter:	XD, YD, ZD, AD: Jerk values [m/s³]
Example:	LS.GetDecelJerk(&XD, &YD, &ZD, &AD);

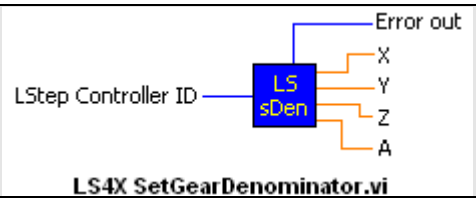
LS_SetDecelJerk	
Description:	Adjust jerk during the delay
Delphi:	function LS_SetDecelJerk(XD, YD, ZD, AD: Double): Integer; function LSX_SetDecelJerk(LSID: Integer; XD, YD, ZD, AD: Double): Integer;
C++:	int SetDecellJerk(double dXD, double dYD, double dZD, double dAD);
LabView:	
Parameter:	X, Y, Z and A: 1, 2, 3, ... ∞ [m/s³] (Natural numbers)
Example:	LS.SetDecelJerk(1.0, 1.5, 0, 0);

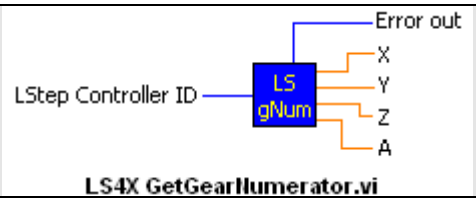
LS_SetDecelSingleAxis	
Description:	Adjust delay
Delphi:	function LS_SetDecelSingleAxis(Axis: Integer; Accel: Double): Integer; function LSX_SetDecelSingleAxis(LSID: Integer; Axis: Integer; Accel: Double): Integer;
C++:	int SetDecelSingleAxis (int lAxis,double dAccel);
LabView:	
Parameter:	Axis: (X, Y, Z, A numbered from 1 to 4) Accel delay 0.01 – 10.00 [m/s2]
Example:	LS.SetDecelSingleAxis(1, 1.0); // delay X-axis 1.0 m/s²

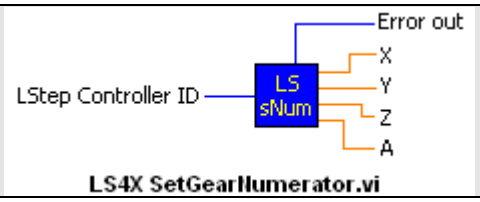
LS_GetDimensions	
Description:	Inquiry dimensions of the axes
Delphi:	function LS_GetDimensions(var XD, YD, ZD, AD: Integer): Integer; function LSX_GetDimensions(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;
C++:	int GetDimensions (int *plXD, int *plYD, int *plZD, int *plAD);
LabView:	
Parameter:	XD, YD, ZD, AD: Dimension values
	0 ➔ Microsteps 1 ➔ µm 2 ➔ Millimeters 3 ➔ Degrees 4 ➔ Revolutions
Example:	LS. GetDimensions (&XD, &YD, &ZD, &AD);

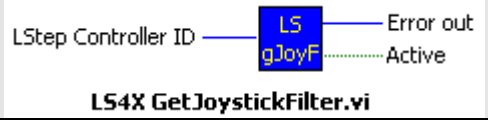
LS_SetDimensions	
Description:	Set dimensions of the axes
Delphi:	function LS_SetDimensions(XD, YD, ZD, AD: Integer): Integer; function LSX_SetDimensions(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;
C++:	int SetDimensions (int lXD,int lYD,int lZD,int lAD);
LabView:	
Parameter:	Dimensions of the X, Y, Z and A-axes: 0 ➔ Microsteps 1 ➔ µm 2 ➔ Millimeters 3 ➔ Degrees 4 ➔ Revolutions
Example:	LS.SetDimensions(3, 2, 2); // X-axis in degrees; Y and Z in mm

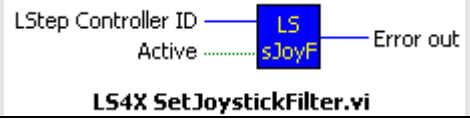
LS_GetGearDenominator	
Description	Ask denominator of gear ratio
Delphi	function LS_GetGearDenominator (var X, Y, Z, A: Integer): Integer; function LSX_ GetGearDenominator (LSID: Integer; var X, Y, Z, A: Integer): Integer;
C++	int GetGearDenominator (int *plX, int *plY, int *plZ, int *plA);
LabView:	
Parameter	X, Y, Z, A: Nominator (Denominator) of gear ratio
Example	LS. GetGearDenominator (&X, &Y, &Z, &A);

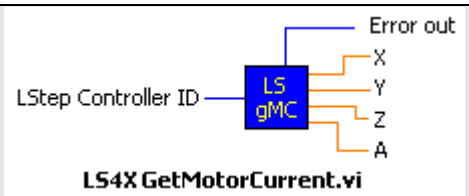
LS_SetGearDenominator	
Description	Adjust denominator or gear ratio
Delphi	function LS_ SetGearDenominator (X, Y, Z, A: Integer): Integer; function LSX_ SetGearDenominator (LSID: Integer; X, Y, Z, A: Integer): Integer;
C++	int SetGearDenominator (int lX, int lY, int lZ, int lA);
LabView:	
Parameter	X, Y, Z, A: 1, 2, 3, ... ∞ (Natural numbers)
Example	LS. SetGearDenominator (2, 0, 0, 0); // gear ratio 2/γ bei x

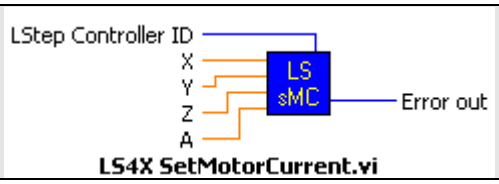
LS_GetGearNumerator	
Description	Ask counter of gear ratio
Delphi	function LS_GetGearNumerator (var X, Y, Z, A: Integer): Integer; function LSX_ GetGearNumerator (LSID: Integer; var X, Y, Z, A: Integer): Integer;
C++	int GetGearNumerator (int *plX, int *plY, int *plZ, int *plA);
LabView:	
Parameter	X, Y, Z, A: Numerator (counter) of gear ratio
Example	LS. GetGearNumerator (&X, &Y, &Z, &A);

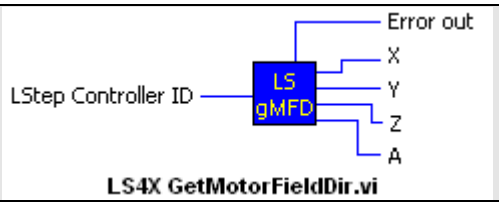
LS_SetGearNumerator	
Description	Adjust counter of gear ratio
Delphi	function LS_ SetGearNumerator (X, Y, Z, A: Integer): Integer; function LSX_ SetGearNumerator (LSID: Integer; X, Y, Z, A: Integer): Integer;
C++	int SetGearNumerator (int lX, int lY, int lZ, int lA);
LabView:	
Parameter	X, Y, Z, A: 1, 2, 3, ... ∞ (Natural numbers)
Example	LS. SetGearNumerator (4, 0, 0, 0);

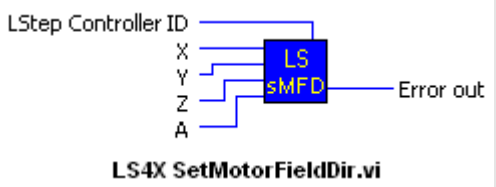
LS_GetJoystickFilter	
Description:	Indicates, if the filtering and hysteresis is activated in joystick operation
Delphi:	function LS_GetJoystickFilter(var bActive: LongBool): Integer; function LSX_ GetJoystickFilter (LSID: Integer; var bActive: LongBool): Integer;
C++:	int GetJoystickFilter (BOOL *pbActive);
LabView:	
Parameter:	bActive: True – filtering activates False – deactivated
Example:	LS.GetJoystickFilter (&Active);

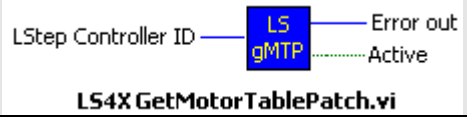
LS_SetJoystickFilter	
Description:	Activating/Deactivating the filtering and hysteresis in joystick operation
Delphi:	function LS_SetJoystickFilter(bActive: LongBool): Integer; function LSX_ SetJoystickFilter (LSID: Integer; bActive: LongBool): Integer;
C++:	int SetJoystickFilter (BOOL bActive);
LabView:	
Parameter:	bActive: True – filtering activates False – deactivated
Example:	LS. SetJoystickFilter (True);

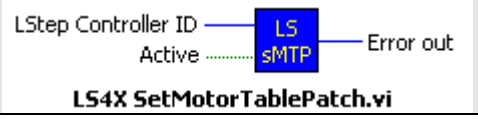
LS_GetMotorCurrent	
Description:	Inquiry motor current
Delphi:	function LS_GetMotorCurrent(var X, Y, Z, A: Double): Integer; function LSX_GetMotorCurrent(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetMotorCurrent (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Motor current [A]
Example:	LS.GetMotorCurrent(&X, &Y, &Z, &A);

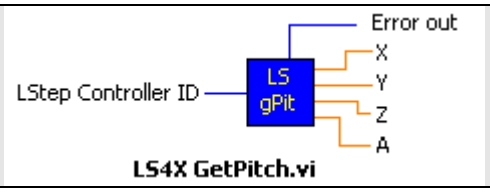
LS_SetMotorCurrent	
Description:	Set motor current
Delphi:	function LS_SetMotorCurrent(X, Y, Z, A: Double): Integer; function LSX_SetMotorCurrent(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetMotorCurrent (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	Motor current X, Y, Z, A-axis [A]
Example:	LS.SetMotorCurrent(1.5, 1.5, 1.0, 1.0); // Motor current for X and Y is 1.5 amperes; for Z and A, 1.0 amperes

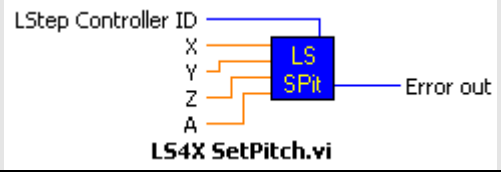
LS_GetMotorFieldDir	
Description	Query rotating direction of the motor
Delphi	function LS_GetMotorFieldDir (var X, Y, Z, A: Integer): Integer; function LSX_ GetMotorFieldDir (LSID: Integer; var X, Y, Z, A: Integer): Integer;
C++	int GetMotorFieldDir (int *plX, int *plY, int *plZ, int *plA);
LabView:	
Parameter	X, Y, Z, A: 32-bit-Integers 0 => normal rotation direction 1 => Reverse-turning direction
Example	LS. GetMotorFieldDir (&X, &Y, &Z, &A);

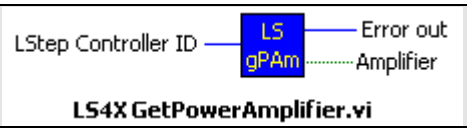
LS_SetMotorFieldDir	
Description	Adjust rotation direction of the motor
Delphi	function LS_ SetMotorFieldDir (X, Y, Z, A: Integer): Integer; function LSX_ SetMotorFieldDir (LSID: Integer; X, Y, Z, A: Integer): Integer;
C++	int SetMotorFieldDir (int lX, int lY, int lZ, int lA);
LabView:	
Parameter	X, Y, Z, A: 0 => normal rotation direction 1 => Reverse-turning direction
Example	LS. SetMotorFieldDir (1, 0, 0, 0); // Reverse turning direction of X-axis

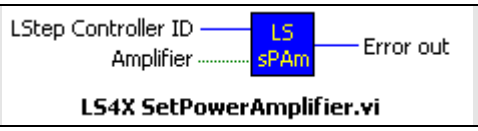
LS_GetMotorTablePatch	
Description:	Indicates, if the correction table is activated.
Delphi:	function LS_GetMotorTablePatch(bActive: var LongBool): Integer; function LSX_ GetMotorTablePatch (LSID: Integer; var bActive: LongBool): Integer;
C++:	int GetMotorTablePatch (BOOL *pbActive);
LabView:	
Parameter:	bActive: True – table is activated False – deactivated
Example:	LS. GetMotorTablePatch (&Active);

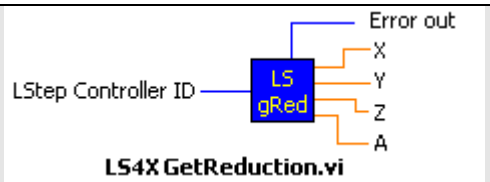
LS_SetMotorTablePatch	
Description:	The correction table becomes activated The correction table was determined for a special motor by measurement. Correction tables can be determined on customer wish.
Delphi:	function LS_SetMotorTablePatch(bActive: LongBool): Integer; function LSX_ SetMotorTablePatch (LSID: Integer; bActive: LongBool): Integer;
C++:	int SetMotorTablePatch (BOOL bActive);
LabView:	
Parameter:	bActive: True – table is activated False – deactivated
Example:	LS. SetMotorTablePatch (True);

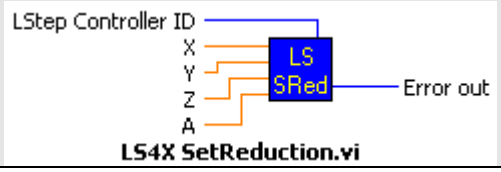
LS_GetPitch	
Description:	delivers spindle pitch
Delphi:	function LS_GetPitch(var X, Y, Z, R: Double): Integer; function LSX_GetPitch(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetPitch (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Spindle pitch [mm]
Example:	LS. GetPitch (&X, &Y, &Z, &A);

LS_SetPitch	
Description:	Set spindle pitch
Delphi:	function LS_SetPitch(X, Y, Z, R: Double): Integer; function LSX_SetPitch(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetPitch(double dX, double dY, double dZ, double dA);
LabView:	
Parameter:	X, Y, Z and A 0.001 – 68 [mm]
Example:	LS.SetPitch(4, 4, 4, 4); // Set spindle pitches to 4 mm for all axes

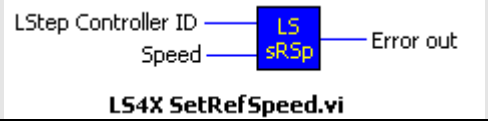
LS_GetPowerAmplifier	
Description:	Indicates if the power amplifiers of the LS44 are switched ON or OFF. This command only exists for the LS44-controller.
Delphi:	function LS_GetPowerAmplifier (bAmplifier: var LongBool): Integer; function LSX_ GetPowerAmplifier (LSID: Integer; var bAmplifier: LongBool): Integer;
C++:	int GetPowerAmplifier (BOOL *pbAmplifier);
LabView:	
Parameter:	Amplifier: True – the power amplifiers are switched on False – switched off
Example:	LS. GetPowerAmplifier (&Amplifier);

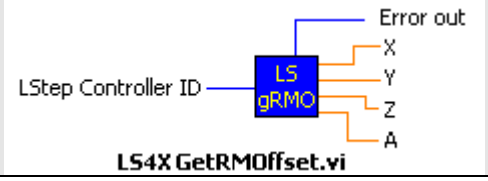
LS_SetPowerAmplifier	
Description:	Switches the power amplifiers of the LS44 On/Off. This command only exists for the LS44-controller.
Delphi:	function LS_SetPowerAmplifier (bAmplifier: LongBool): Integer; function LSX_ SetPowerAmplifier (LSID: Integer; bAmplifier: LongBool): Integer;
C++:	int SetPowerAmplifier (BOOL bAmplifier);
LabView:	
Parameter:	bAmplifier: True – On False – Off
Example:	LS. SetPowerAmplifier (True); // The power amplifiers are switched on

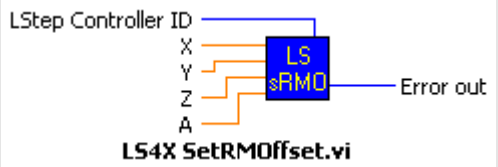
LS_GetReduction	
Description:	Inquiry of current reduction
Delphi:	function LS_GetReduction(var X, Y, Z, R: Double): Integer; function LSX_GetReduction(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetReduction (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Current Reduction
Example:	LS.GetReduction(&X, &Y, &Z, &A);

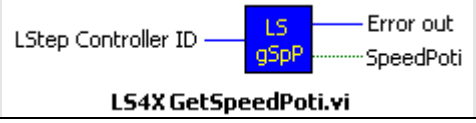
LS_SetReduction	
Description:	Set current reduction In quiescent state the rated motor current is reduced to the parameterized ratio.
Delphi:	function LS_SetReduction(X, Y, Z, R: Double): Integer; function LSX_SetReduction(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetReduction(double dX, double dY, double dZ, double dA);
LabView:	
Parameter:	X, Y, Z and A 0 – 1.0
Example:	LS.SetReduction(0.1, 0.7, 0.5, 0.5); /* Quiescent current for X-axis = 0.1*Rated current; Y-axis = 0.7*rated current ... */

LS_GetRefSpeed	
Description:	Reads the reverse speed, the axes move while searching the reference mark. The speed is equivalent to the issued value * 0.01 rp/s.
Delphi:	function LS_GetRefSpeed(var lSpeed: Integer): Integer; function LSX_ GetRefSpeed (LSID: Integer; var lSpeed: Integer): Integer;
C++:	int GetRefSpeed (int *plSpeed);
LabView:	
Parameter:	lSpeed: Speed value
Example:	LS. GetRefSpeed (&lSpeed);

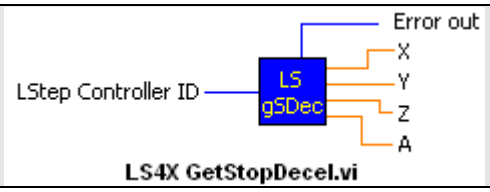
LS_SetRefSpeed	
Description:	Sets the reverse speed, the axes move while searching the reference mark. The speed is equivalent to the issued value * 0.01 rp/s.
Delphi:	function LS_SetRefSpeed(lSpeed: Integer): Integer; function LSX_ SetRefSpeed (LSID: Integer; lSpeed: Integer): Integer;
C++:	int SetRefSpeed (int lSpeed);
LabView:	
Parameter:	lSpeed: Speed , value range 0 to 100
Example:	LS. SetRefSpeed (10); //Speed while searching the reference mark is 0.1 rp/s.

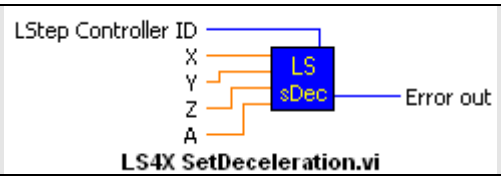
LS_GetRMOffset	
Description:	Inquiry RM-Offset
Delphi:	function LS_GetRMOffset(var X, Y, Z, A: Double): Integer; function LSX_GetRMOffset(LSID: Integer; var X, Y, Z, R: Double): Integer;
C++:	int GetRMOffset (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z and A: RM-Offset, depending on dimension
Example:	LS.GetRMOffset(&X, &Y, &Z, &A);

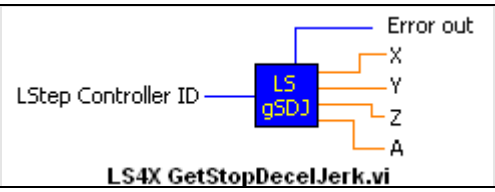
LS_SetRMOffset	
Description:	RM-Offset
Delphi:	function LS_SetRMOffset(X, Y, Z, A: Double): Integer; function LSX_SetRMOffset(LSID: Integer; X, Y, Z, R: Double): Integer;
C++:	int SetRMOffset (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, y, z and a 0 – 32*50000 (32*spindle pitch)
Example:	LS.SetRMOffset(1, 1, 1, 1); (When the table stroke is measured, the X, Y, and Z-axes are each moved 1 mm (for Dim 2 2 2) away from the end limit switch towards the center of the table and the software limit is then set.

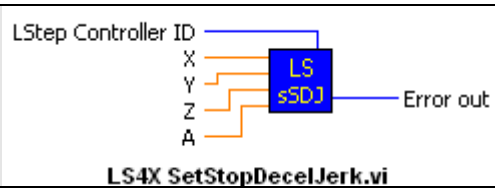
LS_GetSpeedPoti	
Description:	Indicates if the potentiometer On/Off
Delphi:	function LS_GetSpeedPoti(var SpePoti: LongBool): Integer; function LSX_GetSpeedPoti(LSID: Integer; var SpePoti: LongBool): Integer;
C++:	int GetSpeedPoti (BOOL *pbSpePoti);
LabView:	
Parameter:	The SpePoti Flag indicates, if the potentiometer is On or Off.
Example:	LS.GetSpeedPoti(&flag);

LS_SetSpeedPoti	
Description:	Potentiometer On/Off
Delphi:	function LS_SetSpeedPoti(SpeedPoti: LongBool): Integer; function LSX_SetSpeedPoti(LSID: Integer; SpeedPoti: LongBool): Integer;
C++:	int SetSpeedPoti (BOOL SpeedPoti);
LabView:	
Parameter:	If SpeedPoti = false, the preset speed (vel) is used as the speed of travel. If SpeedPoti = true, travelling is done at a percentage of the preset speed (vel), depending on the setting of the potentiometer.
Example:	LS.SetSpeedPoti(true); // Poti On

LS_GetStopDecel	
Description:	Stop input query deceleration
Delphi:	function LS_GetStopDecel (var XD, YD, ZD, AD: Double): Integer; function LSX_GetStopDecel (LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetStopDecel (double *pdXD, double *pdYD, double *pdZD, double *pdAD);
LabView:	
Parameter:	XD, YD, ZD, AD: delay values [m/s²]
Example:	LS.GetStopDecel (&XD, &YD, &ZD, &AD);

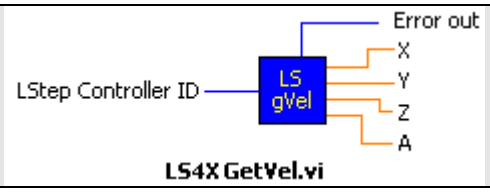
LS_SetStopDecel	
Description:	Adjust the value, with which the axes in case of a Stop signal shall decelerate
Delphi:	function LS_SetStopDecel (XD, YD, ZD, AD: Double): Integer; function LSX_ SetStopDecel (LSID: Integer; XD, YD, ZD, AD: Double): Integer;
C++:	int SetStopDecel (double dXD, double dYD, double dZD, double dAD);
LabView:	
Parameter:	XD, YD, ZD and AD: delay values 0.01 – 20.00 [m/s2]
Example:	LS. SetStopDecel (1.0, 1.5, 0, 0);

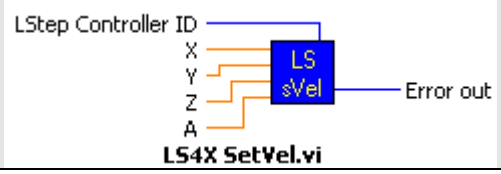
LS_GetStopDecelJerk	
Description:	Ask jerk during delay of system in case of Emergency Stop signal
Delphi:	function LS_GetStopDecelJerk (var XD, YD, ZD, AD: Double): Integer; function LSX_GetStopDecelJerk (LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetStopDecelJerk (double *pdXD, double *pdYD, double *pdZD, double *pdAD);
LabView:	
Parameter:	XD, YD, ZD, AD: Jerk values [m/s³]
Example:	LS.GetStopDecelJerk (&XD, &YD, &ZD, &AD);

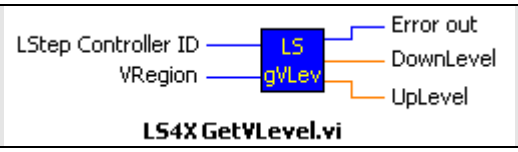
LS_SetStopDecelJerk	
Description:	Adjust jerk during delay of system in case of Emergency Stop signal
Delphi:	function LS_SetStopDecelJerk (XD, YD, ZD, AD: Double): Integer; function LSX_ SetStopDecelJerk (LSID: Integer; XD, YD, ZD, AD: Double): Integer;
C++:	int SetStopDecelJerk (double dXD, double dYD, double dZD, double dAD);
LabView:	
Parameter:	XD, YD, ZD and AD: 1, 2, 3, ... ∞ [m/s³] (Natural numbers)
Example:	LS. SetStopDecelJerk (1.0, 1.5, 0, 0);

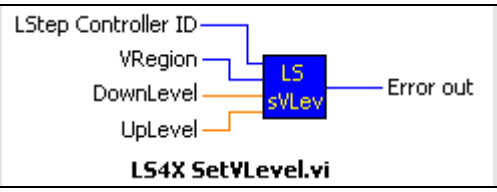
LS_GetStopPolarity	
Description:	Read stop input polarity.
Delphi:	function LS_GetStopPolarity(var bHighActiv: LongBool): Integer; function LSX_ GetStopPolarity (LSID: Integer; var bHighActiv: LongBool): Integer;
C++:	int GetStopPolarity (BOOL *pbHighActiv);
LabView:	
Parameter:	bHighActiv: True – Stop input high active False – low active
Example:	LS. GetStopPolarity (&HighActiv);

LS_SetStopPolarity	
Description:	Set stop input polarity. Because the stop entrance has a Pull Up after 5V, a closer has to be set low active and an opener high active.
Delphi:	function LS_SetStopPolarity(bHighActiv: LongBool): Integer; function LSX_ SetStopPolarity (LSID: Integer; bHighActiv: LongBool): Integer;
C++:	int SetStopPolarity (BOOL bHighActiv);
LabView:	
Parameter:	bHighActiv: True – Stop input high active False – low active
Example:	LS. SetStopPolarity (False); // The stop input is low active

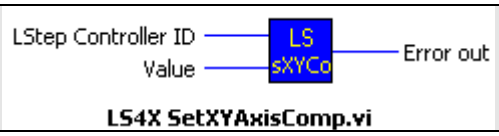
LS_GetVel	
Description:	Inquiry speed
Delphi:	function LS_GetVel(var X, Y, Z, R: Double): Integer; function LSX_GetVel(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetVel (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Speed values [rp/s]
Example:	LS.GetVel(&X, &Y, &Z, &A);

LS_SetVel	
Description:	Set speed (velocity)
Delphi:	function LS_SetVel(X, Y, Z, R: Double): Integer; function LSX_SetVel(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetVel(double dX, double dY, double dZ, double dA);
LabView:	
Parameter:	X, Y, Z and A 0 – maximum speed [r/s]
Example:	LS.SetVel(1.0, 15.0, 0, 0);

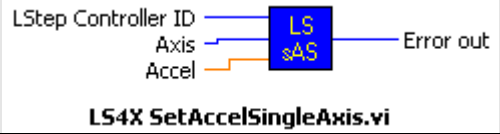
LS_GetVLevel	
Description:	Delivers the speed limits of the indicated speed range.
Delphi:	function LS_GetVLevel(lVRegion: Integer; var dDownLevel, dUppLevel: Double): Integer; function LSX_ GetVLevel (LSID: Integer; lVRegion: Integer; var dDownLevel, dUppLevel: Double): Integer;
C++:	int GetVLevel (int lVRegion, double *pdDownLevel, double *pdUppLevel);
LabView:	
Parameter:	lVRegion: Value range 1-4. 1 – First/lowest speed range 2 – Second/middle speed range 3 – Third/highest speed range 4 – Up to this speed limit a correction table is used. dDownLevel : Lower limit of the range(for lVRegion = 4 speed limit) [rp/s] dUppLevel : Upper limit of the range (for lVRegion = 4 has no meaning) [rp/s]
Example:	LS. GetVLevel (2, &DownLevel, &UppLevel); // DownLevel = Lower limit of the second speed range, UppLevel = Upper limit of the second speed range.

