Programming Interface

LSTEP-API

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

Contents

1 Introduction

The LSTEP-API (programming interface for the LSTEP precision positioning systems) is intended to assist 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. Two DLLs are available: The LSTEP4X.DLL and the LSTEP64.DLL.

1.1 Included Functions

- Windows 32-bit DLL
- Windows 64-bit DLL
- Support of the motor controllers LSTEP xx, LSTEP xx/2, LSTEP-PC, ECO-STEP, LSTEP-44, LSTEP-PCI, LSTEP-PCI express and LSTEP express
- Activation via RS232, USB, Ethernet, ISA, PCI (DPRAM), or PCIexpress. Interface control depend on the controller type.
- Automatic identification of the connected controller
- Configuration of the controller
- Execution of all commands supported by the controller
- Up to 4 axes
- Multithreading capable

1.2 System requirements

With the LSTEP-API it is possible to develop applications with MS Windows 2000, Windows XP, Windows Vista, Windows 7, Windows 8, Windows 8.1.

1.3 Supported Development Environments

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

Borland/Inprise Delphi 3-7, Embarcadero Delphi XE2

Microsoft Visual C++ 2013

Microsoft Visual C# 2010

National Instruments LabVIEW 32 and 64 Bit

They 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 used by other applications or DLLs.)

2 DLL Interface

2.1 LSTEP-API

The main component of the LSTEP-APIs is the file LSTEP4X.DLL or LSTEP64.DLL. You use this DLL for developing your own programs, to configure one or multiple LSTEPS, to send commands, to inquire position values or inputs/outputs, etc.

2.2 General notes

The DLL LSTEP4X.DLL or LSTEP64.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 Zero indicates the error-free execution of the function; if errors (e.g. timeouts) occur, the relevant error code (see Section 6 Error codes) is returned.

The first parameter sent for all functions of the API is a 32-bit integer value (between 1 and 32), which indicates the number of the LSTEP to where the command is supposed to be sent.

The function LSX_CreateLSID must be used to create such an ID-value. With a call of LSX_FreeLSID, an ID-value is released again (see Delphi-example).

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

2.3 Integration in Delphi

All function names of the LSTEP-API start with "LSX_" for easier differentiation (see LStep4x.pas). In order to be able to use the functions of the LSTEP-API, the LSTEP4X.pas or LStep64.pas must be included in the uses clause of the relevant unit and be part of one of the pre-set search paths.

Delphi example for the parallel control of two LSTEP controllers

Required files: LSTEP4X.DLL and LSTEP4X.pas or LSTEP64.DLL and LSTEP64.pas

```
uses … LSTEP4X, …
...
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;
```

2.4 Integration in Visual C++

For Visual C++, an encapsulation of the LSTEP4X.DLL orLSTEP64.DLL has been created. The class CLStep4X or CLStep64 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)

You do not have to call the functions LSX_CreateLSID and LSX_FreeLSID in C++ for using the LSTEP4X.DLL or LSTEP64.DLL, since the wrapper class CLStep4X or CLStep64 administers the integer value indicating the number of the LSTEP itself. This means, the methods of CLStep4X do not have any additional parameter for the number of the LSTEP.

Visual C++ example for the parallel control of two LSTEP controllers
Required files: LSTEP4X.DLL, LSTEP4X.h and LSTEP4X.cpp or LSTEP64.DLL, LSTEP64.h and LSTEP64.cpp

```
...
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;
```

2.5 Integration in LabVIEW

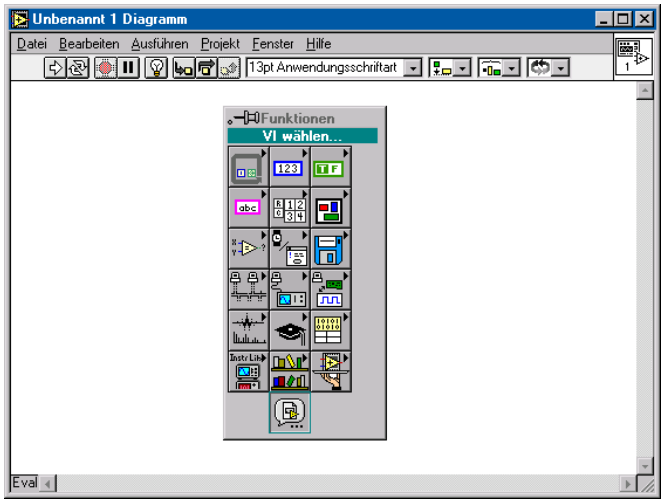
NI LabVIEW is a development environment based on the graphic programming language C. It allows for facilitated and quick programming using graphic symbols. Complicated 32-bit or 64-bit programs can be created, thus ensuring the required execution speed for control, test and measuring applications.