LS_SetVLevel	
Description:	Exclude speed ranges, in which the system shows resonances.
Delphi:	function LS_SetVLevel(lVRegion: Integer; dDownLevel, dUppLevel: Double): Integer; function LSX_ SetVLevel (LSID: Integer; lVRegion: Integer; dDownLevel, dUppLevel: Double): Integer;
C++:	int SetVLevel (int lVRegion, double dDownLevel, double dUppLevel);
LabView:	
Parameter:	lVRegion: Value range 1-4. 1 – First/lowest speed range 2 – Second/middle speed range 3 – Third/highest speed range 4 – Up to this speed limit a correction table is used. dDownLevel : Lower limit of the range(for lVRegion = 4 speed limit) [rp/s], value range 0 – max. speed. dUppLevel : Upper limit of the range (for lVRegion = 4 has no meaning) [rp/s], value range 0 – max. speed.
Example:	LS. SetVLevel (4, 10.0, 0.0); //The correction table is active to a speed of 10 rp/s.

LS_GetXYAxisComp	
Description	Inquiry XY-axis overlay
Delphi	function LS_GetXYAxisComp(var Value: Integer): Integer; function LSX_GetXYAxisComp(LSID: Integer; var Value: Integer): Integer;
C++	int GetXYAxisComp (int *plValue);
LabView:	
Parameter	Value: Mode for axis overlay (see LStep-documentation)
Example	LS.SetXYAxisComp(&mode) ;

LS_SetXYAxisComp	
Description	activate XY-axis overlay
Delphi	function LS_SetXYAxisComp(Value: Integer): Integer; function LSX_SetXYAxisComp(LSID: Integer; Value: Integer): Integer;
C++	int SetXYAxisComp (int lValue);
LabView:	
Parameter	Value: Mode for axis overlay (see LStep-documentation)
Example	LS.SetXYAxisComp(1) ;

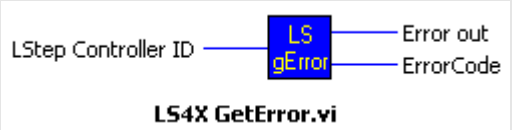
LS_LstepSave	
Description	Save current configuration in LStep (EEPROM)
Delphi	function LS_LStepSave(): Integer; function LSX_LStepSave(LSID: Integer): Integer;
C++	int LStepSave ();
LabView:	
Parameter	-
Example	LS.LStepSave() ;

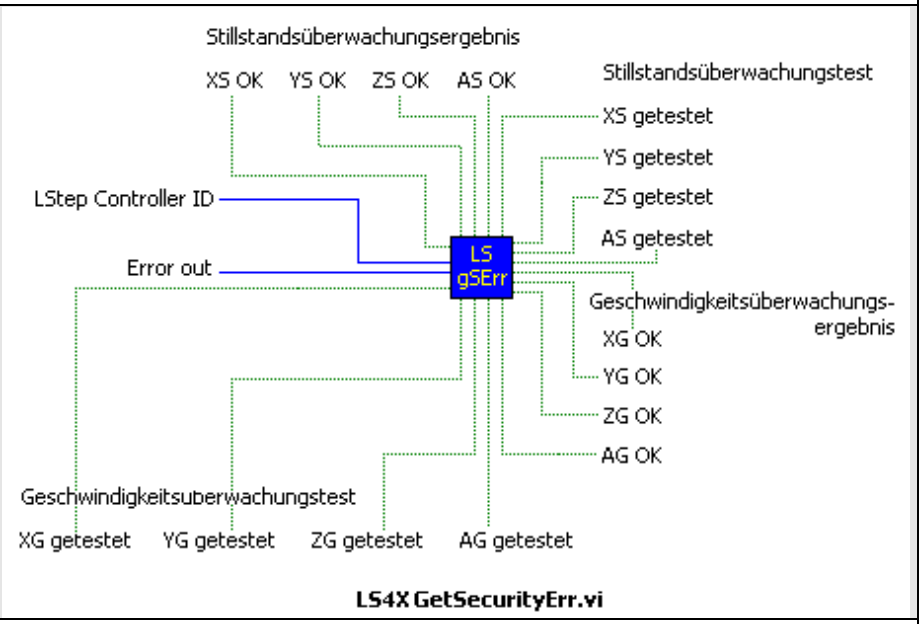
LS_SetAccelSingleAxis	
Description:	Set acceleration
Delphi:	function LS_SetAccelSingleAxis(Axis: Integer; Accel: Double): Integer; function LSX_SetAccelSingleAxis(LSID: Integer; Axis: Integer; Accel: Double): Integer;
C++:	int SetAccelSingleAxis (int lAxis,double dAccel);
LabView:	
Parameter:	Axis: (X, Y, Z, A numbered from 1 to 4) Accel: Acceleration 0.01 – 10.00 [m/s²]
Example:	LS.SetAccelSingleAxis(4, 1.0); // Accelerate A-axis 1.0 m/s²

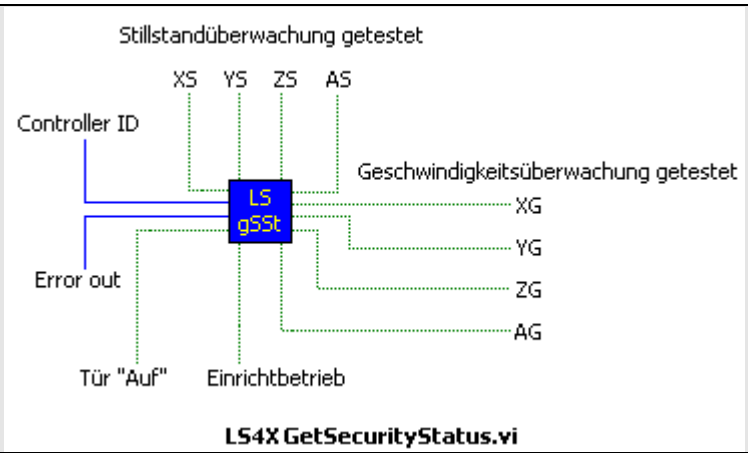
LS_SetVelSingleAxis	
Description:	Set speed for individual axis
Delphi:	function LS_SetVelSingleAxis(Axis: Integer; Vel: Double): Integer; function LSX_SetVelSingleAxis(LSID: Integer; Axis: Integer; Vel: Double): Integer;
C++:	int SetVelSingleAxis (int lAxis,double dVel);
LabView:	
Parameter:	Axis: (X, Y, Z, A numbered from 1 to 4) Vel: 0 – maximum speed [r/s]
Example:	LS.SetVelSingleAxis(1, 10.0) // Speed of X-axis 10 r/s

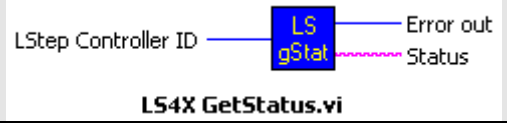
LS_SoftwareReset	
Description:	Reset the software to starting status.
Delphi:	function LS_SoftwareReset: Integer; function LSX_SoftwareReset(LSID: Integer): Integer;
C++:	int SoftwareReset ();
LabView:	
Parameter:	-
Example:	LS.SoftwareReset ();

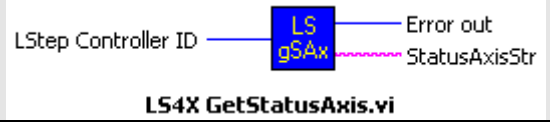
Status request

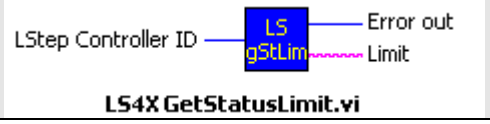
LS_GetError	
Description:	shows the current error number
Delphi:	function LS_GetError(var ErrorCode: Integer): Integer; function LSX_GetError(LSID: Integer; var ErrorCode: Integer): Integer;
C++:	int GetError (int *plErrorCode);
LabView:	
Parameter:	ErrorCode: Error number
Example:	LS.GetError(&ErrCode);

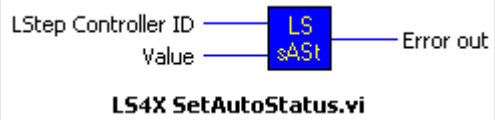
LS_GetSecurityErr	
Description:	Reads all statuses and results of the GAL-safety monitoring (only with LS44-controller)
Delphi:	function LS_ Get SecurityErr (var Value: LongWord): Integer; function LSX_ GetS SecurityErr (LSID: Integer; var Value: LongWord): Integer;
C++:	int GetSecurityErr (LongWord *pValue);
LabView:	
Parameter:	Value: 32-Bit LongWord without preceding sign, which contains the bit mask in the bits 0-15 after activating the function. Bit 0 X-axis standstill monitoring result (OK [1] / not OK [0]) Bit 1 Y-axis standstill monitoring result Bit 2 Z-axis standstill monitoring result Bit 3 A-axis standstill monitoring result Bit 5 X-axis standstill monitoring test (tested [1] /not tested [0]) Bit 6 Y-axis standstill monitoring test Bit 7 Z-axis standstill monitoring test Bit 8 A-axis standstill monitoring test Bit 9 X-axis standstill monitoring result Bit 10 Y-axis standstill monitoring result Bit 11 Z-axis standstill monitoring result Bit 12 A-axis standstill monitoring result Bit 13 X-axis standstill monitoring test Bit 14 Y-axis standstill monitoring test Bit 15 Z-axis standstill monitoring test
Example:	LS.GetSecurityErr (&Value);

LS_GetSecurityStatus	
Description:	Delivers the current statuses the safety monitoring (only with LS44-controller)
Delphi:	function LS_ Get SecurityStatus (var Value: LongWord): Integer; function LSX_ GetS SecurityStatus (LSID: Integer; var Value: LongWord): Integer;
C++:	int GetSecurityStatus (LongWord *pValue);
LabView:	
Parameter:	Value: 32-Bit LongWord without preceding sign, which contains the bit mask in the bits 0-15 after activating the function. Bit 0-3 internal memory Bit 4 X-axis standstill monitoring tested Bit 5 Y-axis standstill monitoring tested Bit 6 Z-axis standstill monitoring tested Bit 7 A-axis standstill monitoring tested Bit 8 X-axis speed monitoring tested Bit 9 Y-axis speed monitoring tested Bit 10 Z-axis speed monitoring tested Bit 11 A-axis speed monitoring tested Bit 14 Condition setup mode (setup mode = 1) Bit 15 Condition door (door „Open“ = 1)
Example:	LS.GetSecurityStatus (&Value);

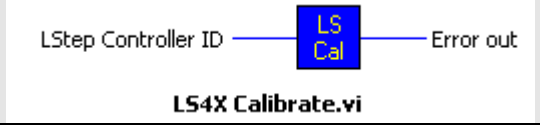
LS_GetStatus	
Description:	Gives the current status of the controller.
Delphi:	function LS_GetStatus(Stat: PChar; MaxLen: Integer): Integer; function LSX_GetStatus(LSID: Integer; Stat: PChar; MaxLen: Integer): Integer;
C++:	int GetStatus (char *pcStat,int lMaxLen);
LabView:	
Parameter:	Stat: Pointer to a buffer in which the status string is returned MaxLen: Maximum number of characters, that can be copied into the buffer
Example:	LS.GetStatus(pcStat, 256);

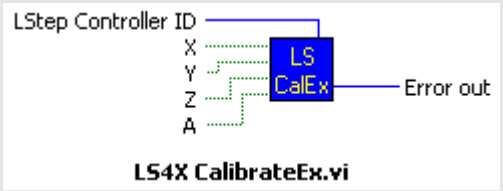
LS_GetStatusAxis	
Description:	Gives the present status of the individual axes
Delphi:	function LS_GetStatusAxis(StatusAxisStr: PChar; MaxLen: Integer): Integer; function LSX_GetStatusAxis(LSID: Integer; StatusAxisStr: PChar; MaxLen: Integer): Integer;
C++:	int GetStatusAxis (char *pcStatusAxisStr,int lMaxLen);
LabView:	
Parameter:	StatusAxisStr: Pointer to a buffer in which the status string is returned MaxLen: Maximum number of characters, that can be copied into the buffer e.g.: @ - M – J – C – S – A – D – U - T @ = Axis standing M = Axis is moving (Motion) - = Axis is not enabled J = Joystick switched on C = Axis in control A = feedback after calibration E = Fault during calibration (Limit switch was not set free correctly) D = feedback after table stroke measuring U = Set up mode T = Timeout
Example:	LS.GetStatusAxis(pcStatAxis, 256);

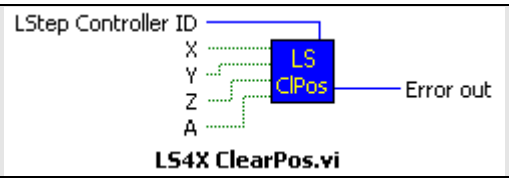
LS_GetStatusLimit	
Description:	Delivers the current condition of the software-limits of each axis.
Delphi:	function LS_ GetStatusLimit (Limit: PChar; MaxLen: Integer): Integer; function LSX_ GetStatusLimit (LSID: Integer; Limit: PChar; MaxLen: Integer): Integer;
C++:	int GetStatusLimit (char *pcLimit, int lMaxLen);
LabView:	
Parameter:	pc Limit: Pointer to a buffer, in which the condition of the axis is returned to. E.g.: AA- A- - DD - LL- L- - L A = Axis was calibratet D = table stroke was measured L = Software-limit was set - = Software-limit was not changed MaxLen: Maximum number of characters, that can be copied into the buffer
Example:	LS.GetStatusLimit (pc Limit, 64);

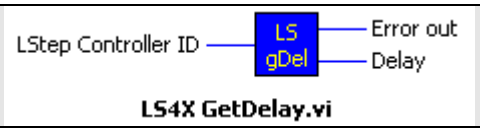
LS_SetAutoStatus	
Description:	AutoStatus On/Off
	Information: The AutoStatus mode should not normally be changed, as LSTEP API sets the correct mode for travel commands etc. Changing to 0 or 2 could pitch to errors
Delphi:	function LS_SetAutoStatus(Value: Integer): Integer; function LSX_SetAutoStatus(LSID: Integer; Value: Integer): Integer;
C++:	int SetAutoStatus (int lValue);
LabView:	
Parameter:	Value: AutoStatus mode:
	0 ➔ No status is transmitted by the controller. 1 ➔ "Position reached" signals are sent automatically by the controller. 2 ➔ "Position reached" and status messages are sent automatically by the controller. 3 ➔ For "Position reached" only a Carriage Return is returned.
Example:	LS.SetAutoStatus(3);

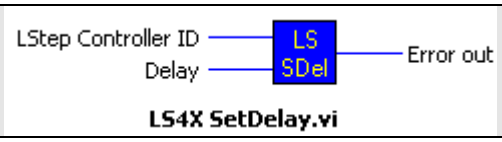
Moving commands and position administration

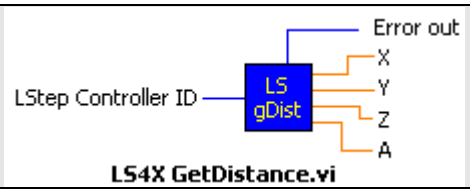
LS_Calibrate	
Description:	Calibrate
	Moves all enabled axes towards lower position values. The movements are interrupted as soon as the limit switch is reached. The postion values are set to 0.
Delphi:	function LS_Calibrate: Integer; function LSX_Calibrate(LSID: Integer): Integer;
C++:	int Calibrate();
LabView:	
Parameter:	-
Example:	LS.Calibrate();

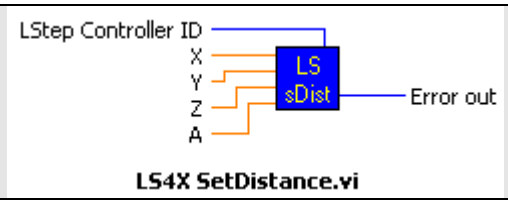
LS_CalibrateEx	
Description:	Calibrate
	(Only the axes are calibrated, whose corresponding bit was set in the transmitted integer-value)
Delphi:	function LS_CalibrateEx(Flags: Integer): Integer; function LSX_CalibrateEx(LSID: Integer; Flags: Integer): Integer;
C++:	int CalibrateEx (int lFlags);
LabView:	
Parameter:	Flags: Bit mask, Bit 2 = 1 → Calibrate Z-axis Bit 2 = 0 → Do not calibrate Z-axis ...
Example:	LS.CalibrateEx(6); // Calibrate only Y- and Z-axis

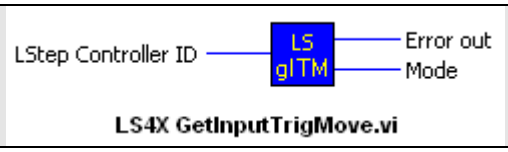
LS_ClearPos	
Description:	Sets the position to 0, also the internal counter. This function is needed for endless axes, because the control can process only ±1000 motor revolutions of the value range. In recognised encoders, the function is not carried out for corresponding axis.
Delphi:	function LS_ClearPos (lFlags: Integer): Integer; function LSX_ClearPos (LSID: Integer; lFlags: Integer): Integer;
C++:	int ClearPos (int lFlags);
LabView:	
Parameter:	lFlags: Bit mask Bit 0 = 1 → Position of the x-axis is zeroized Bit 1 = 0 → For the y-axis the function is not carried out
Example:	LS. ClearPos(5); //Positions of the x- and z- axes are zeroized.

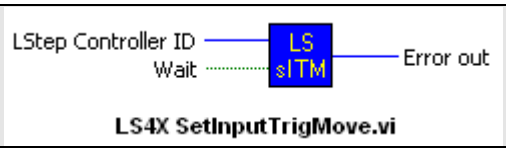
LS_GetDelay	
Description:	Reads the delay of the vector start.
Delphi:	function LS_GetDelay(var Delay: Integer): Integer; function LSX_GetDelay(LSID: Integer; var Delay: Integer): Integer;
C++:	int GetDelay (int *plDelay);
LabView:	
Parameter:	Delay: in ms
Example:	LS.GetDelay(&Delay);

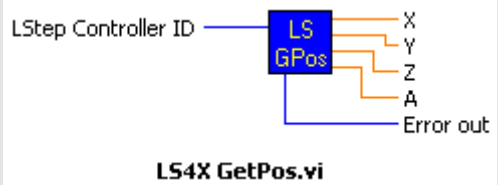
LS_SetDelay	
Description:	The delay command is used to produce a vector start delay.
Delphi:	function LS_SetDelay(Delay: Integer): Integer; function LSX_SetDelay(LSID: Integer; Delay: Integer): Integer;
C++:	int SetDelay (int lDelay);
LabView:	
Parameter:	0 – 10000 (ms)
Example:	LS.SetDelay(1000); // 1s delay

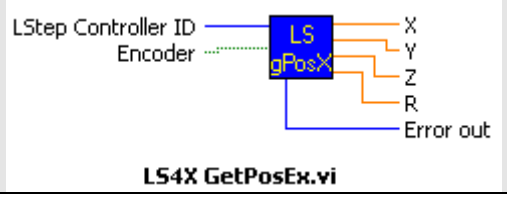
LS_GetDistance	
Description:	Delivers the distance for LS_MoveRelShort
Delphi:	function LS_GetDistance(var X, Y, Z, A: Double): Integer; function LSX_GetDistance(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetDistance (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z and A: the current distances of all axes, dependent on the dimensions.
Example:	LS.GetDistance(&X, &Y, &Z, &A);

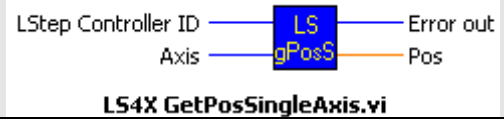
LS_SetDistance	
Description:	Set distance (for LS_MoveRelShort)
Delphi:	function LS_SetDistance(X, Y, Z, A: Double): Integer; function LSX_SetDistance(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetDistance (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, Y, Z and A Min./max. range of travel (Values depend on the dimension)
Example:	LS.SetDistance(1, 2, 0, 0); /* Distances are set for the X-, and Y-axes, Z and A are not moved when the function LS_MoveRelShort is called. */

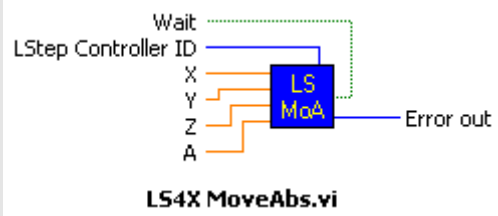
LS_GetInputTrigMove	
Description:	Supplies the configuration of Pin1 on the MFP.
Delphi:	function LS_GetInputTrigMove (var Mode: Integer): Integer; function LSX_GetInputTrigMove (LSID: Integer; var Mode: Integer): Integer;
C++:	int GetInputTrigMove (int *plMode);
LabView:	
Parameter:	**lMode** – Mode. lMode = 0 → Function not active lMode = 1 → absolute positioning at positive edge lMode = 2 → absolute positioning at negative edge lMode = 3 → relative positioning at positive edge lMode = 4 → relative positioning at negative edge
Example:	LS. GetInputTrigMove (&lMode);

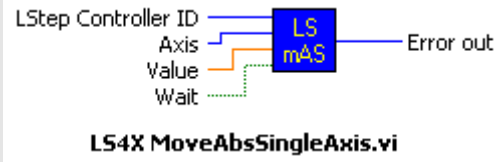
LS_SetInputTrigMove	
Description:	Configurate the Pin 1 on the MFP, so that you can start with an external signal a Move.
Delphi:	function LS_SetInputTrigMove (Mode: Integer; Wait: LongBool): Integer; function LSX_SetInputTrigMove (LSID: Integer; Mode: Integer; Wait: LongBool): Integer;
C++:	int SetInputTrigMove (int lMode, BOOL bWait);
LabView:	
Parameter:	**lMode** – Mode. lMode = 0 → Function not active lMode = 1 → absolute positioning at positive edge lMode = 2 → absolute positioning at negative edge lMode = 3 → relative positioning at positive edge lMode = 4 → relative positioning at negative edge Positioned will be the value in "distance". **bWait** – Waiting for a Move. bWait = 1 → you have to wait so long until, after an external signal, a Move will be executed. Then the Mode will placed to 0. bWait = 0 → you don't wait for a Move. bWait will not be interpreted when lMode = 0.
Example:	LS. SetInputTrigMove (3, False);

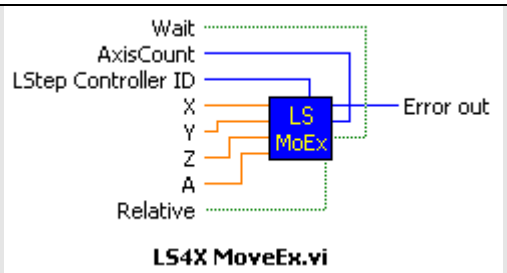
LS_GetPos	
Description:	Inquires the current positions of all axes
	For non-existing axes, a value of 0.0 is returned
Delphi:	function LS_GetPos(var X, Y, Z, A: Double): Integer; function LSX_GetPos(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetPos (double *pdX,double *pdY,double *pdZ,double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Position values
Example:	double X, Y, Z, A; LS.GetPos(&X, &Y, &Z, &A);

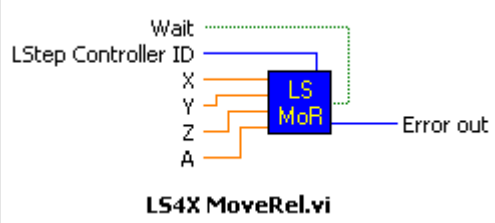
LS_GetPosEx	
Description:	Inquires the current encoder or position values of all axes
	For non-existing axes, a value of 0.0 is returned
Delphi:	function LS_GetPosEx(var X, Y, Z, A: Double; Encoder: LongBool): Integer; function LSX_GetPosEx(LSID: Integer; var X, Y, Z, R: Double; Encoder: LongBool): Integer;
C++:	int GetPosEx (double *pdX,double *pdY,double *pdZ,double *pdA,BOOL Encoder);
LabView:	
Parameter:	X, Y, Z, A: Position values Encoder = true → Shows encoder values, if any encoder is connected Encoder = false → Shows position values
Example:	double X, Y, Z, A; LS.GetPosEx(&X, &Y, &Z, &A, true);

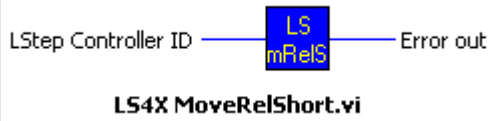
LS_GetPosSingleAxis	
Description:	Inquire the current position of an axis
	For non-existing axes, a value of 0.0 is returned
Delphi:	function LS_GetPosSingleAxis(Axis: Integer; var Pos: Double): Integer; function LSX_GetPosSingleAxis(LSID: Integer; Axis: Integer; var Pos: Double): Integer;
C++:	int GetPosSingleAxis (int lAxis,double *pdPos);
LabView:	
Parameter:	Axis: Axis for which the position value is to be inquired (X, Y, Z, A numbered from 1 to 4) Pos: position value
Example:	LS.GetPosSingleAxis(2, &YPos); // Read position of Y-axis

LS_MoveAbs	
Description:	Move to absolute position
	(The X-. Y-, and Z- axes are positioned at the transmitted position values.)
Delphi:	function LS_MoveAbs(X, Y, Z, A: Double; Wait: LongBool): Integer; function LSX_MoveAbs(LSID: Integer; X, Y, Z, A: Double; Wait: LongBool): Integer;
C++:	int MoveAbs (double dX, double dY, double dZ, double dA, BOOL Wait);
LabView:	
Parameter:	X, Y, Z and A +- Moving range Input depends on the set dimension Wait: Specifies whether the function should return after the position has been reached (= true) or directly (= false)
Example:	LS.MoveAbs(10.0, 10.0, 10.0, 10.0, true);

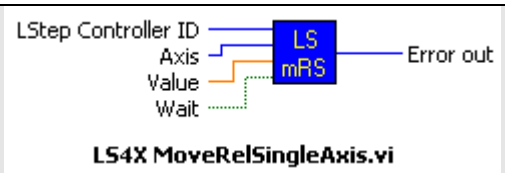
LS_MoveAbsSingleAxis	
Description:	Position single axis absolute
Delphi:	function LS_MoveAbsSingleAxis(Axis: Integer; Value: Double; Wait: LongBool): Integer; function LSX_MoveAbsSingleAxis(LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;
C++:	int MoveAbsSingleAxis (int lAxis,double dValue,BOOL Wait);
LabView:	
Parameter:	Axis: (X, Y, Z, A numbered from 1 to 4) Value: Position (Input depends on the set dimension)
Example:	LS.MoveAbsSingleAxis(2, 10.0); // Move Y-axis to 10mm absolute position

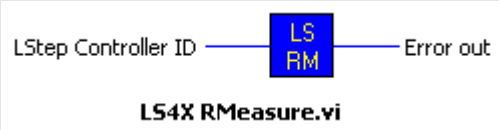
LS_MoveEx	
Description	extended moving command The function LS_MoveEx can carry out relative and absolute move commands, synchronically and asynchronous. The number of axes that are to be moved can be defined with the parameter AxisCount . This function can be used, for example, to move only X and Y on an LStep44.
Delphi	function LS_MoveEx(X, Y, Z, R: Double; Relative, Wait: LongBool; AxisCount: Integer): Integer; function LSX_MoveEx(LSID: Integer; X, Y, Z, R: Double; Relative, Wait: LongBool; AxisCount: Integer): Integer;
C++	int MoveEx (double dX, double dY, double dZ, double dR, BOOL bRelative, BOOL bWait, int lAxisCount);
LabView:	
Parameter	X, Y, Z, R : Position-vector Relative: with relative=false all values X, Y, Z and R are interpreted as absolute co-ordinates. Wait: is Wait=true, the function returns only after reaching the target position otherwize it returns directly after sending the command to the LStep. AxisCount: Amount of axes, that should move Is AxisCount=1, only X moves Is AxisCount=2, X and Y move...
Example	LS_MoveEx(2.0, 3.0, 0, 0, true, true, 2) ; // X and Y are moved relative by 2 resp. 3

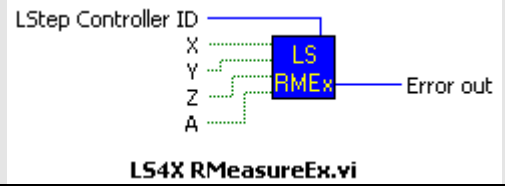
LS_MoveRel	
Description:	Move relative vector
	(The X-, Y-, and Z-axes are moved the transmitted distances.)
Delphi:	function LS_MoveRel(X, Y, Z, A: Double; Wait: LongBool): Integer; function LSX_MoveRel(LSID: Integer; X, Y, Z, A: Double; Wait: LongBool): Integer;
C++:	int MoveRel (double dX, double dY, double dZ, double dA, BOOL Wait);
LabView:	
Parameter:	X, Y, Z and A +- Moving range Input depends on the set dimension Wait: Specifies whether the function should return after the position has been reached (= true) or directly (= false)
Example:	LS.MoveRel(10.0, 10.0, 10.0, 10.0, true);

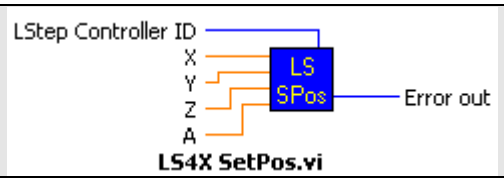
LS_MoveRelShort	
Description:	Move to relative position (short command)
	This command should be used, so that a series of consecutive relative travel commands (of the same distance) are approached more quickly. The distance must have been set beforehand with LS_SetDistance .
Delphi:	function LS_MoveRelShort: Integer; function LSX_MoveRelShort(LSID: Integer): Integer;
C++:	int MoveRelShort ();
LabView:	
Parameter:	-

Example:	LS.SetDistance(1.0, 1.0, 0, 0);
	for (i = 0; i < 10; i++) LS.MoveRelShort();
	// Move the X- and Y- axes 10times 1 mm to the relative position

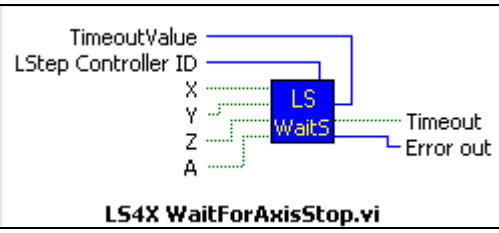
LS_MoveRelSingleAxis	
Description:	Move single axis absolute
Delphi:	function LS_MoveRelSingleAxis(Axis: Integer; Value: Double; Wait: LongBool): Integer; function LSX_MoveRelSingleAxis(LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;
C++:	int MoveRelSingleAxis (int lAxis,double dValue,BOOL Wait);
LabView:	
Parameter:	Axis: (X, Y, Z, A numbered from 1 to 4) Value: Stretch (Input depends on the set dimension)
Example:	LS.MoveRelSingleAxis(3, 5.0); // Move Z-axis 5mm in positive direction

LS_RMeasure	
Description:	Measure Table Stroke
	Moves all enabled axes towards greater position values. The movements are interrupted as soon as the limit switch is reached. The position value is saved.
Delphi:	function LS_RMeasure: Integer; function LSX_RMeasure(LSID: Integer): Integer;
C++:	int RMeasure();
LabView:	
Parameter:	-
Example:	LS.RMeasure();

LS_RMeasureEx	
Description:	Measure Table Stroke (The table stroke is only measured for axes for which the relevant bit has been set in the transmitted integer value.
Delphi:	function LS_RMeasureEx(Flags: Integer): Integer; function LSX_RMeasureEx(LSID: Integer; Flags: Integer): Integer;
C++:	int RMeasureEx (int lFlags);
LabView:	
Parameter:	Flags: Bit mask, Bit 2 = 1 → Calibrate Z-axis Bit 2 = 0 → Do not calibrate Z-axis ...
Example:	LS.RMeasureEx(2); // Measure table stroke (Y-axis only)

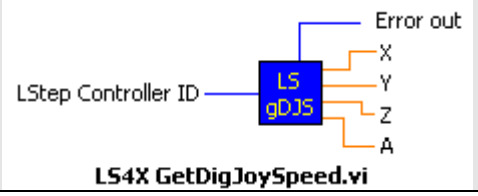
LS_SetPos	
Description:	Set position values
Delphi:	function LS_SetPos(X, Y, Z, R: Double): Integer; function LSX_SetPos(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetPos(double dX, double dY, double dZ, double dA);
LabView:	
Parameter:	X, Y, Z and A Min./max. range of travel Input depends on the set dimension
Example:	LS.SetPos(10, 10, 0, 0);

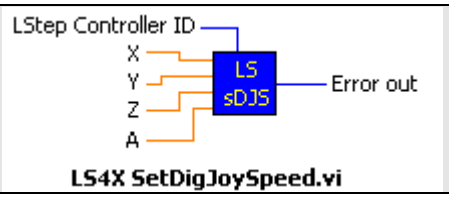
LS_StopAxes	
Description	Abort (All movements are stopped)
Delphi	function LS_StopAxes: Integer; function LSX_StopAxes(LSID: Integer): Integer;
C++	int StopAxes ();
LabView	
Parameter	-
Example	LS.StopAxes();

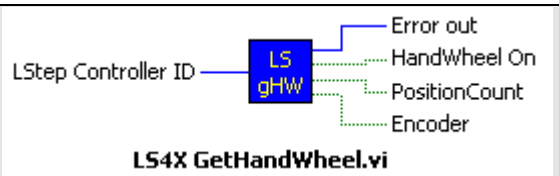
LS_WaitForAxisStop	
Description	The function returns, as soon as the selected axes in the bit-mask AFlags reached its goal position. LS_WaitForAxisStop uses ‚?statusaxis', to pollen the status of the axes.
Delphi	function LS_WaitForAxisStop(AFlags: Integer; ATimeoutValue: Integer; var ATimeout: LongBool): Integer; function LSX_WaitForAxisStop(LSID: Integer; AFlags: Integer; ATimeoutValue: Integer; var ATimeout: LongBool): Integer;
C++	int WaitForAxisStop (int lAFlags, int lATimeoutValue, BOOL *pbATimeout);
LabView:	
Parameter	AFlags: Bit mask Bit 0: X-axis Bit 1: Y-axis Bit 2: Z-axis Bit 3: A-axis AtimeoutValue: Timeout in milliseconds, WaitForAxisStop returns after this time with Atimeout=true, if the axes are still in motion AtimeoutValue = 0 set the timeout to 'endless' The Atimeout Flag indicates, if a timeout occurred.

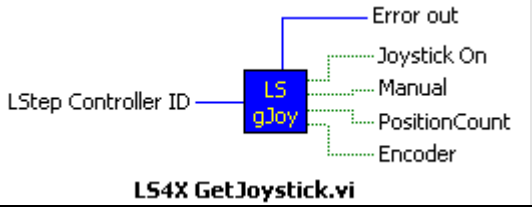
Example	LS.WaitForAxisStop(3, 0, flag);
	// Wating until X and Y-Axis stopped, no timeout
	LS.WaitForAxisStop(7, 10000, flag);
	// Wating until X and Y-Axis stopped, 10 seconds timeout

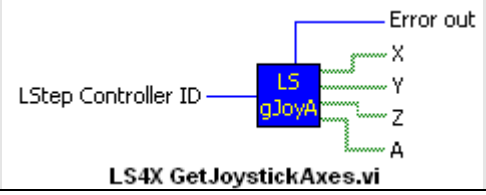
Joystick and Handwheel

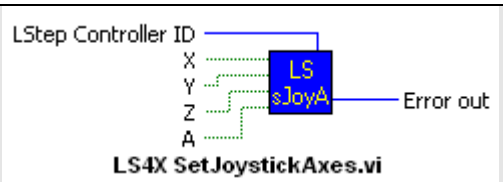
LS_GetDigJoySpeed	
Description:	Read out the set speeds
Delphi:	function LS_GetDigJoySpeed(var dX, dY, dZ, dR: Double): Integer; function LSX_GetDigJoySpeed(LSID: Integer; var dX, dY, dZ, dR: Double): Integer;
C++:	int GetDigJoySpeed (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	dX, dY, dZ, dR: Speed values [rp/s]
Example:	LS. GetDigJoySpeed(&X, &Y, &Z, &R);

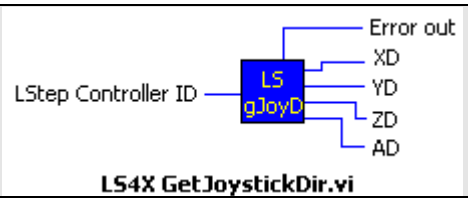
LS_SetDigJoySpeed	
Description:	With this command, single axes can be operated with a constant speed. If the positioning should be done absolutely or relatively after carrying out this function, the speed needs to be set new.
Delphi:	function LS_SetDigJoySpeed(dX, dY, dZ, dR: Double): Integer; function LSX_SetDigJoySpeed(LSID: Integer; dX, dY, dZ, dR: Double): Integer;
C++:	int SetDigJoySpeed (double dX, double dY, double dZ, double dR);
LabView:	
Parameter:	dX, dY, dZ, dR: Speed [rp/s], Value range: +- max. speed
Example:	LS. SetDigJoySpeed(0, 10.0, 25.0, 0); //axes X and R – speed 0 and Joystick-operation „OFF“, Axis Y – speed 10.0 rp/s and Joystick-operation „ON“, Axes Z – speed 25.0 rp/s and Joystick-operation „ON“.

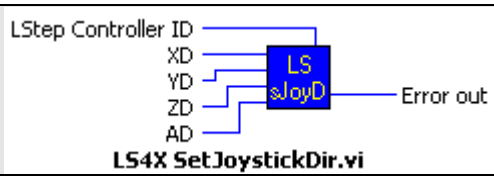
LS_GetHandWheel	
Description:	Read hand wheel condition
Delphi:	function LS_GetHandWheel(var PositionCount, Encoder: Boolean): Integer; function LSX_GetHandWheel(LSID: Integer; var PositionCount, Encoder: LongBool): Integer;
C++:	int GetHandWheel (BOOL *pbHandWheelOn, BOOL *pbPositionCount, BOOL *pbEncoder);
LabView:	
Parameter:	HWOn: True = Hand wheel is switched on PosCount: True = Position counter is switched on Encoder: True = Encoder values, if available
Example:	LS. GetHandWheel (&HWOn, &PosCount, &Encoder);

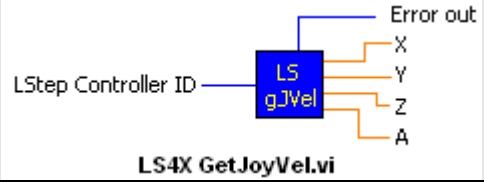
LS_GetJoystick	
Description:	Inquiry of the current condition of the analogy-Joystick.
Delphi:	function LS_GetJoystick (var JoystickOn, Manual, PositionCount, Encoder: LongBool): Integer; function LSX_ GetJoystick (LSID: Integer; var JoystickOn, Manual, PositionCount, Encoder: LongBool): Integer;
C++:	int GetJoystick (BOOL *pbJoystickOn, BOOL *pbManual, BOOL *pbPositionCount, BOOL *pbEncoder);
LabView:	
Parameter:	JoyOn: True = Joystick is switched on Manual: False = Joystick switch is set to automatic True = Joystick is manually switched on via switch PosCount: True = Position counter is switched on Enc: True = Encoder values, if available
Example:	LS. GetJoystick (&JoyOn, &Manual, &PosCount, &Enc);

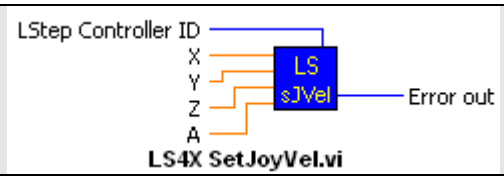
LS_GetJoystickAxes	
Description:	Shows, for which axes Joystick is switched on
Delphi:	function LS_GetJoystickAxes(var Flags: Integer): Integer; function LSX_GetJoystickAxes (LSID: Integer; var Flags: Integer): Integer;
C++:	int GetJoystickAxes (int *plFlags);
LabView:	
Parameter:	Flags: 32-bit-Integer, which contains after calling of the function in the Bits 0-4 the Bit-mask. Bit 0 = 1 → X-Axis Joystick „ON“ Bit 2 = 0 → Z-Axis Joystick „OUT“
Example:	LS. GetJoystickAxes (&Flags);

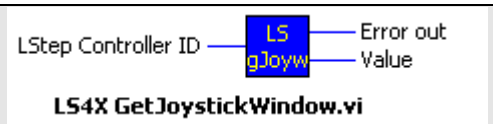
LS_SetJoystickAxes	
Description:	Switches Joystick for specified axes
Delphi:	function LS_ SetJoystickAxes (Flags: Integer): Integer; function LSX_ SetJoystickAxes (LSID: Integer; Flags: Integer): Integer;
C++:	int SetJoystickAxes (int Flags);
LabView:	
Parameter:	Flags: Bit mask Bit 0 = 1 → X-Axis switch on Joystick Bit 2 = 0 → Z-Axis switch out Joystick
Example:	LS. SetJoystickAxes (3); /* X- and Y-Axis – Joystick switched on (Bits 0 a. 1 placed), Z- and A-Axis – Joystick „OUT“ (Bit 2 = 0) */

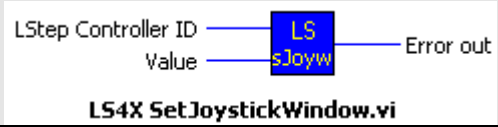
LS_GetJoystickDir	
Description:	Reads motor turning direction for joystick
Delphi:	function LS_GetJoystickDir(var XD, YD, ZD, AD: Integer): Integer; function LSX_GetJoystickDir(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;
C++:	int GetJoystickDir (int *plXD, int *plYD, int *plZD, int *plRD);
LabView:	
Parameter:	X, Y, Z, and A
	0 ➔ Axis disabled 1 ➔ Positive direction of rotation -1 ➔ Negative direction of rotation 2 ➔ with current reduction -2 ➔ with current reduction
Example:	LS.GetJoystickDir(&X, &Y, &Z, &A);

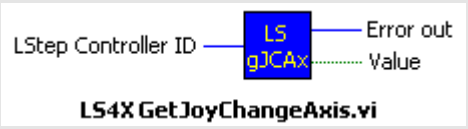
LS_SetJoystickDir	
Description:	Joystick direction
Delphi:	function LS_SetJoystickDir(XD, YD, ZD, AD: Integer): Integer; function LSX_SetJoystickDir(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;
C++:	int SetJoystickDir (int lXD,int lYD,int lZD,int lAD);
LabView:	
Parameter:	X, Y, Z, and A
	0 ➔ Axis disabled 1 ➔ Positive direction of rotation -1 ➔ Negative direction of rotation 2 ➔ with current reduction -2 ➔ with current reduction
Example:	LS.SetJoystickDir(1, 1, -1, 0); /* X- and Y-axis positive direction of rotation; Z-axis negative direction of rotation; A-axis disabled */

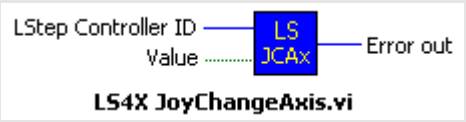
LS_GetJoyVel	
Description:	Ask for the max. positioning speeds in Joystick operation
Delphi:	function LS_GetJoyVel(var XD, YD, ZD, AD: Double): Integer; function LSX_GetJoyVel(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;
C++:	int GetJoyVel (double *pdXD, double *pdYD, double *pdZD, double *pdAD);
LabView:	
Parameter:	XD, YD, ZD, AD: Speed values [rp/s]
Example:	LS.GetJoyVel(&XD, &YD, &ZD, &AD);

LS_SetJoyVel	
Description:	Adjust the max. positioning speed in Joystick operation
Delphi:	function LS_SetJoyVel (XD, YD, ZD, AD: Double): Integer; function LSX_SetJoyVel (LSID: Integer; XD, YD, ZD, AD: Double): Integer;
C++:	int SetJoyVel (double dXD, double dYD, double dZD, double dAD);
LabView:	
Parameter:	XD, YD, ZD and AD: maximal speed in Joystick operation [R/s]
Example:	LS.SetJoyVel (1.0, 15.0, 0, 0);

LS_GetJoystickWindow	
Description	Read Joystick-window
Delphi	function LS_GetJoystickWindow(var AValue: Integer): Integer; function LSX_GetJoystickWindow(LSID: Integer; var AValue: Integer): Integer;
C++	int GetJoystickWindow (int *plAValue);
LabView:	
Parameter	AValue: the analogous range in which the axes do not move.
Example	LS.GetJoystickWindow(&AValue) ;

LS_SetJoystickWindow	
Description	Set joystick window
Delphi	function LS_SetJoystickWindow(AValue: Integer): Integer; function LSX_SetJoystickWindow(LSID: Integer; AValue: Integer): Integer;
C++	int SetJoystickWindow (int lAValue);
LabView:	
Parameter	AValue: 0-100
Example	LS.SetJoystickWindow(20) ;

LS_GetJoyChangeAxis	
Description:	Read Joystick allocation of the axes
Delphi:	LS_GetJoyChangeAxis(var Value: LongBool): Integer; LSX_GetJoyChangeAxis(LSID: Integer; var Value: LongBool): Integer;
C++:	int GetJoyChangeAxis (BOOL *pbValue);
LabView:	
Parameter:	Value: true => Conventional evaluation of Joystick false => allocation of X and Y axes have been exchanged
Example:	LS. GetJoyChangeAxis (&Value);

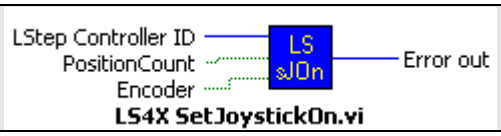
LS_JoyChangeAxis	
Description:	sets allocation of axes of Joystick
Delphi:	LS_JoyChangeAxis(Value: LongBool): Integer; LSX_JoyChangeAxis(LSID: Integer; Value: LongBool): Integer;
C++:	int JoyChangeAxis (BOOL bValue);
LabView:	
Parameter:	Value: 0 – Changes the allocation of the AD-Joystick channels (conventional evaluation of Joystick) 1 - Allocation of X and Y axes will be exchanged
Example:	LS.JoyChangeAxis (true);

LS_SetDigJoyOff	
Description:	Switches off digital Joystick
Delphi:	function LS_ SetDigJoyOff: Integer; function LSX_ SetDigJoyOff (LSID: Integer): Integer;
C++:	int SetDigJoyOff () ;
LabView:	
Parameter:	
Example:	LS. SetDigJoyOff () ;

LS_SetHandWheelOff	
Description:	Handwheel Off
Delphi:	function LS_SetHandWheelOff: Integer; function LSX_SetHandWheelOff(LSID: Integer): Integer;
C++:	int SetHandWheelOff ();
LabView:	
Parameter:	-
Example:	LS.SetHandWheelOff();

LS_SetHandWheelOn	
Description:	Handwheel On
Delphi:	function LS_SetHandWheelOn(PositionCount, Encoder: Boolean): Integer; function LSX_SetHandWheelOn(LSID: Integer; PositionCount, Encoder: LongBool): Integer;
C++:	int SetHandWheelOn (BOOL fPositionCount,BOOL fEncoder);
LabView:	
Parameter:	PositionCount: Position count On/Off Encoder: Encoder values (positions), if any
Example:	LS. SetHandWheelOn (true, true); // Handwheel On with position count (encoder values)

LS_SetJoystickOff	
Description:	Analogue joystick Off
Delphi:	function LS_SetJoystickOff: Integer; function LSX_SetJoystickOff(LSID: Integer): Integer;
C++:	int SetJoystickOff();
LabView:	
Parameter:	-
Example:	LS.SetJoystickOff();

LS_SetJoystickOn	
Description:	Analogue joystick On
Delphi:	function LS_SetJoystickOn(PositionCount, Encoder: LongBool): Integer; function LSX_SetJoystickOn(LSID: Integer; PositionCount, Encoder: LongBool): Integer;
C++:	int SetJoystickOn (BOOL PositionCount,BOOL Encoder);
LabView:	
Parameter:	PositionCount: Position count On/Off Encoder: Encoder values (positions), if any
Example:	LS.SetJoystickOn(true, true); // Joystick On with position count (encoder values)

Control panel with Trackball and Joyspeed-keys

LS_GetBPZ	
Description:	Reads the condition of the additional control panel with track ball
Delphi:	function LS_GetBPZ(var AValue: Integer): Integer; function LSX_GetBPZ(LSID: Integer; var AValue: Integer): Integer;
C++:	int GetBPZ (int *plAValue);
LabView:	-
Parameter:	AValue: 0 => Control panel is „OFF“. 1 => Control panel active, track ball runs with 0.1µ step resolution. 2 => Control panel active, track ball runs with factor.
Example:	LS.GetBPZ(&AValue);

LS_SetBPZ	
Description	Control panel On/ Off
Delphi	function LS_SetBPZ(AValue: Integer): Integer; function LSX_SetBPZ(LSID: Integer; AValue: Integer): Integer;
C++	int SetBPZ (int lAValue);
LabView:	-
Parameter	AValue: 0-2 0 => Control panel „OFF“ 1 => activate operating control panel and trackball with 0,1µ step resolution 2 => activate opearting control panel and trackball with factor.
Example	LS.SetBPZ(1);

LS_GetBPZJoyspeed	
Description:	Control panel joystick-speed
Delphi:	function LS_GetBPZJoyspeed(APar: Integer; var AValue: Double): Integer; function LSX_GetBPZJoyspeed(LSID: Integer; APar: Integer; var AValue: Double): Integer;
C++:	int GetBPZJoyspeed (int lAPar, double *pdAValue);
LabView:	-
Parameter:	APar: 1,2 or 3 AValue: maximun speed [rp/s]
Example:	GetBPZJoyspeed(1, &AValue); // Read out the set speed of parameter 1.

LS_SetBPZJoyspeed	
Description	Control panel joystick-speed
Delphi	function LS_SetBPZJoyspeed(APar: Integer; AValue: Double): Integer; function LSX_SetBPZJoyspeed(LSID: Integer; APar: Integer; AValue: Double): Integer;
C++	int SetBPZJoyspeed (int lAPar, double dAValue);
LabView:	-
Parameter	APar: 1,2 or 3 AValue: +- Maximum speed (vel)
Example	SetBPZJoyspeed(1, 25) // Set parameter 1 to speed 25

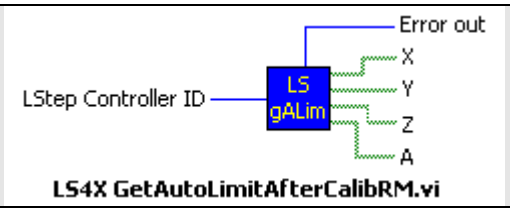
LS_GetBPZTrackballBacklash	
Description	Read out control panel track ball-back lash
Delphi	function LS_GetBPZTrackballBackLash(var X, Y, Z, R: Double): Integer; function LSX_GetBPZTrackballBackLash(LSID: Integer; var X, Y, Z, R: Double): Integer;
C++	int GetBPZTrackballBackLash (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	-
Parameter	X, Y, Z, R: Back lash, mm.
Example	LS.GetBPZTrackballBackLash(&X, &Y, &Z, &R);

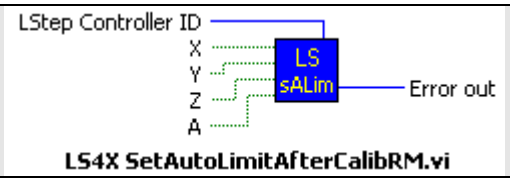
LS_SetBPZTrackballBacklash	
Description	Control panel trackball-reverse backlash
Delphi	function LS_SetBPZTrackballBackLash(X, Y, Z, R: Double): Integer; function LSX_SetBPZTrackballBackLash(LSID: Integer; X, Y, Z, R: Double): Integer;
C++	int SetBPZTrackballBackLash (double dX, double dY, double dZ, double dR);
LabView:	-
Parameter	X, Y, Z, R: 0.001 to 0.15 mm
Example	LS.SetBPZTrackballBackLash(0.01, 0.01, 0.01, 0.01);

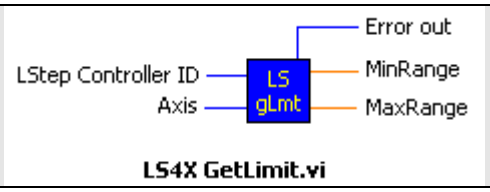
LS_GetBPZTrackballFactor	
Description	Read out control panel trackball-factor
Delphi	function LS_GetBPZTrackballFactor(AValue: Double): Integer; function LSX_GetBPZTrackballFactor(LSID: Integer; AValue: Double): Integer;
C++	int GetBPZTrackballFactor (double *pdAValue);
LabView:	-
Parameter	AValue: Trackball – Factor. e. g. AValue = 3 means: One trackball-impulse results 3 motor-Increment.
Example	LS.GetBPZTrackballFactor(&AValue) ;

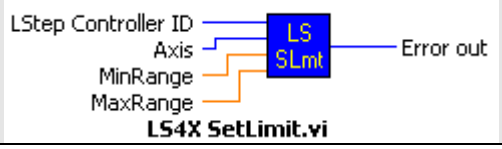
LS_SetBPZTrackballFactor	
Description	Control panel trackball-factor
Delphi	function LS_SetBPZTrackballFactor(AValue: Double): Integer; function LSX_SetBPZTrackballFactor(LSID: Integer; AValue: Double): Integer;
C++	int SetBPZTrackballFactor (double dAValue);
LabView:	-
Parameter	AValue: 0.01 – 100 AValue=1 => Trackball – Factor = 1, i.e. One trackball-impulse results one motor-Increment.
Example	LS.SetBPZTrackballFactor(1.0) ;

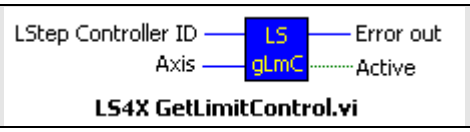
Limit switch (Hardware a. Software)

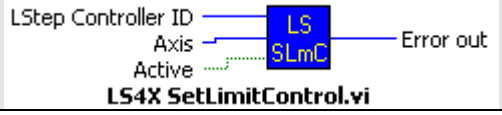
LS_GetAutoLimitAfterCalibRM	
Description:	Indicates if the internal software limits will be set during calibration and table stroke measuring.
Delphi:	function LS_GetAutoLimitAfterCalibRM(var lFlags: Integer): Integer; function LSX_ GetAutoLimitAfterCalibRM(LSID: Integer; var lFlags: Integer): Integer;
C++:	int GetAutoLimitAfterCalibRM (int *plFlags);
LabView:	
Parameter:	lFlags: Bit mask Bit 0 = 1 → No travel limits are set for the x-axis Bit 1 = 0 → Software limits are set for the y-axis (calib/ rm)
Example:	LS. SetAutoLimitAfterCalibRM(&lFlags);

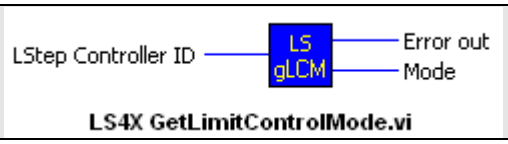
LS_SetAutoLimitAfterCalibRM	
Description:	Prevents that the internal software limits are set during calibration and table stroke measuring.
Delphi:	function LS_SetAutoLimitAfterCalibRM(lFlags: Integer): Integer; function LSX_ SetAutoLimitAfterCalibRM(LSID: Integer; lFlags: Integer): Integer;
C++:	int SetAutoLimitAfterCalibRM (int lFlags);
LabView:	
Parameter:	lFlags: Bit mask Bit 0 = 1 → No travel limits are set for the x-axis Bit 1 = 0 → Software limits are set for the y-axis (calib/ rm)
Example:	LS. SetAutoLimitAfterCalibRM(lFlags);

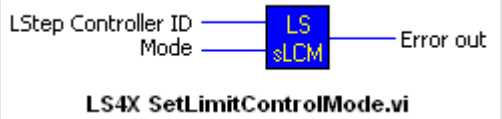
LS_GetLimit	
Description:	Set travel limits
Delphi:	function LS_GetLimit(Axis: Integer; var MinRange, MaxRange: Double): Integer; function LSX_GetLimit(LSID: Integer; Axis: Integer; var MinRange, MaxRange: Double): Integer;
C++:	int GetLimit (int lAxis, double *pdMinRange, double *pdMaxRange);
LabView:	
Parameter:	Axis: the axis for which the travel limits are to be read (X, Y, Z, A numbered from 1 to 4) MinRange: minimum travel limit, depending on dimension MaxRange: maximum travel limit, depending on dimension
Example:	LS.GetLimit(1, &MinRange, &MaxRange);

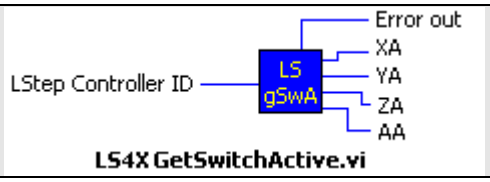
LS_SetLimit	
Description:	Set travel limits
Delphi:	function LS_SetLimit(Axis: Integer; MinRange, MaxRange: Double): Integer; function LSX_SetLimit(LSID: Integer; Axis: Integer; MinRange, MaxRange: Double): Integer;
C++:	int SetLimit (int lAxis,double dMinRange,double dMaxRange);
LabView:	
Parameter:	Axis: the axis for which the travel limits are to be set (X, Y, Z, A numbered from 1 to 4) MinRange: minimum travel limit MaxRange: maximum travel limit
Example:	LS.SetLimit(1, -10.0, 20.0); (Set 10 as minimum limit and 20 as maximum limit for the X-axis)

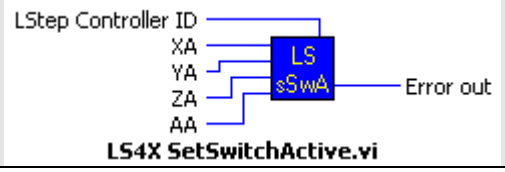
LS_GetLimitControl	
Description:	Reads, if travel range monitoring is active
Delphi:	function LS_GetLimitControl(Axis: Integer; var Active: LongBool): Integer; function LSX_GetLimitControl(LSID: Integer; Axis: Integer; var Active: LongBool): Integer;
C++:	int GetLimitControl (int lAxis, BOOL *pbActive);
LabView:	
Parameter:	Active: True = travel range monitoring is active
Example:	LS.GetLimitControl(2, &Active); // Activ = False means: Travel range monitoring of axis y is deactivated.

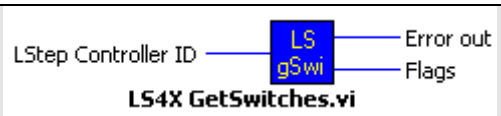
LS_SetLimitControl	
Description:	Range monitoring
Delphi:	function LS_SetLimitControl(Axis: Integer; Active: LongBool): Integer; function LSX_SetLimitControl(LSID: Integer; Axis: Integer; Active: LongBool): Integer;
C++:	int SetLimitControl (int lAxis,BOOL Active);
LabView:	
Parameter:	Axis: (X, Y, Z, A numbered from 1 to 4) Active: Activate range monitoring for the axis in question
Example:	LS.SetLimitControl(2, true); // Range monitoring for Y-axis active

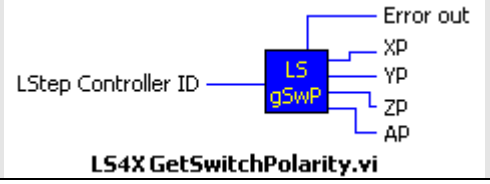
LS_GetLimitControlMode	
Description:	Supply the mode for monitoring of the Software limits.
Delphi:	function LS_GetLimitControlMode (var Mode: Integer): Integer; function LSX_GetLimitControlMode (LSID: Integer; var Mode: Integer): Integer;
C++:	int GetLimitControlMode (int *plMode);
LabView:	 LS4X GetLimitControlMode.vi
Parameter:	**lMode** – Mode. lMode = 0→ Moves, which are outside of the positioning capacity, will only be executed up to the limits of the positioning capacity lMode = 1 → Moves, which are outside of the positioning capacity will not be executed.
Example:	LS. GetLimitControlMode (&lMode);

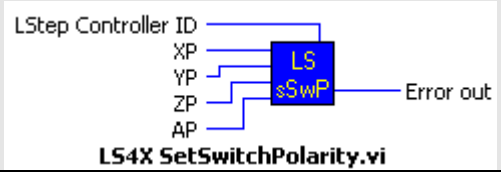
LS_SetLimitControlMode	
Description:	Place the mode for monitoring of the Software limits.
Delphi:	function LS_SetLimitControlMode (Mode: Integer): Integer; function LSX_SetLimitControlMode (LSID: Integer; Mode: Integer): Integer;
C++:	int SetLimitControlMode (int lMode);
LabView:	 LS4X SetLimitControlMode.vi
Parameter:	**lMode** – Mode. lMode = 0 → Moves, which are outside of the positioning capacity, will only be executed up to the limit of the positioning capacity lMode = 1 → Moves, which are outside of the positioning capacity will not be executed
Example:	LS. SetLimitControlMode(1);

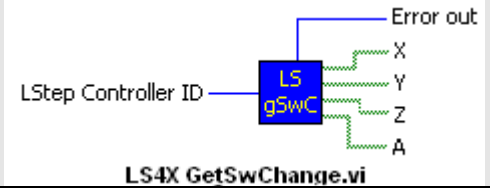
LS_GetSwitchActive	
Description:	Read status of limit switch On / Off
Delphi:	function LS_GetSwitchActive(var XA, YA, ZA, AA: Integer): Integer; function LSX_GetSwitchActive(LSID: Integer; var XA, YA, ZA, AA: Integer): Integer;
C++:	int GetSwitchActive (int *plXA, int *plYA, int *plZA, int *plRA);
LabView:	
Parameter:	A bit mask is transmitted for each axis:
	Bit 0 ➔ Zero limit switch Bit 1 ➔ Reference limit switch Bit 2 ➔ End limit switch
	To activate the respective switch, the appropriate bit must be set.
Example:	LS.GetSwitchActive(&XA, &YA, &ZA, &RA);

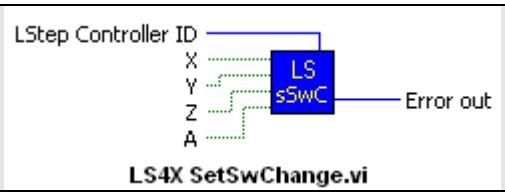
LS_SetSwitchActive	
Description:	Limit Switch On/Off
Delphi:	function LS_SetSwitchActive(XA, YA, ZA, AA: Integer): Integer; function LSX_SetSwitchActive(LSID: Integer; XA, YA, ZA, AA: Integer): Integer;
C++:	int SetSwitchActive (int lXA,int lYA,int lZA,int lAA);
LabView:	
Parameter:	A bit mask is transmitted for each axis:
	Bit 0 ➔ Zero limit switch Bit 1 ➔ Reference limit switch Bit 2 ➔ End limit switch
	To activate the respective switch, the appropriate bit must be set.
Example:	LS.SetSwitchActive(7, 1, 5, 0); (All X-axis limit switches ON; Y-axis zero limit switch ON; Z-axis E0 and EE ON; A-axis: all limit switches OFF)

LS_GetSwitches	
Description:	Reads the status of all limit switches
Delphi:	function LS_GetSwitches(var Flags: Integer): Integer;
C++:	int GetSwitches (int *plFlags);
LabView:	
Parameter:	Value: Pointer to integer value which contains the status of all limit switches as a bit mask. The condition of the limit switch is coded in the bit mask as follows: Limit switch EE Ref. E0 Axis AZYX AZYX AZYX Bit 0000 0000 0000 e.g. Flags = 0x003 → E0 of X- and Y-Axis are reached Flags = 0x200 → EE of Y-Axis is reached
Example:	LS.GetSwitches(&Flags);

LS_GetSwitchPolarity	
Description:	Reads limit switch polarity
Delphi:	function LS_GetSwitchPolarity(var XP, YP, ZP, AP: Integer): Integer; function LSX_GetSwitchPolarity(LSID: Integer; var XP, YP, ZP, AP: Integer): Integer;
C++:	int GetSwitchPolarity (int *plXP, int *plYP, int *plZP, int *plRP);
LabView:	
Parameter:	A bit mask is transmitted for each axis:
	Bit 0 → Zero limit switch Bit 1 → Reference limit switch Bit 2 → End limit switch
	To activate the respective switch, the appropriate bit must be set.
Example:	LS.GetSwitchPolarity(&XP, &YP, &ZP, &RP);

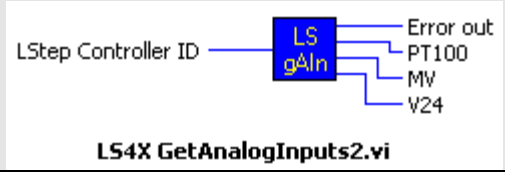
LS_SetSwitchPolarity	
Description:	Set limit switch polarity
Delphi:	function LS_SetSwitchPolarity(XP, YP, ZP, AP: Integer): Integer; function LSX_SetSwitchPolarity(LSID: Integer; XP, YP, ZP, AP: Integer): Integer;
C++:	int SetSwitchPolarity (int lXP,int lYP,int lZP,int lRA);
LabView:	
Parameter:	A bit mask is transmitted for each axis:
	Bit 0 ➔ Zero limit switch Bit 1 ➔ Reference limit switch Bit 2 ➔ End limit switch
	If the respective switch reacts to the positive flank, the bit must be set.
Example:	LS.SetSwitchPolarity(7, 0, 0, 0); (All X-axis limit switches high-active, all Y-axis limit switches low-active...)

LS_GetSwChange	
Description:	Displays adjustments of limit switch
Delphi:	function LS_GetSwChange(var Flags: Integer): Integer; function LSX_GetSwChange (LSID: Integer; var Flags: Integer): Integer;
C++:	int GetSwChange (int *plFlags);
LabView:	
Parameter:	Flags: 32-bit-Integer, which contains after calling of the function in the Bits 0-4 the Bit-mask. Bit 0 = 1 → Change of limit switch at X-axis Bit 2 = 0 → No change of limit switch at Z-axis
Example:	LS. GetSwChange (&Flags);

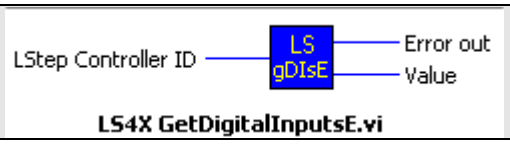
LS_SetSwChange	
Description:	Change limit switch
Delphi:	function LS_ SetSwChange (Flags: Integer): Integer; function LSX_ SetSwChange (LSID: Integer; Flags: Integer): Integer;
C++:	int SetSwChange (int Flags);
LabView:	
Parameter:	Flags: Bit mask Bit 0 = 1 → X-axis change limit switches Bit 2 = 0 →Z-axis no change of limit switches
Example:	LS. SetSwChange (3); /* X- and Y-axis – change limit switch (Bits 0 a. 1 set), Z- and A-axis – no change of limit switch (Bit 2 = 0) */

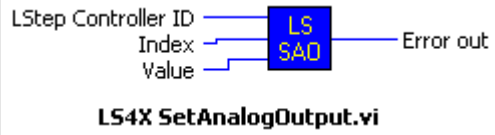
Digital and analogue Input and Output

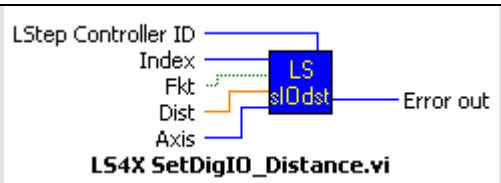
LS_GetAnalogInput	
Description:	Reads the current status of an analogue channel
Delphi:	function LS_GetAnalogInput(Index: Integer; var Value: Integer): Integer; function LSX_GetAnalogInput(LSID: Integer; Index: Integer; var Value: Integer): Integer;
C++:	int GetAnalogInput (int lIndex,int *plValue);
LabView:	
Parameter:	Index: 0-9 (Analogue channels) Value: Pointer to integer value, that indicates the current condition of the analogous channel.
Example:	LS.GetAnalogInput(0, &Inputs0);

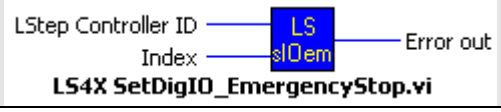
LS_GetAnalogInputs2	
Description	read the current condition of the analogous channels (Channel 6, 7, 8) only with the LSTEP-PCI
Delphi	function LS_GetAnalogInputs2(var PT100, MV, V24: Integer): Integer; function LSX_GetAnalogInputs2(LSID: Integer; var PT100, MV, V24: Integer): Integer;
C++	int GetAnalogInputs2 (int *plPT100, int *plMV, int *plV24);
LabView	
Parameter	PT100, MV, V24: Pointer to integer value, in which GetAnalogInputs2 supposed to write the current condition of the analogue channels.
Example	LS.GetAnalogInputs2(&PT100, &MV, &V24);

LS_GetDigitalInputs	
Description:	Read all input pins
Delphi:	function LS_GetDigitalInputs(var Value: Integer): Integer; function LSX-GetDigitalInputs(LSID: Integer; var Value: Integer): Integer;
C++:	int GetDigitalInputs (int *plValue);
LabView:	
Parameter:	Value: Pointer to integer value which contains the status of all inputs as a bit mask.
Example:	int inputs; LS.GetDigitalInputs(&Inputs); if (Inputs & 16) ... // when input pin 4 is set

LS_GetDigitalInputsE	
Description	read additional digital inputs (16-31)
Delphi	function LS_GetDigitalInputsE(var Value: Integer): Integer; function LSX_GetDigitalInputsE(LSID: Integer; var Value: Integer): Integer;
C++	int GetDigitalInputsE (int *plValue);
LabView:	
Parameter	Value: Pointer to a 32-bit-Integer, which after activating the function contains the status of the inputs 16-31 in the Bits 0-15.
Example	LS.GetDigitalInputsE(i);

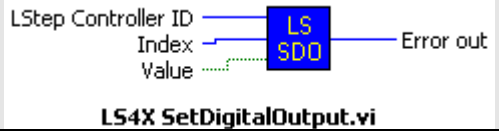
LS_SetAnalogOutput	
Description:	Set analogue output
Delphi:	function LS_SetAnalogOutput(Index: Integer; Value: Integer): Integer; function LSX_SetAnalogOutput(LSID: Integer; Index: Integer; Value: Integer): Integer;
C++:	int SetAnalogOutput (int lIndex,int lValue);
LabView:	
Parameter:	Index: 0-1 (Analogue channels) Value: 0-100 [%]
Example:	LS.SetAnalogOutput(0, 100); // Set output 0 to maximum

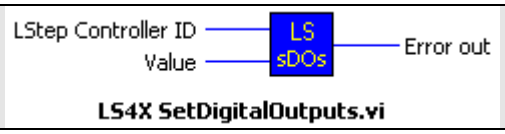
LS_SetDigIO_Distance	
Description:	Function of the digital inputs / outputs
	Activation of an output dependent on the set distance before /after the target position.
Delphi:	function LS_SetDigIO_Distance(Index: Integer; Fkt: LongBool; Dist: Double; Axis: Integer): Integer; function LSX_SetDigIO_Distance(LSID: Integer; Index: Integer; Fkt: LongBool; Dist: Double; Axis: Integer): Integer;
C++:	int SetDigIO_Distance (int lIndex,BOOL Fkt,double dDist,int lAxis);
LabView:	
Parameter:	Index: 0 to 15 (Output pin)
	Fkt = false ➔ Activation of an output dependent on the set distance before the target position. Fkt = true ➔ Activation of an output dependent on the set distance after the start position.
	Dist: Stretch (Input depends on the set dimension) Axis: (X, Y, Z, A numbered from 1 to 4)
Example:	LS.SetDigIO_Distance(7, false, 78.9, 3); /* Output 7 is activated 78.9mm before the target position (Z-axis) is reached. */

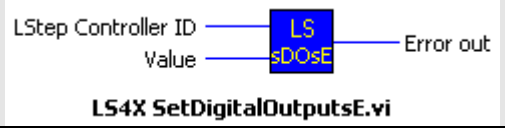
LS_SetDigIO_EmergencyStop	
Description:	Function of the digital inputs / outputs Allocating of the Emergency Stop pin
Delphi:	function LS_SetDigIO_EmergencyStop(Index: Integer): Integer; function LSX_SetDigIO_EmergencyStop(LSID: Integer; Index: Integer): Integer;
C++:	int SetDigIO_EmergencyStop (int lIndex);
LabView:	
Parameter:	Index: 0 to 15 (Input/Output)
Example:	LS.SetDigIOEmergencyStop(15); // Emergency Stop pin 15

LS_SetDigIO_Off	
Description:	"Off" function of the digital inputs/outputs (no influence of the inputs/outputs)
Delphi:	function LS_SetDigIO_Off(Index: Integer): Integer; function LSX_SetDigIO_Off(LSID: Integer; Index: Integer): Integer;
C++:	int SetDigIO_Off (int lIndex);
LabView:	
Parameter:	Index: 0 to 15 (Input/Output), 16 (all 16 port pins)
Example:	LS.SetDigIO_Off(0); // dig. fkt. Input/Output pin 0 off

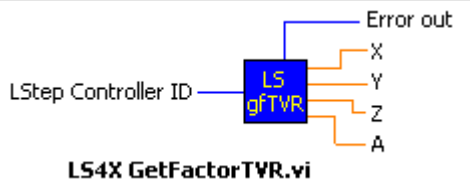
LS_SetDigIO_Polarity	
Description:	Function of the digital inputs / outputs Set polarity
Delphi:	function LS_SetDigIO_Polarity(Index: Integer; High: LongBool): Integer; function LSX_SetDigIO_Polarity(LSID: Integer; Index: Integer; High: LongBool): Integer;
C++:	int SetDigIO_Polarity (int lIndex,BOOL High);
LabView:	
Parameter:	Index: 0 to 15 (Input/Output), 16 (all 16 port pins)
	High = true ➔ High-active High = false ➔ Low-active
Example:	LS.SetDigIO_Polarity(3, True); // Input/Output pin 3 high-active

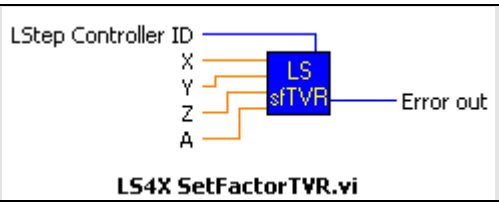
LS_SetDigitalOutput	
Description:	Set output pin
Delphi:	function LS_SetDigitalOutput(Index: Integer; Value: LongBool): Integer; function LSX_SetDigitalOutput(LSID: Integer; Index: Integer; Value: LongBool): Integer;
C++:	int SetDigitalOutput (int lIndex,BOOL Value);
LabView:	
Parameter:	Index: 0-15 Value: Set status to “0” or “1”
Example:	LS.SetDigitalOutput(0, true); // Set output pin 0 to “1”

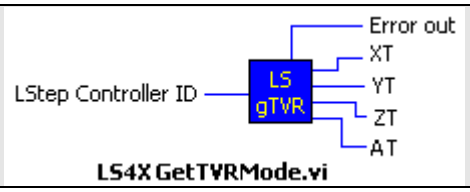
LS_SetDigitalOutputs	
Description	Set digital outputs (0-15)
Delphi	function LS_SetDigitalOutputs(Value: Integer): Integer; function LSX_SetDigitalOutputs(LSID: Integer; Value: Integer): Integer;
C++	int SetDigitalOutputs (int lValue);
LabView:	
Parameter	Value: Bit mask, the value that the outputs 0-15 are set to, is determined via the Bits 0-15.
Example	LS.SetDigitalOutputs($03); // Set outputs 0 and 1 to 1, the remaining 0

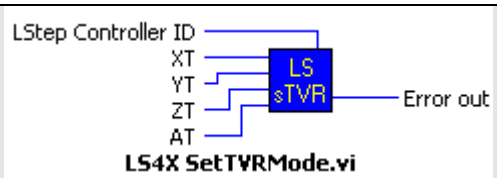
LS_SetDigitalOutputsE	
Description	Set additional outputs (16-31)
Delphi	function LS_SetDigitalOutputsE(Value: Integer): Integer; function LSX_SetDigitalOutputsE(LSID: Integer; Value: Integer): Integer;
C++	int SetDigitalOutputsE (int lValue);
LabView:	
Parameter	Value: Bit mask, the value that the outputs 16-31 are set to, is determined via the Bits 0-15.
Example	LS.SetDigitalOutputsE($03); // Set outputs 16 and 17 to 1, the remaining 0

Clock pulse Forward / Back

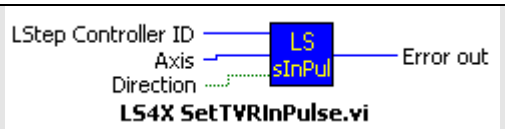
LS_GetFactorTVR	
Description:	Reads factor for clock pulse Forward/ Back
Delphi:	function LS_GetFactorTVR(var X, Y, Z, A: Double): Integer; function LSX_GetFactorTVR(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetFactorTVR (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z and A: Factor clock pulse Forward/ Back. e.g. X = 10 means: One pulse = ten Motor increments
Example:	LS.GetFactorTVR(&X, &Y, &Z, &A);

LS_SetFactorTVR	
Description:	Factor for clock pulse Forward/ Back
Delphi:	function LS_SetFactorTVR(X, Y, Z, A: Double): Integer; function LSX_SetFactorTVR(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetFactorTVR (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, Y, Z and A 0.01 – 100.00
Example:	LS.SetFactorTVR(2.0, 2.0, 0, 0); /* Clock pulse Forward/Back is to work with the factor 2 for the X- and Y-axis */

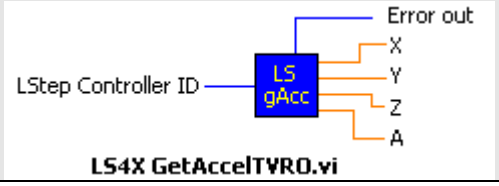
LS_GetTVRMode	
Description:	Read setup of clock pulse Forward /Back (= TVR Mode)
Delphi:	function LS_ GetTVRMode(var XT, YT, ZT, AT: Integer): Integer; function LSX_ GetTVRMode(LSID: Integer; var XT, YT, ZT, AT: Integer): Integer;
C++:	int GetTVRMode (int *plXT, int *plYT, int *plZT, int *plRT);
LabView:	
Parameter:	TVR-mode for X, Y, Z and A:
	0 ➔ Clock pulse Forward /Back (= TVR mode) “OFF” 1 ➔ Normal clock pulse Forward/Back processing 2 ➔ Processing of clock pulse Forward/Back with a factor 3 ➔ Clock pulse Forward /Back processing must be externally enabled with the triggerout pin (MFP). 4 ➔ Combination of 2 & 3.
Example:	LS. GetTVRMode(&XT, &YT, &ZT, &RT);

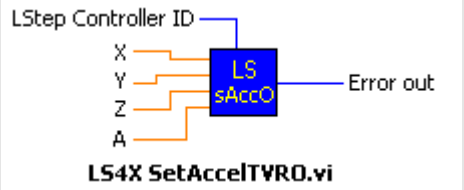
LS_SetTVRMode	
Description:	Set clock pulse Forward / Back
Delphi:	function LS_SetTVRMode(XT, YT, ZT, AT: Integer): Integer; function LSX_SetTVRMode(LSID: Integer; XT, YT, ZT, AT: Integer): Integer;
C++:	int SetTVRMode (int lXT,int lYT,int lZT,int lAT);
LabView:	
Parameter:	TVR-mode for X, Y, Z and A:
	0 ➔ Clock pulse Forward /Back (= TVR mode) "OFF" 1 ➔ Normal clock pulse Forward/Back processing 2 ➔ Processing of clock pulse Forward/Back with a factor 3 ➔ Clock pulse Forward /Back processing must be externally enabled with the triggerout pin (MFP). 4 ➔ Combination of 2 & 3.
Example:	LS.SetTVRMode(1, 1, 0, 0); // TVR ON for X- and Y-axes

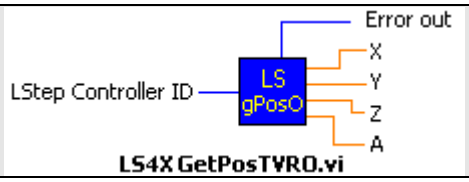
Clock pulse Forward /Back via Interface

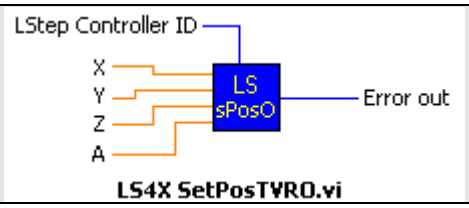
LS_SetTVRInPulse	
Description:	This function has the same influence as an external clock pulse with direction information.
Delphi:	function LS_SetTVRInPulse (Axis: Integer; Direction: Boolean): Integer; function LSX_SetTVRInPulse (LSID: Integer; Axis: Integer; Direction: Boolean): Integer;
C++:	int SetTVRInPulse (int Axis, BOOL Direction);
LabView:	
Parameter:	Value: Amount of executed Trigger
Example:	LS.SetTVRInPulse (2, true); // 1 clock pulse Forward y-Axis.

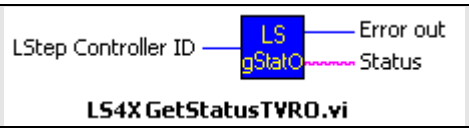
Clock pulse Forward / Back for the additional axes

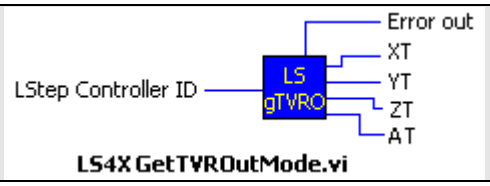
LS_GetAccelTVRO	
Description:	Reads the set acceleration for the additional axes.
Delphi:	function LS_GetAccelTVRO(var X, Y, Z, A: Double): Integer; function LSX_GetAccelTVRO(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetAccelTVRO (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Acceleration values, R/s²
Example:	LS.GetAccelTVRO(&X, &Y, &Z, &A);

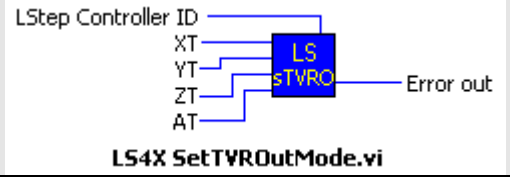
LS_SetAccelTVRO	
Description:	Set acceleration for the additional axes
Delphi:	function LS_SetAccelTVRO(X, Y, Z, A: Double): Integer; function LSX_SetAccelTVRO(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetAccelTVRO (double dX, double dY, double dZ, double dA);
LabView:	
Parameter:	X, Y, Z and A: Acceleration, value range 0.01 – 1500 [R/s²]
Example:	LS.SetAccelTVRO(1.0, 1.5, 0, 0);

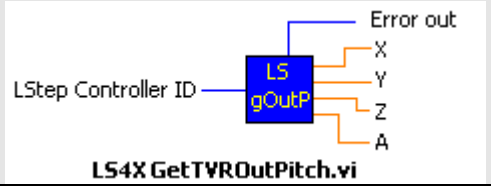
LS_GetPosTVRO	
Description:	Read position of the additional axis
Delphi:	function LS_GetPosTVRO(var dX, dY, dZ, dR: Double): Integer; function LSX_GetPosTVRO(LSID: Integer; var dX, dY, dZ, dR: Double): Integer;
C++:	int GetPosTVRO (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	dX, dY, dZ, dR: Position value, depending on the dimension
Example:	LS. GetPosTVRO(&X, &Y, &Z, &R);

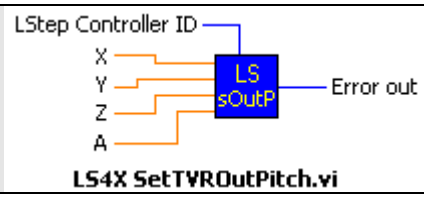
LS_SetPosTVRO	
Description:	Set position of the additional axis
Delphi:	function LS_ SetPosTVRO(dX, dY, dZ, dR: Double): Integer; function LSX_ SetPosTVRO(LSID: Integer; dX, dY, dZ, dR: Double): Integer;
C++:	int SetPosTVRO (double dX, double dY, double dZ, double dR);
LabView:	
Parameter:	dX, dY, dZ, dR: Position value, depending on the dimension. Value range: min. range limit to max. range limit
Example:	LS. SetPosTVRO(10.0, 5.0, 0.0, 0.0);

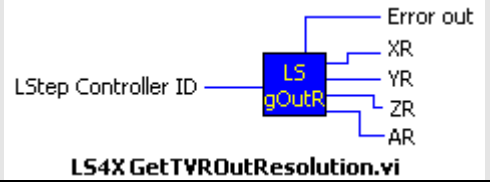
LS_GetStatusTVRO	
Description:	Delivers the current status of the additional axis
Delphi:	function LS_GetStatusTVRO(pcStat: PChar; MaxLen: Integer): Integer; function LSX_ GetStatusTVRO(LSID: Integer; pcStat: PChar; MaxLen: Integer): Integer;
C++:	int GetStatusTVRO (char *pcStat, int lMaxLen);
LabView:	
Parameter:	pcStat: Pointer to a buffer in which the status string is returned MaxLen: Maximum number of characters, that can be copied into the buffer e.g.: @ - M – @ = Axis standing M = Axis moving (Motion) - = Axis is not enabled
Example:	LS. GetStatusTVRO(pcStat, 256); // Move the additional Z-axis 5mm in positive direction

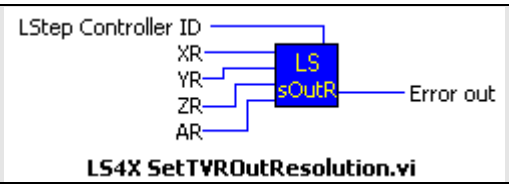
LS_GetTVROutMode	
Delphi:	function LS_GetTVROutMode(var X, Y, Z, A: Integer): Integer; function LSX_GetTVROutMode(LSID: Integer; var X, Y, Z, A: Integer): Integer;
C++:	int GetTVROutMode (int *plXT, int *plYT, int *plZT, int *plAT);
LabView:	
Parameter:	X, Y, Z and A: 0 => Pulse Forw/Back is “OFF” 1 => Pulse Forw/Back is “ON”
Example:	LS.GetTVROutMode(&X, &Y, &Z, &A);

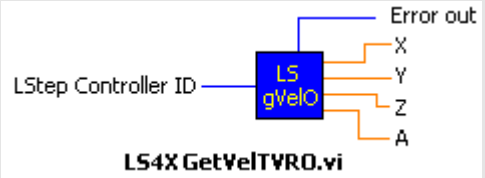
LS_SetTVROutMode	
Description:	Set additional axis X, Y, Z and A, beside the actual main axis X, Y, Z and A
Delphi:	function LS_SetTVROutMode(X, Y, Z, A: Integer): Integer; function LSX_SetTVROutMode(LSID: Integer; X, Y, Z, A: Integer): Integer;
C++:	int SetTVROutMode (int lXT, int lYT, int lZT, int lAT);
LabView:	
Parameter:	X, Y, Z and A: 0 or 1
Example:	LS.SetTVROutMode(1, 0, 1, 0); //Pulse Forw/Back of the x and z is activated, and deactivated for y and a.

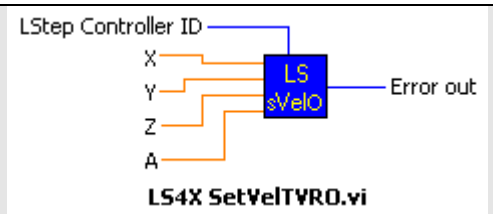
LS_GetTVROutPitch	
Description:	Reads the spindle pitch of the additional axis
Delphi:	function LS_GetTVROutPitch(var X, Y, Z, R: Double): Integer; function LSX_ GetTVROutPitch (LSID: Integer; var X, Y, Z, R: Double): Integer;
C++:	int GetTVROutPitch (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z and R: Spindle pitch [mm]
Example:	LS. GetTVROutPitch(&X, &Y, &Z, &A);

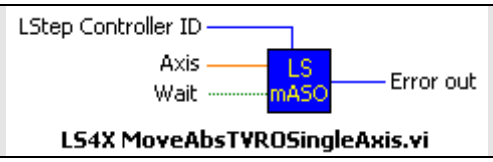
LS_SetTVROutPitch	
Description:	Sets the spindle pitch for the additional axis
Delphi:	function LS_SetTVROutPitch(X, Y, Z, R: Double): Integer; function LSX_ SetTVROutPitch (LSID: Integer; X, Y, Z, R: Double): Integer;
C++:	int SetTVROutPitch (double dX, double dY, double dZ, double dR);
LabView:	
Parameter:	X, Y, Z and R: Spindle pitch [mm], value range 0.001 to 100
Example:	LS. SetTVROutPitch(1.0, 4.0, 1.0, 1.0); /* Spindle pitch of y-axis is 4 mm. For x-, z- and a-axis spindle with 1mm pitch are used */

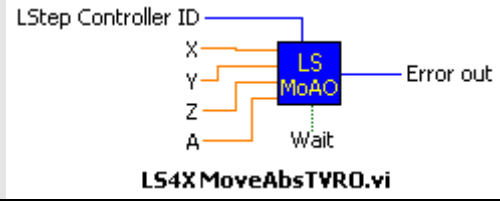
LS_GetTVROutResolution	
Description:	Reads the resolution of the power amplifier which is to be controlled
Delphi:	function LS_GetTVROutResolution(var X, Y, Z, A: Integer): Integer; function LSX_ GetTVROutResolution (LSID: Integer; var X, Y, Z, A: Integer): Integer;
C++:	int GetTVROutResolution (int *plX, int *plY, int *plZ, int *plA);
LabView:	
Parameter:	X, Y, Z and A: Impulses per rotation
Example:	LS. GetTVROutResolution (&X, &Y, &Z, &A);

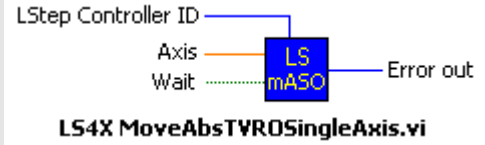
LS_SetTVROutResolution	
Description:	Sets the resolution of the power amplifier which is to be controlled
Delphi:	function LS_SetTVROutResolution(X, Y, Z, A: Integer): Integer; function LSX_ SetTVROutResolution (LSID: Integer; X, Y, Z, A: Integer): Integer;
C++:	int SetTVROutResolution (int lX, int lY, int lZ, int lA);
LabView:	
Parameter:	X, Y, Z and A: Impulses per rotation value range 0 to 51200
Example:	LS. SetTVROutResolution (1000, 1000, 0, 0); /* The resolution of axis X and Y is 1000 impulses per rotation */

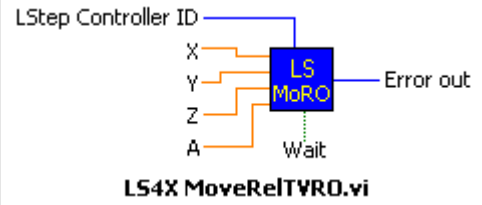
LS_GetVelTVRO	
Description:	Reads the set speed of the additional axis
Delphi:	function LS_GetVelTVRO(var X, Y, Z, A: Double): Integer; function LSX_GetVelTVRO(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetVelTVRO (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Speed values, [rp/s]
Example:	LS.GetVelTVRO(&X, &Y, &Z, &A);

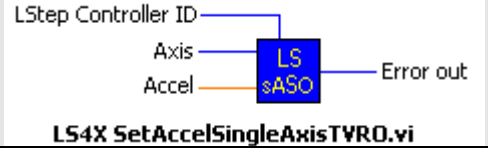
LS_SetVelTVRO	
Description:	set speed of the additional axis
Delphi:	function LS_SetVelTVRO(X, Y, Z, A: Double): Integer; function LSX_SetVelTVRO(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetVelTVRO (double dX, double dY, double dZ, double dA);
LabView:	
Parameter:	X, Y, Z and A: speed, 0 – 40.0 [rp/s]
Example:	LS.SetVelTVRO(1.0, 1.5, 0, 0);

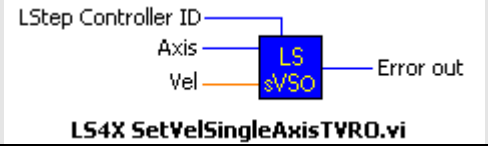
LS_MoveAbsTVROSingleAxis	
Description:	Position single axis absolute
Delphi:	function LS_MoveAbsTVROSingleAxis (Axis: Integer; Value: Double; Wait: LongBool): Integer; function LSX_MoveAbsTVROSingleAxis (LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;
C++:	int MoveAbsTVROSingleAxis (int lAxis, double dValue, BOOL bWait);
LabView:	
Parameter:	Axis: (X, Y, Z, A numbered from 1 to 4) Value: Position (Input depends on the set dimension)
Example:	LS.MoveAbsTVROSingleAxis (2, 10.0); //Position additional Y-axis to 10mm absolute

LS_MoveAbsTVRO	
Description:	Move to absolute position (The additional axes x, y, z and a are positioned on the given position values)
Delphi:	function LS_MoveAbsTVRO(X, Y, Z, A: Double; Wait: LongBool): Integer; function LSX_MoveAbsTVRO (LSID: Integer; X, Y, Z, A: Double; Wait: LongBool): Integer;
C++:	int MoveAbsTVRO (double dX, double dY, double dZ, double dR, BOOL bWait);
LabView:	
Parameter:	X, Y, Z and A +- Moving range Input depends on the set dimension Wait: Specifies whether the function should return after the position has been reached (= true) or directly (= false)
Example:	LS.MoveAbsTVRO (10.0, 10.0, 10.0, 10.0, true);

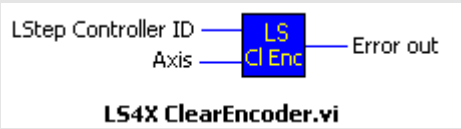
LS_MoveRelTVROSingleAxis	
Description:	Move single axis absolute
Delphi:	function LS_MoveRelTVROSingleAxis (Axis: Integer; Value: Double; Wait: LongBool): Integer; function LSX_MoveRelTVROSingleAxis (LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;
C++:	int MoveRelTVROSingleAxis (int lAxis,double dValue,BOOL Wait);
LabView:	
Parameter:	Axis: (X, Y, Z, A numbered from 1 to 4) Value: Stretch (Input depends on the set dimension)
Example:	LS.MoveRelTVROSingleAxis (3, 5.0); // Move the additional Z-axis 5mm in positive direction

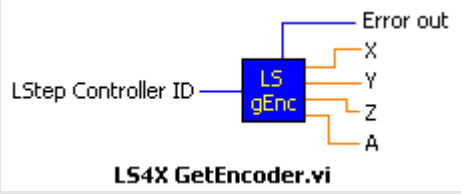
LS_MoveRelTVRO	
Description:	Move relative vector (The additional axes x, y, z and a are moved the length of the set vector)
Delphi:	function LS_MoveRelTVRO(X, Y, Z, A: Double; Wait: LongBool): Integer; function LSX_MoveRelTVRO(LSID: Integer; X, Y, Z, A: Double; Wait: LongBool): Integer;
C++:	int MoveRelTVRO (double dX, double dY, double dZ, double dR, BOOL bWait);
LabView:	
Parameter:	X, Y, Z and A +- Moving range Input depends on the set dimension Wait: Specifies whether the function should return after the position has been reached (= true) or directly (= false)
Example:	LS.MoveRelTVRO(10.0, 10.0, 10.0, 10.0, true);

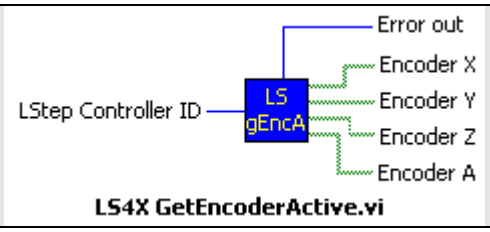
LS_SetAccelSingleAxisTVRO	
Description:	Acceleration of single additional axis
Delphi:	function LS_SetAccelSingleAxisTVRO(Axis: Integer; Accel: Double): Integer; function LSX_SetAccelSingleAxisTVRO (LSID: Integer; Axis: Integer; Accel: Double): Integer;
C++:	int SetAccelSingleAxisTVRO (int lAxis, double dAccel);
LabView:	
Parameter:	Axis: (X, Y, Z, A numbered from 1 to 4) Accel: 0.01 – 1500 [R/s²]
Example:	LS.SetAccelSingleAxis(2, 50.0); // The Z-axis will be accelerated with 50 rp/s²

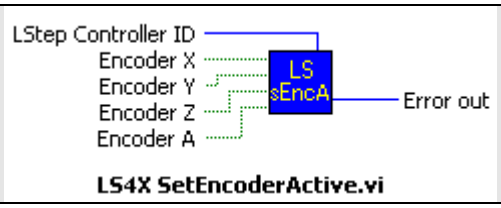
LS_SetVelSingleAxisTVRO	
Description:	Set acceleration of single additional axis
Delphi:	function LS_SetVelSingleAxisTVRO(Axis: Integer; Vel: Double): Integer; function LSX_SetVelSingleAxisTVRO (LSID: Integer; Axis: Integer; Vel: Double): Integer;
C++:	int SetVelSingleAxisTVRO (int lAxis, double dVel);
LabView:	
Parameter:	Axis: (X, Y, Z, A numbered from 1 to 4) Vel: 0 – 40.0 [R/s]
Example:	LS.SetVelSingleAxis(1, 10.0); // The X-axis should run with max. speed of 10 rp/s

Encoder-Settings

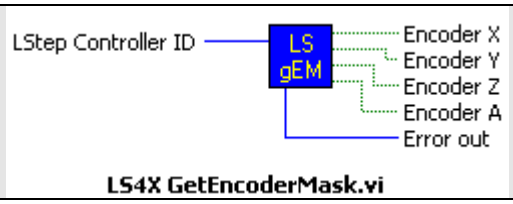
LS_ClearEncoder	
Description:	Set encoder-counter to zero
Delphi:	function LS_ClearEncoder(lAxis: Integer): Integer; function LSX_ClearEncoder (LSID: Integer; lAxis: Integer): Integer;
C++:	int ClearEncoder (int lAxis);
LabView:	
Parameter:	lAxis: (X, Y, Z, A numbered from 1 to 4)
Example:	LS. ClearEncoder (2); // Set encoder-counter of y-axis to zero

LS_GetEncoder	
Description:	Reads all encoder positions
Delphi:	function LS_GetEncoder(XP, YP, ZP, AP: Double): Integer; function LSX_GetEncoder (LSID: Integer; XP, YP, ZP, AP: Double): Integer;
C++:	int GetEncoder (double *pdXP, double *pdYP, double *pdZP, double *pdRP);
LabView:	
Parameter:	XP, YP, ZP, AP: Meter value, 4-fold interpolated
Example:	LS. GetEncoder (&XP, &YP, &ZP, &AP);

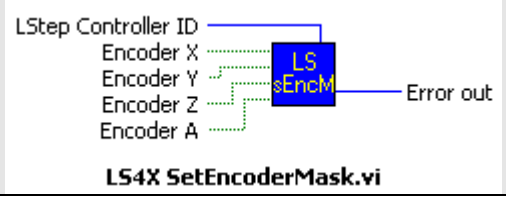
LS_GetEncoderActive	
Description:	Reads, which encoders are activated after the calibration.
Delphi:	function LS_GetEncoderActive(var Flags: Integer): Integer; function LSX_GetEncoderActive(LSID: Integer; var Flags: Integer): Integer;
C++:	int GetEncoderActive (int *plFlags);
LabView:	
Parameter:	Flags: Encoder mask
Example:	LS.GetEncoderActive(&Flags);

LS_SetEncoderActive	
Description:	This function is used to select which encoder is to be activated after calibration.
Delphi:	function LS_SetEncoderActive(Flags: Integer): Integer; function LSX_SetEncoderActive(LSID: Integer; Flags: Integer): Integer;
C++:	int SetEncoderActive (int lFlags);
LabView:	
Parameter:	Value: Encoder mask

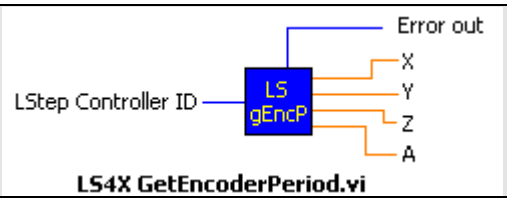
Example:	LS.SetEncoderActive(0);
	// Deactivate all encoders
	LS.SetEncoderMask(2);
	// Activate Y-axis encoder

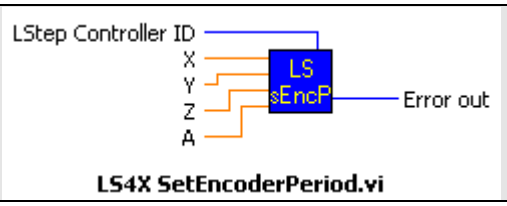
LS_GetEncoderMask	
Description:	Read encoder statuses
Delphi:	function LS_GetEncoderMask (var Flags: Integer): Integer; function LSX_GetEncoderMask (LSID: Integer; var Flags: Integer): Integer;
C++:	int GetEncoderMask (int *plFlags);
LabView:	
Parameter:	Flags: Encoder mask

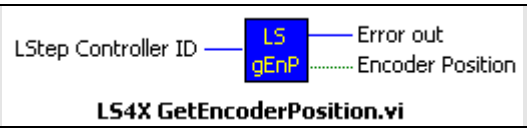
Example:	int EncMask;
	LS.GetEncoderMask(&EncMask);
	if (EncMask & 2) ...
	// If Y-axis encoder is connected +active

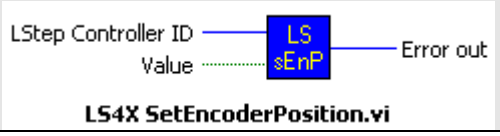
LS_SetEncoderMask	
Description:	(de-)activate encoder
Delphi:	function LS_SetEncoderMask(Value: Integer): Integer; function LSX_SetEncoderMask(LSID: Integer; Value: Integer): Integer;
C++:	int SetEncoderMask (int lValue);
LabView:	
Parameter:	Value: Encoder mask

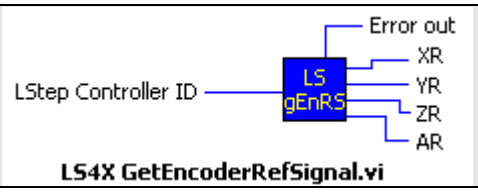
Example:	LS.SetEncoderMask(0);
	// Deactivate all encoders
	LS.SetEncoderMask(2);
	// Activate Y-axis encoder

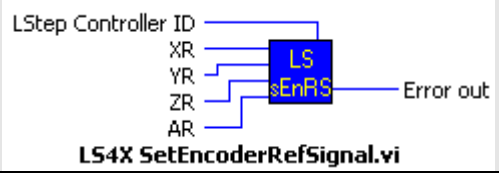
LS_GetEncoderPeriod	
Description:	Read length of encoder period
Delphi:	function LS_GetEncoderPeriod(var X, Y, Z, A: Double): Integer; function LSX_GetEncoderPeriod(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetEncoderPeriod (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z and A: Period length [mm]
Example:	LS.GetEncoderPeriod(&X, &Y, &Z, &A);

LS_SetEncoderPeriod	
Description:	Set length of encoder period
Delphi:	function LS_SetEncoderPeriod(X, Y, Z, A: Double): Integer; function LSX_SetEncoderPeriod(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetEncoderPeriod (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, Y, Z and A 0.0001 – Spindle pitch * 0.8 (mm)
Example:	LS.SetEncoderPeriod(0.1, 0.1, 0.1, 0.1); // Encoder period length is 0.1 mm for all axes

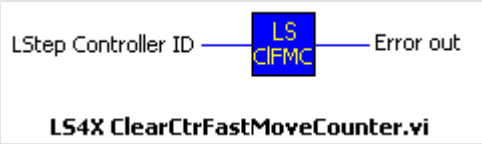
LS_GetEncoderPosition	
Description	Read encoder position setting
Delphi	function LS_GetEncoderPosition(Value: Boolean): Integer; function LSX_GetEncoderPosition(LSID: Integer; Value: LongBool): Integer;
C++	int GetEncoderPosition (BOOL *pbValue);
LabView	
Parameter	Value: true → The encoder values of the detected encoders are displayed when the position inquiry is placed false → encoder postion display is "OFF"
Example	LS.GetEncoderPosition(&Value);

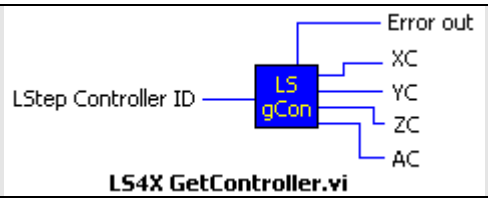
LS_SetEncoderPosition	
Description	Encoder position display On/Off
Delphi	function LS_SetEncoderPosition(Value: Boolean): Integer; function LSX_SetEncoderPosition(LSID: Integer; Value: LongBool): Integer;
C++	int SetEncoderPosition (BOOL fValue);
LabView	
Parameter	Value = true → The encoder values of the detected encoders are displayed when the position inquiry is placed
Example	LS.SetEncoderPosition(true);

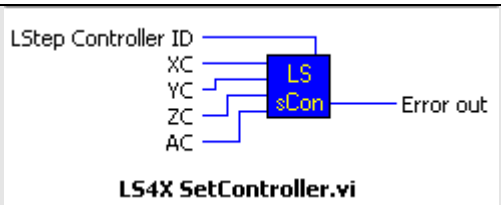
LS_GetEncoderRefSignal	
Description:	Reads if interpret reference signal from encoder when calibration is done
Delphi:	function LS_GetEncoderRefSignal(var XR, YR, ZR, AR: Integer): Integer; function LSX_GetEncoderRefSignal(LSID: Integer; var XR, YR, ZR, AR: Integer): Integer;
C++:	int GetEncoderRefSignal (int *plXR, int *plYR, int *plZR, int *plRR);
LabView:	
Parameter:	X, Y, Z and A: 1 => When calibration the reference signal is interpreted. 0 => The reference signal is not interpreted
Example:	LS.GetEncoderRefSignal(&X, &Y, &Z, &A);

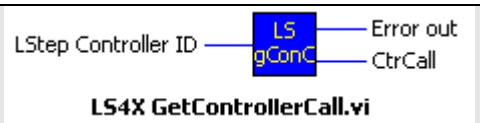
LS_SetEncoderRefSignal	
Description:	Interpret reference signal from encoder when calibration is done
Delphi:	function LS_SetEncoderRefSignal(XR, YR, ZR, AR: Integer): Integer; function LSX_SetEncoderRefSignal(LSID: Integer; XR, YR, ZR, AR: Integer): Integer;
C++:	int SetEncoderRefSignal (int lXR,int lYR,int lZR,int lAR);
LabView:	
Parameter:	X, Y, Z and A 0 or 1
Example:	LS.SetEncoderRefSignal(1, 1, 0, 0); /* When calibration is done, the reference signal of the encoders x and y are interpreted. */

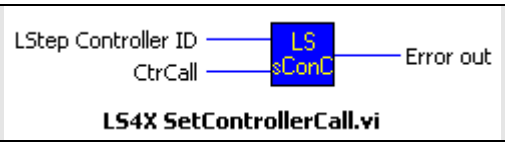
Controller Setting

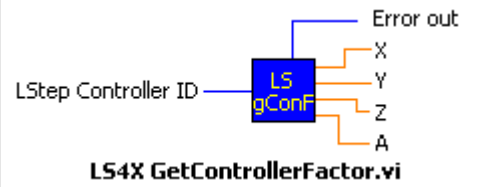
LS_ClearCtrFastMoveCounter	
Description:	If the controller difference is larger than the catch range, a new vector is started and the corresponding counter is extended by one. This function sets Fast Move Counters of all axes to zero.
Delphi:	function LS_ClearCtrFastMoveCounter: Integer; function LSX_ClearCtrFastMoveCounter(LSID: Integer): Integer;
C++:	int ClearCtrFastMoveCounter;
LabView:	
Parameter:	
Example:	LS. ClearCtrFastMoveCounter;

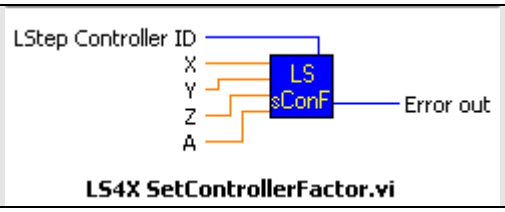
LS_GetController	
Description:	Read controller mode
Delphi:	function LS_GetController(var XC, YC, ZC, AC: Integer): Integer; function LSX_GetController(LSID: Integer; var XC, YC, ZC, AC: Integer): Integer;
C++:	int GetController (int *plXC, int *plYC, int *plZC, int *plRC);
LabView:	
Parameter:	Controller mode X, Y, Z and A :
	0 ➔ Controller "OFF" 1 ➔ Controller "OFF after reaching target position" 2 ➔ Controller "Always ON"
	3 ➔ Controller "OFF after reaching target position" with reduced current
	4 ➔ Controller "Always ON" with reduced current
Example:	LS.GetController(&X, &Y, &Z, &A);

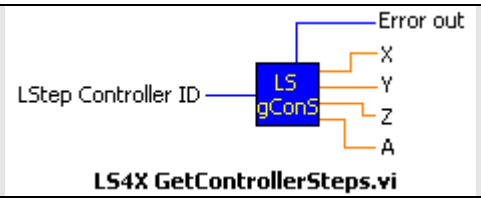
LS_SetController	
Description:	Set controller mode
Delphi:	function LS_SetController(XC, YC, ZC, AC: Integer): Integer; function LSX_SetController(LSID: Integer; XC, YC, ZC, AC: Integer): Integer;
C++:	int SetController (int lXC,int lYC,int lZC,int lAC);
LabView:	
Parameter: **time**	Controller mode X, Y, Z and A : 0 ➔ Controller "OFF" 1 ➔ Controller "OFF after reaching target position" 2 ➔ Controller "Always ON" 3 ➔ Controller "OFF after reaching target position" with reduced current 4 ➔ Controller "Always ON" with reduced current
Example:	LS.SetController(1, 2, 0, 0);

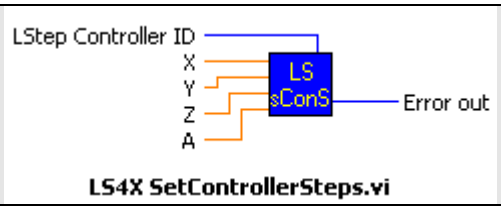
LS_GetControllerCall	
Description:	Reads controller call time
Delphi:	function LS_GetControllerCall(var CtrCall: Integer): Integer; function LSX_GetControllerCall(LSID: Integer; var CtrCall: Integer): Integer;
C++:	int GetControllerCall (int *plCtrCall);
LabView:	
Parameter:	CtrCall: Controller call time [ms]
Example:	LS.GetControllerCall(&CtrCall); //After function call CtrCall = 10 means: controller call every 10 ms

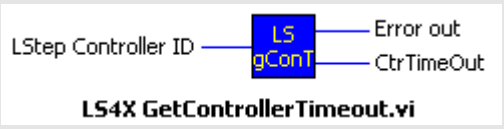
LS_SetControllerCall	
Description:	Call controller
Delphi:	function LS_SetControllerCall(CtrCall: Integer): Integer; function LSX_SetControllerCall(LSID: Integer; CtrCall: Integer): Integer;
C++:	int SetControllerCall (int lCtrCall);
LabView:	
Parameter:	CtrCall: Controller call time [ms]
Example:	LS.SetControllerCall(10);

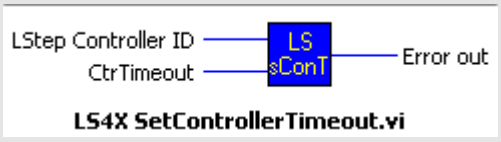
LS_GetControllerFactor	
Description:	Reads controller factor see chap. 4.13 „Controller setting for LSTEP“
Delphi:	function LS_GetControllerFactor(var X, Y, Z, A: Double): Integer; function LSX_GetControllerFactor(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetControllerFactor (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z and A: Controller factor
Example:	LS.GetControllerFactor(&X, &Y, &Z, &A);

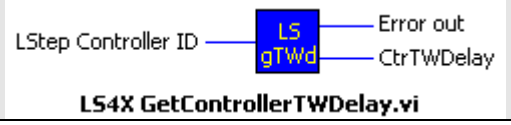
LS_SetControllerFactor	
Description:	Controller factor
Delphi:	function LS_SetControllerFactor(X, Y, Z, A: Double): Integer; function LSX_SetControllerFactor(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetControllerFactor (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, Y, Z and A 1 – 64

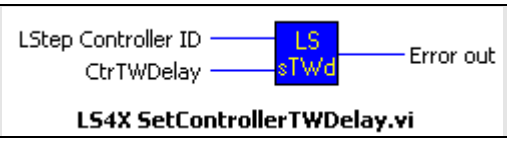
LS_GetControllerSteps	
Description:	Reads controller steps length
Delphi:	function LS_GetControllerSteps(var X, Y, Z, A: Double): Integer; function LSX_GetControllerSteps(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetControllerSteps (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z and A: Controller steps length [mm]
Example:	LS.GetControllerSteps(&X, &Y, &Z, &A);

LS_SetControllerSteps	
Description:	Controller Steps
Delphi:	function LS_SetControllerSteps(X, Y, Z, A: Double): Integer; function LSX_SetControllerSteps(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetControllerSteps (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, Y, Z and A 1 – Spindle pitch (Values depend on the dimension)
Example:	LS.SetControllerSteps(4, 5, 7, 9);

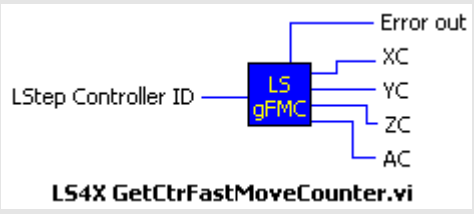
LS_GetControllerTimeout	
Description:	Reads controller timeout
Delphi:	function LS_GetControllerTimeout(var ACtrTimeout: Integer): Integer; function LSX_GetControllerTimeout(LSID: Integer; var ACtrTimeout: Integer): Integer;
C++:	int GetControllerTimeout (int *plACtrTimeout);
LabView:	
Parameter:	ACtrTimeout: Timeout [ms], time after which a travel command returns with an error message (error code 4013), if the controller could not definitively find a position.
Example:	LS.GetControllerTimeout(&ACtrTimeout);

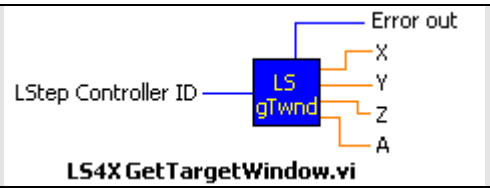
LS_SetControllerTimeout	
Description:	Controller timeout
Delphi:	function LS_SetControllerTimeout(ACtrTimeout: Integer): Integer; function LSX_SetControllerTimeout(LSID: Integer; ACtrTimeout: Integer): Integer;
C++:	int SetControllerTimeout (int ACtrTimeout);
LabView:	
Parameter:	ACtrTimeout: Timeout [ms], time after which a travel command returns with an error message (error code 4013) , if the controller could not definitively find a position.
Example:	LS.SetControllerTimeout(500);

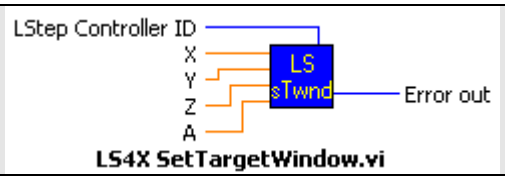
LS_GetControllerTWDelay	
Description	Read controller relay
Delphi	function LS_GetControllerTWDelay(var CtrTWDelay: Integer): Integer; function LSX_GetControllerTWDelay(LSID: Integer; var CtrTWDelay: Integer): Integer;
C++	int GetControllerTWDelay (int *plCtrTWDelay);
LabView	
Parameter	CtrTWDelay: Controller delay [ms]
Example	LS.GetControllerTWDelay(&CtrTWDelay);

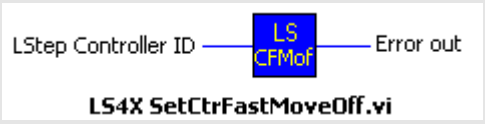
LS_SetControllerTWDelay	
Description	Controller delay
Delphi	function LS_SetControllerTWDelay(CtrTWDelay: Integer): Integer; function LSX_SetControllerTWDelay(LSID: Integer; CtrTWDelay: Integer): Integer;
C++	int SetControllerTWDelay (int lCtrTWDelay);
LabView	
Parameter	CtrTWDelay: Controller delay 0 – 100 [ms]
Example	LS.SetControllerTWDelay(0); // Controller delay off

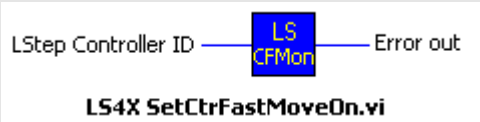
LS_GetCtrFastMove	
Description:	Reads setting of the Fast Move Function
Delphi:	function LS_ GetCtrFastMoveOff(var bActive: LongBool): Integer; function LSX_ GetCtrFastMoveOff(LSID: Integer; var bActive: LongBool): Integer;
C++:	int GetCtrFastMove (BOOL *pbActive);
LabView:	
Parameter:	bActive: True => Fast Move Function active
Example:	LS. GetCtrFastMoveOff (&bActive);

LS_GetCtrFastMoveCounter	
Description:	If the controller difference is larger than the catch range, a new vector is started and the corresponding counter is extended by one. The Function delivers Fast Move Counters
Delphi:	function LS_ GetCtrFastMoveCounter(var XC, YC, ZC, RC: Integer): Integer; function LSX_ GetCtrFastMoveCounter(LSID: Integer; var XC, YC, ZC, RC: Integer): Integer;
C++:	int GetCtrFastMoveCounter (int *plXC, int *plYC, int *plZC, int *plRC);
LabView:	
Parameter:	XC, YC, ZC, RC: Amount of finished Fast Move functions
Example:	LS. SetCtrFastMoveCounter (&XC, &YC, &ZC, &RC);

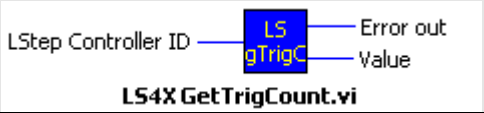
LS_GetTargetWindow	
Description:	Reads the target window
Delphi:	function LS_GetTargetWindow(var X, Y, Z, A: Double): Integer; function LSX_GetTargetWindow(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetTargetWindow (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z and A: Target window , depend on the dimension)
Example:	LS.GetTargetWindow(&X, &Y, &Z, &A);

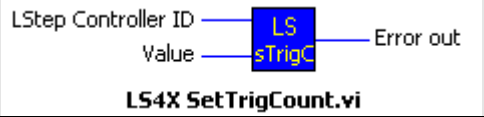
LS_SetTargetWindow	
Description:	Target Window
Delphi:	function LS_SetTargetWindow(X, Y, Z, A: Double): Integer; function LSX_SetTargetWindow(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetTargetWindow (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, Y, Z and A 1 – 25000 (motor increments) 0.1 – Spindle pitch/2 (µm) 0.0001 – Spindle pitch/2 (mm) (Values depend on the dimension)
Example:	LS.SetTargetWindow(1.0, 0.002, 1.0, 1.0);

LS_SetCtrFastMoveOff	
Description:	Deactivate Fast Move Function
Delphi:	function LS_ SetCtrFastMoveOff: Integer; function LSX_ SetCtrFastMoveOff(LSID: Integer): Integer;
C++:	int SetCtrFastMoveOff ();
LabView:	
Parameter:	
Example:	LS. SetCtrFastMoveOff ();

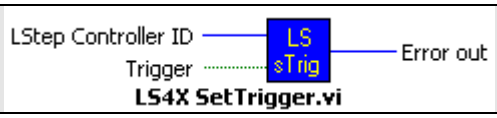
LS_SetCtrFastMoveOn	
Description:	Fast Move function activated i.e. In a regulator difference that is larger than the capture area, a new vector is started.
Delphi:	function LS_ SetCtrFastMoveOn: Integer; function LSX_ SetCtrFastMoveOn(LSID: Integer): Integer;
C++:	int SetCtrFastMoveOn ();
LabView:	
Parameter:	
Example:	LS. SetCtrFastMoveOn();

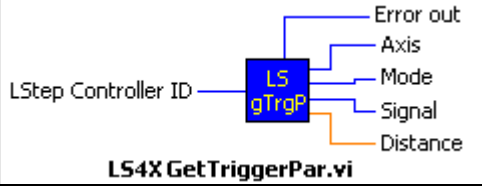
Trigger-Output

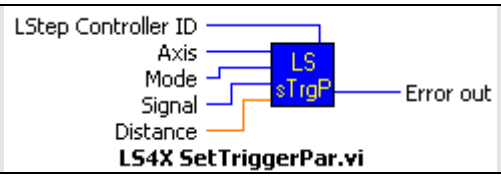
LS_GetTrigCount	
Description:	Read Trigger counter.
Delphi:	function LS_GetTrigCount (var Value: Integer): Integer; function LSX_GetTrigCount (LSID: Integer; var Value: Integer): Integer;
C++:	int GetTrigCount (int *pValue);
LabView:	
Parameter:	Value: Amount of executed Trigger
Example:	LS.GetTrigCount (&Value);

LS_SetTrigCount	
Description:	Set Trigger counter.
Delphi:	function LS_ SetTrigCount (Value: Integer): Integer; function LSX_ SetTrigCount (LSID: Integer; Value: Integer): Integer;
C++:	int SetTrigCount (int Wert);
LabView:	
Parameter:	Value: 0 through 2147483647
Example:	LS.SetTrigCount (0);

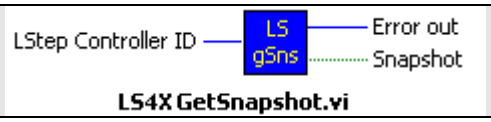
LS_GetTrigger	
Description:	Reads the current Trigger-Condition
Delphi:	function LS_GetTrigger(var ATrigger: LongBool): Integer; function LSX_GetTrigger(LSID: Integer; var ATrigger: LongBool): Integer;
C++:	int GetTrigger (BOOL *pbATrigger);
LabView:	
Parameter:	ATrigger: True => Trigger „ON“ False => Trigger „OFF“
Example:	LS.GetTrigger(&ATrigger);

LS_SetTrigger	
Description:	Trigger On/Off
Delphi:	function LS_SetTrigger(ATrigger: LongBool): Integer; function LSX_SetTrigger(LSID: Integer; ATrigger: LongBool): Integer;
C++:	int SetTrigger (BOOL bATrigger);
LabView:	
Parameter:	ATrigger: Trigger On/Off
Example:	LS.SetTrigger(true);

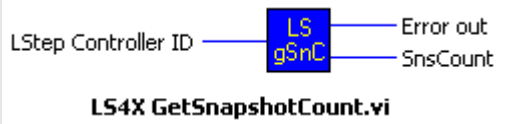
LS_GetTriggerPar	
Description:	Reads Trigger-Parameter
Delphi:	function LS_GetTriggerPar(var Axis, Mode, Signal: Integer; var Distance: Double): Integer; function LSX_GetTriggerPar(LSID: Integer; var Axis, Mode, Signal: Integer; var Distance: Double): Integer;
C++:	int GetTriggerPar (int *plAxis, int *plMode, int *plSignal, double *pdDistance);
LabView:	
Parameter:	Axis: Axis (1..4) Mode: Trigger mode (see command !trigm) Signal: Trigger signal (see command !trigs) Distance: Trigger distance (see command !trigd)
Example:	LS.GetTriggerPar(&Axis, & Mode, & Signal, & Distance);

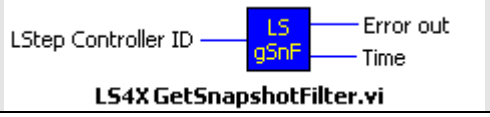
LS_SetTriggerPar	
Description:	Trigger parameters
Delphi:	function LS_SetTriggerPar(Axis, Mode, Signal: Integer; Distance: Double): Integer; function LSX_SetTriggerPar(LSID: Integer; Axis, Mode, Signal: Integer; Distance: Double): Integer;
C++:	int SetTriggerPar (int lAxis, int lMode, int lSignal, double dDistance);
LabView:	
Parameter:	Axis: Axis (1..4) Mode: Trigger mode (see command !trigm) Signal: Trigger signal (see command !trigs) Distance: Trigger distance (see command !trigd)
Example:	LS.SetTriggerPar(1, 3, 2, 5.0);

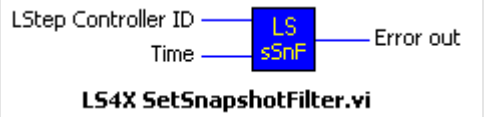
Snapshot-Input

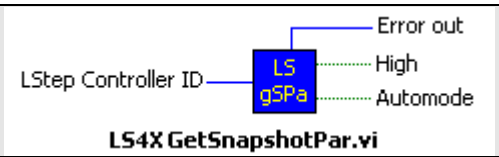
LS_GetSnapshot	
Description:	Reads the current Snapshot-condition
Delphi:	function LS_GetSnapshot(var ASnapshot: LongBool): Integer; function LSX_GetSnapshot(LSID: Integer; var ASnapshot: LongBool): Integer;
C++:	int GetSnapshot (BOOL *pbASnapshot);
LabView:	
Parameter:	ASnapshot: True => Snapshot „ON“ False => Snapshot „OFF“
Example:	LS.GetSnapshot(&ASnapshot);

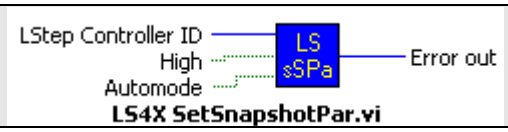
LS_SetSnapshot	
Description:	Snapshot On/Off
Delphi:	function LS_SetSnapshot(ASnapshot: LongBool): Integer; function LSX_SetSnapshot(LSID: Integer; ASnapshot: LongBool): Integer;
C++:	int SetSnapshot (BOOL bASnapshot);
LabView:	
Parameter:	ASnapshot: Snapshot On/Off
Example:	LS.SetSnapshot(true);

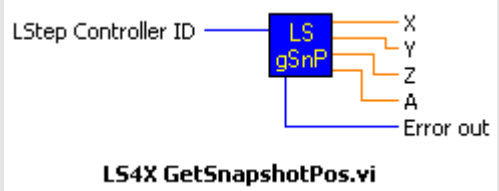
LS_GetSnapshotCount	
Description:	Snapshot Counter
Delphi:	function LS_GetSnapshotCount(var SnsCount: Integer): Integer; function LSX_GetSnapshotCount(LSID: Integer; var SnsCount: Integer): Integer;
C++:	int GetSnapshotCount (int *plSnsCount);
LabView:	
Parameter:	SnsCount: Snapshot Counter
Example:	LS.GetsnapshotCount(&SnsCount);

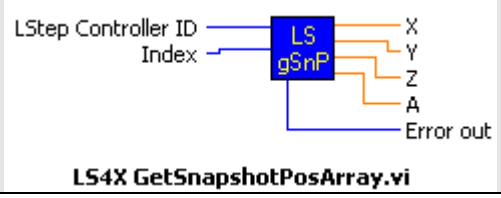
LS_GetSnapshotFilter	
Description	Reads input filter (snapshot-filter)
Delphi:	function LS_GetSnapshotFilter(var lTime: Integer): Integer; function LSX_GetSnapshotFilter(LSID: Integer; var lTime: Integer): Integer;
C++:	int GetSnapshotFilter (int *plTime);
LabView:	
Parameter:	lTime: Filter time [ms]
Example:	LS. GetSnapshotFilter(&lTime);

LS_SetSnapshotFilter	
Description	Set input filter for rebounding switches.
Delphi:	function LS_SetSnapshotFilter(lTime: Integer): Integer; function LSX_ SetSnapshotFilter(LSID: Integer; lTime: Integer): Integer;
C++:	int SetSnapshotFilter (int lTime);
LabView:	
Parameter:	lTime: Filter time, value range 0 – 100 ms
Example:	LS. SetSnapshotFilter(0); // no Snapshot filter

LS_GetSnapshotPar	
Description	Read Snapshot-Parameter
Delphi:	function LS_GetSnapshotPar(var High, AutoMode: LongBool): Integer; function LSX_GetSnapshotPar(LSID: Integer; var High, AutoMode: LongBool): Integer;
C++:	int GetSnapshotPar (BOOL *pbHigh, BOOL *pbAutoMode);
LabView:	
Parameter:	High: True => Snapshot is high-active. False => Low-active AutoMode: True => Snapshot „Automatic“. The position is automatically approached after the first impulse.
Example:	LS.GetSnapshotPar(&High, & AutoMode);

LS_SetSnapshotPar	
Description	Snapshot parameters
Delphi:	function LS_SetSnapshotPar(High, AutoMode: LongBool): Integer; function LSX_SetSnapshotPar(LSID: Integer; High, AutoMode: LongBool): Integer;
C++:	int SetSnapshotPar (BOOL bHigh, BOOL bAutoMode);
LabView:	
Parameter:	High: Snapshot high-active AutoMode: approach snapshot position automatically
Example:	LS.SetSnapshotPar(true, false);

LS_GetSnapshotPos	
Description	Read snapshot position
Delphi	function LS_GetSnapshotPos(var X, Y, Z, A: Double): Integer; function LSX_GetSnapshotPos(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++	int GetSnapshotPos (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter	X, Y, Z, A: Position values
Example	double X, Y, Z, A; LS.GetSnapshotPos(&X, &Y, &Z, &A);

LS_GetSnapshotPosArray	
Description	Read snapshot-position from array
Delphi	function LS_GetSnapshotPosArray(Index: Integer; var X, Y, Z, R: Double): Integer; function LSX_GetSnapshotPosArray(LSID: Integer; Index: Integer; var X, Y, Z, R: Double): Integer;
C++	int GetSnapshotPosArray (int lIndex, double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter	Index: Number of snapshot-position (1-200) X, Y, Z, A: Position values
Example	double X, Y, Z, A; LS.GetSnapshotPos(2, &X, &Y, &Z, &A);

6.5 Error Codes

LStep-number.	API-number	Comment
0	0	no error
	4001.4002	Internal error
	4003	undefined error
	4004	Interface type unknown (may occur with Connect..)
	4005	Interface initialization error
	4006	No connection to the controller (e.g. when SetPitch is called before Connect)
	4007	Timeout whilst reading from the interface
	4008	Command transmission error to LSTEP
	4009	Command terminated (with SetAbortFlag)
	4010	Command not supported by API
11	4011	Joystick set to Manual (may occur with SetJoystickOn/Off)
11	4012	Travel command not possible, as joystick is in Manual
	4013	Controller timeout
12	4015	actuates limit switch in moving direction
14	4017	Fault during calibration (Limit switch was not set free correctly)
1	4101	Valid axis designation missing
2	4102	Non-executable function
3	4103	Command string has too many characters
4	4104	Invalid command
5	4105	Not within valid numerical range
6	4106	Incorrect number of parameters
7	4107	None !or ?
8	4108	TVR not possible because axis is active
9	4109	Axes cannot be switched on or off because TVR is active
10	4110	Function not configured
11	4111	Move command not possible, as joystick is in Manual
12	4112	Limit switch tripped
13	4113	Function cannot be carried out because Encoder was recognized
14	4114	Fault during calibration (Limit switch was not set free correctly)
15	4115	This function is interrupted activated while releasing the encoder during calibrating or table stroke measuring if the opposite encoder is activated.
20	4120	driver relay defective (safety circle K3/K4)
21	4121	only single vectors may be driven (setup mode)
22	4122	no calibrating, measuring table stroke or joystick operation can be carried out (door open or setup mode)
23	4123	SECURITY Error X-axis
24	4124	SECURITY Error Y-axis
25	4125	SECURITY Error Z-axis
26	4126	SECURITY Error A-axis
27	4127	Stop active
28	4128	Fault in the door switch safety circle (only with LS44/Solero
29	4129	Power amplifiers are not switched on (only with LS44)
30	4130	GAL security error (only with LS44)
31	4131	Joy-stick can not be activated because Move is active

6.6 Frequent questions & answers

How are the LSTEP4.DLL and/or the LSTEP4X.DLL embedded in a MS Visual C++ project?
How do I initialise with the LStep API the connection to the LStep? Which of the Connect-commands should be used?
How do I initialise the driver for the LStep-PCI?
Why does my program with the LSTEP4.DLL get no connection to the LStep-PCI?
A fault occurred during the process, in the LSTEP4.DLL or in my program. What is the cause for that problem, and how can I solve it.
Can I make an inquiry during moving commands about the status of inputs, the current position and something similar to that?
Why are messages processed during the run of LSTEP-API-functions and how can I deactivate this?
When should Moves with or without Wait be used?
How can I move single axes independently from one another with the LSTEP-API?
How can I use several LStep-PCI-cards in a PC?
When should LSTEP4.DLL, when the LSTEP4X.DLL be used?
Is the LSTEP-API compatible to the MCL resp. to the old register-command-set?
Why do I get in MS Visual C++ while connecting the LStep4.cpp the message “fatal error C1010“?
How can I use a special /new LStep-command, which has no suitable LSTEP-API-function?
Why do I see in my debugger of my development environment when using the LSTEP-API the message „First chance exception“, „Exception: Timeout read RS232!“ or something similar?
How can I simulate a sort of joystick with the LSTEP-API, which means moving an axis until a key is released?
How can I save durable the adjustments of the LStep?
How many entries will fit into the log window of the LStep-API?

How are the LSTEP4.DLL and/or the LSTEP4X.DLL embedded in a MS Visual C++ project?

- create project
- copy LSTEP4.DLL, LSTEP4.h, LSTEP4.cpp in a project folder
- insert LSTEP4.h and LSTEP4.cpp in the project
- Menu: Select Project\Adjustment\C/C++ Option: [do not use pre compiled]
- embed in LSTEP4.h #include „stdafx.h“
- in project name_Dlg.h #include „LSTEP4.h“
- embed the required instance in public
  Example: CLStep* MyLStep = new CLStep();

The embedding of the LSTEP4X.DLL is analogue to this \procedure.

How do I initialise with the LStep API the connection to the LStep?
Which of the Connect-commands should be used?
The connection to the LSTEP-API is initialised with a Connect-command (Connect, ConnectEx, ConnectSimple). Besides some special cases, **ConnectSimple** should always be used.

ConnectSimple	when transferring the interfaces parameter directly
Connect	after pre loading of the interfaces parameter out of a .ini file using LoadConfig
ConnectEx	when loading the interface parameter out of a data structure

How do I initialise the driver for the LStep-PCI?
After the correct installation of the LStep-PCI Windows requests a driver during the start for a device of the type „network controller". In this dialog-window click to button „Search" o.s.s and switch to the folder, in which the files of the LSTEP-API where unpacked. The driver-files and the Inf-files are in the sub folder „LStepPCI", which are required for the driver-installation.

Why does my program with the LSTEP4.DLL get no connection to the LStep-PCI?
They should review first of all in the Windows-device-manager, whether the installed LStep-PCI is registered as a device. Also the **file DRVX40.DLL must be in the folder of the LSTEP4.DLL, of your program or in a Windows-system folder.** You find this file in the sub folder „LStepPCI" of the LStep API *(see also chapter 9.7)*.

A fault occurred during the process, in the LSTEP4.DLL or in my program. What is the cause for that problem, and how can I solve it.
So that a fault diagnosis is possible, you should unconditional switch on the recording of the LSTEP-API with SetWriteLogText. Afterwards you should try to reproduce the fault while recording it. You can send the Log-file (LStep4.log) to us, to e analysed.

Can I make an inquiry during moving commands about the status of inputs, the current position and something similar to that?
Yes, for example by calling GetDigitalInputs via a Windows-timer or a second Thread function like GetPos, during the moving command. But it is not possible to call the function WaitForAxisStop, during a moving command with Wait=true.

Why are messages processed during the run of LSTEP-API-functions and how can I deactivate this?
While from moving commands, the LSTEP-API processes messages, so that the program does not „stand still", otherwise it would not be possible to perform an interrupt in case of a fault or to stop the axes. SetProcessMessagesProc enables the replacement of an internal Message-Dispatching procedure of the LSTEP-API. The LSTEP-API processes messages during waiting for acknowledgements of the LStep in the Main-Thread. If you want to turn off the Message-Dispatching or replace it with your own code, you can use SetProcessMessagesProc to set Callback-procedure.

When should Moves with or without Wait be used?
Move-commands with WaitForAxisStop are to be used, if all axes supposed to move synchronic and linearly interpolated. The controller accepts new Move-commands only after all axes are stopped.

Move-commands without WaitForAxisStop are to be used, if all axes supposed to move asynchronous. In this case the user has to make sure; that only the axis stands still that gets a new Move-command.

How can I move single axes independently from one another with the LSTEP-API?
The Move-command of the LSTEP-API offers two possibilities: If the (last) parameter is set Wait=true, the function only returns, after the axes reach there goal position. If the **parameter** is set Wait=false, the LSTEP-API-function only sends the Move-command and returns immediately, without performing the movement.
That means by using MoveAbsSingleAxis with Wait=false for the X-axis and some time later call MoveAbsSingleAxis with Wait=false for the Y-axis, the axis can be moved separately. In order to find out, if the axis reached there goal position the command **WaitForAxisStop** can be used.

Example:

```
LS.MoveAbsSingleAxis(Xaxis, 10, false);   // Move the X-axis asynchronous
Delay(1000);                              // Wait 1s to start the Y-axis
LS.MoveAbsSingleAxis(Yaxis, 20, false);   // Move the Y-axis asynchronous
LS.WaitForAxisStop(3, 0, flag);           // Wait until X- and Y-axis stopped,
                                          without timeout
```

But it is **not possible to use Move-commands with Wait=true and the once with Wait=false simultaneous**. This leads to permanent or sporadic faults in the communication.

Example:

Not allowed:

```
LS.MoveAbsSingleAxis(Xaxis, 10, false);   // Move the X-axis asynchronous
LS.MoveAbsSingleAxis(Yaxis, 20, true);    // Move the Y-axis asynchronous without
                                          waiting for the end of the asynchronous
                                          Move-command.
```

How can I use several LStep-PCI-cards in a PC?
The procedure for the installation is like the one for single card. After the start, Windows requests the driver for all LStep-PCI-cards.
Yet it is **problematically to identify**, which physical card belongs to a certain index-number. It is not guaranteed, that with LS_ConnectSimple(4, nil, 0, true) a connection to LStep-PCI in the first PCI-Slot of the mainboard, with LS_ConnectSimple(4, nil, 1, true) a connection to LStep-PCI in the second PCI-Slot will be achieved etc. Therefore to the unambiguous identification of the cards, the series number with **GetSerialNr** should be queried.

When should LSTEP4.DLL, when the LSTEP4X.DLL be used?
If several LSteps/LStep-PCI-cards supposed to be controlled from one PC, LSTEP4X.DLL should be used; otherwise LSTEP4.DLL is more suitable.

Is the LSTEP-API compatible to the MCL resp. to the old register-command-set?
The LSTEP-API is principle down compatible to the Register-command-set that other MCL and older LSteps communicate with. However this command-set does not offer many possibilities, that the LStep API can in controllers related to new command-set. Therefore

some LSTEP-API-commands cannot generally be used as well as WaitForAxisStop in controls with an old command-set

Why do I get in MS Visual C++ while connecting the LStep4.cpp the message "fatal error C1010"?

IN this case it is not a fault in the file LStep4.cpp. The message usually appears, if the compiler is looking for a pre compiled header-file and can not find it. If in MS Visual C++ the message „fatal error C1010 precompiled header files" appears, the option „pre compiled Header-file" for LStep4.cpp must be turned of. If you don't want to use the MFC in your project, you should erase the #include "stdafx.h" from LStep4.cpp.

How can I use a special /new LStep-command, which has no suitable LSTEP-API-function?

The LSTEP-API-function SendString **offers the possibility, to use new LStep commands, that where not planned for the LSTEP-API. Note, that all commands close with #13 resp.\r !**

Why do I see in my debugger of my development environment when using the LSTEP-API the message „First chance exception", „Exception: Timeout read RS232!" or something similar?

Internal Exceptions of the LSTEP4.DLL, which are only visible in the debugger have no meaning. They serve the internal process control. In ConnectSimple frequently an Exception appears, because the LSTEP-API tries, to find out the command-set. At the same time it comes to a Timeout if the controller does not support the tested command-set. In Delphi, you can add in the Debugger-options the corresponding Exception to the Exceptions to be ignored by Debugger.

How can I simulate a sort of joystick with the LSTEP-API, which means moving an axis until a key is released?

Such a key-Joystick can be implemented as follows:
Start the axis with a very long vector during a keystroke
MoveRelSingleAxis(Xaxis, 100000, false)
Important is to set the parameter Wait=false
When releasing the key call the command **StopAxes**

How can I save durable the adjustments of the LStep?
The LSTEP-API-command **LstepSave** can be used to keep settings (spindle pitch, gear factor, axes current a.s.o.) made once, even after a reset of the LStep. If your LStep supports this command you can read in your documentation.

How many entries will fit into the log window of the LStep-API?
The log window of the LStep-API can contain more than 20.000 logs. Arise more logs, the oldest one will overwritten.

How many logs can be written into the log file?
You can write into the log file so long until the programme will be finished or the hard disk is full.

6.7 API- and ASCII commands / Index 2

Command identifier last column (X): 1 = LSTEP; 2 = LSTEPexpress; 3 = both controllers
(This identifier specifies which commands are supported by the respective controller)
A description of the commands that have no DLL call is given in Chapter 4. (these must be transmitted with the DLL call "SendString")

API-Command	Short description	LSTEP-Command	X
Connect	Connect with LSTEP	-	3
ConnectEx	Connect with LSTEP	-	3
ConnectSimple	Connect with LSTEP	-	3
CreateLSID	Creates an ID No for the use of the LSTEP4X APIs	-	3
Disconnect	Disconnect LSTEP	-	3
EnableCommandRetry	With this function repeated sending of commands can be switched On/Off in case of a fault.	-	3
FlushBuffer	Delete the input buffer	-	3
FreeLSID	Sets the created ID No free again	-	3
LoadConfig	Load LSTEP configuration (interface, axis settings, controllers) from INI-file.	-	3
SaveConfig	Save LSTEP configuration (interface, axis settings, controllers) into INI-file.	-	1
SendString	Send string to LSTEP	-	3
SendStringPosCmd	Moving command, which awaits confirmation , send to LSTEP as a string	-	3
SetAbortFlag	Set flag to terminate the communication with the LSTEP	-	3
SetCommandTimeout	Sets the timeout times for waiting for the feedback for positioning and calibration		
SetControlPars	Transmits the parameters, which were loaded with LS_LoadConfig to the LSTEP.	-	3
SetCorrTblOff	Deactivate axis correction	-	3
SetCorrTblOn	Activate axes correction in x/y-matrix with linear interpolation	-	3
SetExtValue	Switch on extensions	-	3
SetFactorMode	Position value-Conversion for ‚krumme' spindle pitch	-	1
SetLanguage	Set language for LSTEP-API (log / messages)	-	3
SetProcessMessagesProc	Enables the replacement of the internal message-dispatching procedure of the LStep API	-	3
SetShowCmdList	LStep-API Command list On/Off	-	3
SetShowProt	Interface protocol On/Off	-	3
SetWriteLogText	Switch on / switch off write log file LSTEP4.log (Writing in LSTEP4-log is normally switched off)	-	3
SetWriteLogTextFN	switch On/Off writing of the interface-protocol in a certain file	-	3
	Set baud rate	(?) !baud	3

Controller-Info

API-Command	Short description	LSTEP-Command	X
GetSerialNr	Read serial number of the controller	?readsn	3
GetVersionStr	Gives the current Firmware version number	?ver	3
GetVersionStrDet	Read out detailed version number of Firmware	?det	3
GetVersionStrInfo	Gives detailed information about version number	?iver	3

Settings

API-Command	Short description	LSTEP-Command	X
ConfigMaxAxes	Number of axes used	(?) !configmaxaxis	3
GetAccel	Inquiry of acceleration	?accel	3
GetAccelJerk	Ask jerk during acceleration	?acceljerk	2
GetActiveAxes	Delivers enable axes	?axis	3
GetAxisDirection	Inquiry of reverse-turning direction	?axisdir	3
GetCalibBackSpeed	Reads the speed, with which the axis are moved back during calibration	?calbspeed	1
GetCaliboffset	Inquiry of calibration-offset	?caliboffset	3
GetCalibrateDir	Inquiry reverse preceding sign when calibrating	?caldir	3
GetCalibRMAccel	Ask acceleration for calibration and stroke measurement	?calibrmaccel	2
GetCalibRMBackSpeed	Speed at which movement away from the limit switch takes place for calibration and range measuring.	?calibrmbspeed	2
GetCalibRMJerk	Jerk acceleration for calibration and stroke measurement	?calibrmjerk	2
GetCalibRMVel	Ask speed for calibration and stroke measurement	?calibrmvel	2
GetCurrentDelay	Indicates time delay for current reduction	?curdelay	3
GetDeceleration	Ask for delays	?decel	2
GetDeceljerk	Ask jerk during delay	?deceljerk	2
GetDimensions	Inquiry dimensions of the axes	?dim	3
GetGear	Inquiry- gear transmission	?gear	1
GetGearDenominator	Gear factor / Denominator	?geardenominator	2
GetGearNumerator	Gear factor / Enumerator	?gearnumerator	2
GetJoystickFilter	Indicates, if the filtering and hysteresis is activated in joystick operation	?joyfilter	1
GetMotorCurrent	Inquiry motor current	?cur	1
GetMotorFieldDir	Direction of rotation of motor positive or negative	?motorfielddir	2
GetMotorMaxVel	Max. speed that can be set for the motor	?motormaxvel	2
GetMotorTablePatch	Indicates, if the correction table is activated.	?mtpatch	1
GetMotorType	Query set motor type	?motortype	2
GetOutFuncLev	Indicates the speed when the current will be switched, from parameterised current to maximum current.	?opfl	1
GetPitch	delivers spindle pitch	?pitch	3
GetPowerAmplifier	Power amplifiers On/Off (only for LS44 + Express)	?pa	2
GetReduction	Inquiry of current reduction	?reduction	3
GetRefSpeed	Reads the reverse speed, the axes move while searching the reference mark.	?calrefspeed	1
GetRMOffset	Inquiry RM-Offset	?rmoffset	3
GetSpeedPoti	Indicates if the potentiometer On/Off	?pot	3
GetStopAccel	Delivers the brake acceleration, if the stop input becomes active.	?stopaccel	1

API-Command	Short description	LSTEP-Command	X
GetStopDecel	Braking acceleration on Stop active	?stopdecel	2
GetStopDecelJerk	Jerk for braking acceleration on Stop active	?stopdeceljerk	2
GetStopPolarity	Read stop input polarity.	?stoppol	3
GetVel	Inquiry speed	?vel	3
GetVelFac	Inquiry speed reduction	?velfac	1
GetVLevel	Delivers the speed limits of the indicated speed range	?vlevel	1
GetXYAxisComp	Inquiry XY-axis overlay	?xycomp	1
LstepSave	Save current configuration in LStep (EEPROM)	save	3
SetAccel	Set acceleration	!accel	3
SetAccelJerk	Adjust jerk for acceleration	!acceljerk	2
SetAccelSingleAxis	Set acceleration for individual axis	!accel x (y,z,a)	3
SetActiveAxes	Enable axes	!axis	3
SetAxisDirection	Reverse turning direction	!axisdir	3
SetCalibBackSpeed	Set speed at which movement away from the limit switch takes place	!calbspeed	1
SetCaliboffset	Calibration Offset	!caliboffset	3
SetCalibrateDir	Reverse preceding sign when calibrating	!caldir	3
SetCalibRMAccel	Set acceleration for calibration and stroke measurement	!calibrmaccel	2
SetCalibRMBackSpeed	Speed at which movement away from the limit switch takes place for calibration and range measuring.	!calibrmbspeed	2
SetCalibRMJerk	Set jerk for calibration and stroke measurement	!calibrmjerk	2
SetCalibRMVel	Set speed for calibration and stroke measurement	!calibrmvel	2
SetCurrentDelay	Time delay for current reduction	!curdelay	3
SetDeceleration	Adjust delay	!decel	2
SetDecelJerk	Adjust jerk for the delay	!deceljerk	2
SetDecelSingleAxis	Set delay for individual axis	!decel x	2
SetDimensions	Set dimensions of the axes	!dim	3
SetGear	Program gear transmission	!gear	1
SetGearDenominator	Gear factor / Denominator	!geardenominator	2
SetGearNumerator	Gear factor / Enumerator	!gearnumerator	2
SetJoystickFilter	Activating/Deactivating the filtering and hysteresis in joystick operation	!joyfilter	1
SetMotorCurrent	Set motor current	!cur	1
SetMotorFieldDir	Direction of rotation of motor positive or negative	!motorfielddir	2
SetMotorMaxVel	Max. speed that can be set for the motor	!motormaxvel	2
SetMotorTablePatch	Correction table ON/OFF	!mtpatch	1
SetMotorType	Set motor type	!motortype	2
SetOutFuncLev	Set the current switch speed	!opfl	1
SetPitch	Set spindle pitch	!pitch	3
SetPowerAmplifier	Switches power amplifiers On/Off for LS44 + Express	!pa	2
SetReduction	Set current reduction	!reduction	3
SetRefSpeed	Sets the reverse speed, the axes move while searching the reference mark.	!calrefspeed	1
SetRMOffset	RM-Offset	!rmoffset	3
SetSpeedPoti	Potentiometer On/ Off	!pot	3
SetStopAccel	Delivers the brake acceleration, if the stop input becomes active.	!stopaccel	1
SetStopDecel	Braking acceleration on Stop active	!stopdecel	2
SetStopDecelJerk	Jerk for braking acceleration on Stop active	!stopdeceljerk	2

API-Command	Short description	LSTEP-Command	X
SetStopPolarity	Set stop input polarity.	!stoppol	3
SetVel	Set speed (velocity)	!vel	3
SetVelFac	Set speed reduction	!velfac	1
SetVelSingleAxis	Set speed for individual axis	!vel x (y,z,a)	3
SetVLevel	Exclude speed ranges, in which the system shows resonances.	!vlevel	1
SetXYAxisComp	Activate XY-axis overlay	!xycomp	1
SoftwareReset	Reset the software to starting status	reset	3
	Activation of the controller parameters	!validpar	2

Status request

API-Command	Short description	LSTEP-Command	X
GetError	Shows the current error number	?err	3
GetSecurityErr	Reads all statuses and results of the GAL-safety monitoring (only with LS44-controller)	?securityerror	1
GetSecurityStatus	Delivers the current statuses the safety monitoring (only with LS44-controller)	?securitystatus	1
GetStatus	Shows the current condition of the controller	?status	3
	Provides the system status of the axes as a number	?sysstat	2
	Provides the system status of the axes as text	?sysstatus	2
GetStatusAxis	Gives the present status of the individual axes	?statusaxis	3
GetStatusLimit	Delivers the current condition of the software-limits of each axis.	?statuslimit	3
SetAutoStatus	AutoStatus On/Off	!autostatus	3

Moving commands and Position administration

API-Command	Short description	LSTEP-Command	X
Calibrate	Calibrate	!cal	3
CalibrateEx	Only the axes are calibrated, whose corresponding bit was set in the transmitted integer-value.	!cal x (xy,z,a)	3
Clearpos	Sets the position values to 0 (for endless turning axes)	!clearpos	1
GetDelay	Reads the delay of the vector start.	?delay	3
GetDistance	Delivers the distance for LS_MoveRelShort	?distance	3
GetInputTrigMove	Query of the input for external vector start	?itm	1
GetPos	Inquires the current positions of all axes	?pos	3
GetPosEx	Inquires the current encoder or position values of all axes		
GetPosSingleAxis	Inquire the current position of an axis	?pos x (y,z,a)	3
MoveAbs	Move to absolute position	!moa	3
MoveAbsSingleAxis	Move individual axis to absolute position	!moa x (y,z,a)	3
MoveEx	extended moving command		
MoveRel	Move relative vector	!mor	3
MoveRelShort	Move to relative position (short command)	m	3
MoveRelSingleAxis	Move individual axis relatively	!mor x (y,z,a)	3
RMeasure	Measure Table Stroke	!rm	3

API-Command	Short description	LSTEP-Command	X
RmeasureEx	Measure table stroke (The table stroke is only measured for axes for which the relevant bit has been set in the transmitted integer value).	!rm x (xy z)	3
SetDelay	The delay command is used to produce a vector start delay	!delay	3
SetDistance	Set distance (for LS_MoveRelShort)	!distance	3
SetInputTrigMove	Set input for external vector start	!itm	1
SetPos	Set positional values	!pos	3
StopAxes	Stop (all movements are stopped)	a	3
WaitForAxisStop	The function returns, as soon as the selected axes in the bit-mask AFlags reached its goal position.	-	
	Set and read a table position [Table Pos]	(?)!tpos	2
	Move to table position [Move Table Pos Absolut]	!mtpa	2
	Automatic generation of a table with equal sections (e.g. for a rotation of a rotary axis) [Index Table Divider]	(?)!itd	2
	Move to the specified index [Move Index Table]	!mita	2

Joystick and Handwheel

API-Command	Short description	LSTEP-Command	X
GetDigJoySpeed	Read out the set speeds	?speed	3
GetHandwheel	Read hand wheel condition	?hw	1
GetJoystick	Reads the delay of the vector start.	?joy	3
GetJoystickAxes	Enable for joystick	?joyenable	2
GetJoystickDir	Joystick direction	?joydir	3
GetJoystickWindow	Read Joystick-window	?joywindow	3
GetJoyChangeAxis	Read Joystick allocation of the axes	?joychangeaxis	1
	Query filter time constant for input signals	?joyoutpass	2
GetJoyVel	Query max. speed for joystick	?joyvel	2
SetDigJoySpeed	Read Digital joystick and speed .	!speed	3
SetDigJoyOff	Switch off digital joystick	!speed 0	3
SetHandwheelOff	Handwheel Off	!hw 0	1
SetHandwheelOn	Handwheel On	!hw 1 (1-4)	1
SetJoystickAxes	Enable for joystick	!joyenable	2
SetJoystickDir	Joystick direction	!joydir	3
SetJoystickOff	Analogue joystick Off	!joy 0	3
SetJoystickOn	Analogue joystick On	!joy 1 (1-4)	3
SetJoystickWindow	Set joystick window	!joywindow	3
	Set filter time constant for input signals	!joyoutpass	2
SetJoyVel	Set max. speed for the joystick	!joyvel	2
JoyChangeAxis	sets allocation of axes of Joystick	!joychangeaxis	1

Control panel with trackball and joyspeed keys (only for LSTEP-xx/2)

API-Command	Short description	LSTEP-Command	
GetBPZ	Reads the condition of the additional control panel with track ball	?bpz	1
GetBPZJoyspeed	Control panel joystick-speed	?joyspeed	1
GetBPZTrackballBackLash	Read out control panel track ball-back lash	?bpzbl	1
GetBPZTrackballFactor	Read out control panel trackball-factor	?bpztf	1
SetBPZ	Control panel On/ Off	!bpz	1
SetBPZJoyspeed	Control panel joystick-speed	!joyspeed	1
SetBPZTrackballBackLash	Control panel trackball-reverse backlash	!bpzbl	1
SetBPZTrackballFactor	Control panel trackball-factor	!bpztf	1

Manual operation using trackball or jog mode
LSTEP-PCIexpress / LSTEPexpress only)

API-Command	Short description	LSTEP-Command	
	Switch jog mode on or off	tipp	2
	Direction select for the keys for jog mode	tippdir	2
	Enable for jog mode	tippenable	2
	Current reduction for jog mode	tippredcur	2
	Speed for jog mode	tippvel	2
	Filter time constant for input signals	tippoutpass	2
	Switch trackball on or off	tb	2
	Direction select for the trackball	tbdir	2
	Enable for the trackball	tbenable	2
	Current reduction for the trackball	tbredcur	2
	Assignment of the trackball axes	tbtoaxis	2
	Velocity for the trackball	tbvel	2
	Filter time constant for input signals	tboutpass	2

Limit switch (Hardware a. Software)

API-Command	Short description	LSTEP-Command	
GetAutoLimitAfterCalibRM	Indicates if the internal software limits will be set during calibration and table stroke measuring.	?nosetlimit	3
GetLimit	Set travel limits	?lim	3
GetLimitControl	Reads, if travel range monitoring is active	?limctr	3
GetLimitControlMode	Supply the mode for monitoring of the Software limits	?limmode	1
GetSwChange	Query whether limit switches have been changed	?swchange	2
GetSwitchActive	Read status of limit switch	?swact	3
GetSwitches	Reads the status of all limit switches	?readsw	3
GetSwitchPolarity	Reads limit switch polarity	?swpol	3
SetAutoLimitAfterCalibRM	Prevents that the internal software limits are set during calibration and table stroke measuring.	!nosetlimit	3
SetLimit	Set travel limits	!lim	3
SetLimitControl	Range monitoring	!limctr	3
SetLimitControlMode	Place the mode for monitoring of the Software limits.	!limmode	1
SetSwChange	Change zero and end position limit switches	!swchange	2
SetSwitchActive	Read status of limit switch On / Off	!swact	3

API-Command	Short description	LSTEP-Command	
SetSwitchPolarity	Set limit switch polarity	!swpol	3

Digital and analogue Input and Output

API-Command	Short description	LSTEP-Command	X
GetAnalogInput	Reads the current status of an analogue channel	?anain	3
GetAnalogInputs2	Reads the current status of the analogue channels PT100, MV and V24	--	
GetDigitalInputs	Read all input pins	?digin	3
GetDigitalInputsE	read additional digital inputs (16-31)	?edigin	1
SetAnalogOutput	Set analogue output	!anaout	3
SetDigIO_Distance	Function of the digital inputs / outputs	(digfkt)	1
SetDigIO_EmergencyStop	Function of the digital inputs / outputs Allocating of the Emergency Stop pin	(digfkt)	1
SetDigIO_Off	"Off" function of the digital inputs/outputs	(digfkt)	1
SetDigIO_Polarity	Set polarity	!digfkt 16 0 0	1
SetDigitalOutput	Set output pin	!digout x	3
SetDigitalOutputs	Set digital outputs (0-15)	!digout 0-15	3
SetDigitalOutputsE	Set additional outputs (16-31)	!edigout	1

Clock pulse Forward / Back

API-Command	Short description	LSTEP-Command	X
GetFactorTVR	Reads factor for clock pulse Forward/ Back	?tvrf	3
GetTVRMode	Read setup of clock pulse Forward /Back (= TVR Mode)	?tvr	3
	Clock forward/backward mode	?tvrm	
SetFactorTVR	Factor for clock pulse Forward/ Back	!tvrf	3
SetTVRMode	Set clock pulse Forward / Back	!tvr (0-4)	3
	Clock forward/backward mode	!tvrm	

Clock pulse Forward /Back via Interface

API-Command	Short description	LSTEP-Command	X
SetTVRInPulse	Clock pulse Forward /Back via Interface	px/nx	1

Clock pulse Forward / Back for the additional axes

API-Command	Short description	LSTEP-Command	X
GetAccelTVRO	Reads the set accelaration for the additional axes.	?tvroa	1
GetPosTVRO	Read position values of the additional axis	?tvropos	1
GetStatusTVRO	Delivers the current status of the additional axis	?tvrostatus	1
GetTVROutMode	Read settings of the additional axis	?tvrout	1
GetTVROutPitch	Reads the spindle pitch of the additional axis	?tvropitch	1
GetTVROutResolution	Reads the resolution of the power amplifier which is to be controlled	?tvrores	1
GetVelTVRO	Reads the set speed of the additional axis	?tvrov	1
MoveAbsTVROSingleAxis	Move individual axis to absolute position	!tvromoa x	1
MoveAbsTVRO	Move to absolute position	!tvromoa	1
MoveRelTVROSingleAxis	Move single axis absolute	!tvromor x	1
MoveRelTVRO	Move relative vector	!tvromor	1
SetAccelSingleAxisTVRO	Acceleration of single additional axis	!tvroa x	1
SetAccelTVRO	Set acceleration	!tvroa	1
SetPosTVRO	Set positional values	!tvropos	1
SetTVROutMode	Set clock pulse Forward / Back	!tvrout	1
SetTVROutPitch	Set spindle pitch	!tvropitch	1
SetTVROutResolution	Sets the resolution of the power amplifier which is to be controlled	!tvrores	1
SetVelSingleAxisTVRO	Set speed of the additional axis	!tvrov x	1
SetVelTVRO	Set speed of the additional axis	!tvrov	1

Encoder-Settings

API-Command	Short description	LSTEP-Command	X
ClearEncoder	Set encoder-counter to zero	!pos 0 0 0 0	3
GetEncoder	Reads all encoder positions	!encpos1 ?pos	3
GetEncoderActive	Reads, which encoders are activated after the calibration.	?encmask	3
GetEncoderMask	Read encoder statuses	?enc	3
GetEncoderPeriod	Read length of encoder period	?encperiod	3
GetEncoderPosition	Read encoder position setting	?pos (to !encpos 1)	3
GetEncoderRefSignal	Reads if interpret reference signal from encoder when calibration is done	?encref	3
SetEncoderActive	This function is used to select which encoder is to be activated after calibration.	!encmask	3
SetEncoderMask	Activate or deactivate encoder	!enc	
SetEncoderPeriod	Set length of encoder period	!encperiod	3
SetEncoderPosition	Encoder position display On/Off	!encpos 1 !pos 0 0 0	3
SetEncoderRefSignal	Interpret reference signal from encoder when calibration is done	!encref	3
	Encoder error	(?) !encerr	1
	Change count direction of the encoder	(?) !encdir	2
	Set encoder type	(?) !enctyp	2
	Number of encoder periods per motor revolution	(?) !encpolepairs	2
	Positioning of transmitter input and axis	(?) !enctoaxis	2
	Polarity of reference marks of QEP-Encoder	(?) !encrefpol	2

Controller Setting

API-Command	Short description	LSTEP-Command	X
ClearCtrFastMoveCounter	This function sets Fast Move Counters of all axes to zero.	!ctrfmc 0	1
GetController	Read controller mode	?ctr	1
GetControllerCall	Reads controller call time	?ctrc	1
GetControllerFactor	Reads controller factor	?ctrf	1
GetControllerSteps	Reads controller steps length	?ctrs	1
GetControllerTimeout	Reads controller timeout	?ctrt	1
GetControllerTWDelay	Read controller relay	?ctrd	1
GetCtrFastMove	Reads setting of the Fast Move Function	?ctrfm	1
GetCtrFastMoveCounter	Read amount od execuded FastMove functions to 0	?ctrfmc	1
GetTargetWindow	Reads the target window	?twi	3
SetController	Set controller mode	!ctr	1
SetControllerCall	Call controller	!ctrc	1
SetControllerFactor	Controller factor	!ctrf	1
SetControllerSteps	Controller steps	!ctrs	1
SetControllerTimeout	Controller timeout	!ctrt	1
SetControllerTWDelay	Controller delay	!ctrd	1
SetCtrFastMoveOff	Fast Move Function „OFF“	!ctrfm 0	1
SetCtrFastMoveOn	Fast Move Function „ON“	!ctrfm 1	1
SetTargetWindow	Target window	!twi	3
	Target window (alternative to "twi")	(?)!poswindowrange	
	Kp- band of the position controller	(?) !posconkp	2
	Filter time constant for the output filter (low pass) of the position controller	(?) !posconoutpass	2
	Axis enable for position controller	(?) !posconenable	2
	Switch-on and switch-off of the position controller	(?) !poscon	2
	Deviation check / range	(?) !deviationsrange	2
	Deviation check / time frame	(?) !deviationstime	2
	Switch deviation check On or Off	(?) !deviationcheck	2

Trigger-Output

API-Command	Short description	LSTEP-Command	X
GetTrigCount	Read Trigger counter	?trigcount	3
GetTrigger	Read Trigger setting	?trig	3
GetTriggerPar	Reads Trigger-Parameter	?triga ?trigm ?trigs ?trigd	3
SetTrigCount	Read Trigger counter	!trigcount	3
SetTrigger	Trigger On/ Off	!trig	3
SetTriggerPar	Trigger parameters	!triga !trigm !trigs !trigd	3

Snapshot-Input

API-Command	Short description	LSTEP-Command	X
GetSnapshot	Reads the current Snapshot-condition	?sns	3
GetSnapshotCount	Snapshot Counter	?snsc	3
GetSnapshotFilter	Reads input filter (snapshot-filter)	?snsf	3
GetSnapshotPar	Read Snapshot-Parameter	?snsl ?snsm ?sns	3
GetSnapshotPos	Read snapshot position	?snsp	3
GetSnapshotPosArray	Read snapshot-position from array	?snsa	3
SetSnapshot	Snapshot On/Off	!sns	3
SetSnapshotFilter	Set input filter for rebounding switches.	!snsf	3
SetSnapshotPar	Snapshot parameters	!snsl !snsm !sns	3

Appendix:

Manufacturer's Declaration LSTEPexpress

CE Declaration of Conformity in acc. with EC Directives 2004/108/EC and 2006/95/EC

We hereby declare that the product,

Type : **Motor Controller LSTEPexpress,**

in the version placed on the market by us, complies with the basic safety and health requirements laid out in the EC Directives listed below.

Any modifications of the product made without our authorization render this declaration null and void.

EMC Directive 2004/108/EC:

Applied, harmonized standards:

EN61800-3 - 2004	Adjustable speed electrical power drive systems
EN61000-6-3 - 2007	Emission - Generic standards - residential environments
EN61000-6-2 -2005	Immunity- Generic standard - industrial environments
EN61000-3-2 - 2006	Emission harmonic current
EN61000-3-3 – 1995+A1, A2	Emission flicker
EN60601-1-2 – 2001+A1	Product standard for medical electrical equipment

The presumption of conformity is based on the documentation of the test results given in test report no. XXXXXXX

Low-Voltage-Guideline 2006/95/EEC:

Applied, harmonized standards:

EN 6010:2001	Safety regulations for electrical measurement, control, regulation and laboratory devices - Part 1: General requirements

Hüttenberg 19.04.2010

Place	*Date*	*Signature*	*Technical Development Manager*

Manufacturer’s Declaration LSTEP-PCIexpress

We hereby declare that the Controller LSTEP-PCIexpress is not a ready-to-use or ready-for-operation device in the sense of the "Device Safety Law", the EMC Law", or the EC Machinery Directive, but is rather a component.

Its final mode of action is only given when it is integrated into the User's construction. Compliance of the User's construction with the applicable safety regulations and legislation is the User's responsibility.

Instructions and recommendations for the installation and proper operation are included in the operating manual.

It is prohibited to put the controller into operation until such time as it has been ascertained that all legal protection and safety requirements have been met. Refer to the manufacturer's declaration for the LSTEPexpress for the underlying requirements.