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).

For the integration of the LSTEP-API (LSTEP4X.DLL or LSTEP64.DLL) VI libraries (LSTEP4X.LLB or LSTEP64.LLB) containing a collection of VIs have been created. These individual VIs (e.g. LS4 ConnectSimple.vi) encapsulate the respective LSTEP API functions. The LSTEP4X.DLL is used by means of the "Call Library Function" (calling ext. libraries).

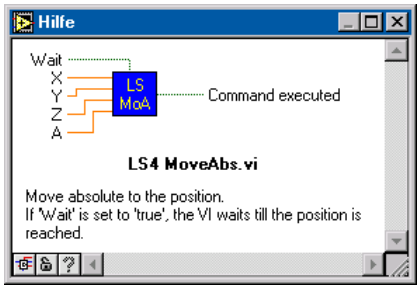
2.5.1 Procedure for using an LSTEP4 VI

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

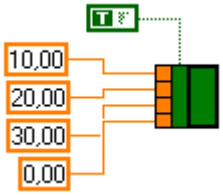


4. Select VI …
5. Open the supplied VI Library LSTEP4X.LLB in the file dialog, and then select the required command (e.g. LS4 MoveAbs.vi)
6. Place VI in the diagram

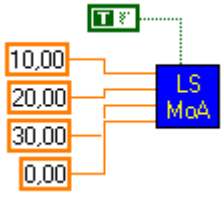
The key combination Ctrl+H opens a help window, which gives you information about the VI on which the mouse pointer is currently located



The transmission of parameters to SubVIs is done by terminals, which have to be "wired". 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. One of several ways to do this: Click the right mouse button on the required terminal then on the menu item "Create a constant".



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

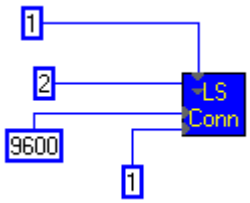


For details of the VI terminal assignment, please refer to the documentation for the API function in question, where you will find a diagram, which is also shown in the help window of LabVIEW. The parameters are more or less identical to those of the DLL function: Only are there some differences as regards functions to which bit masks are transmitted as parameters (e.g. LS4 SetActiveAxes.vi).

The LSTEP4 VIs have a terminal called “Command executed”. If this Boolean 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 using the VI “LS4 ConnectSimple.vi”. It initialises the interface and detects the connected LSTEP.

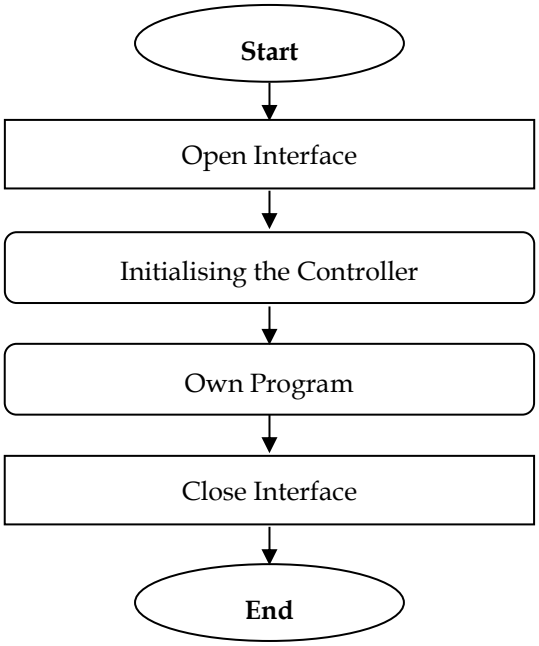
Example for RS232 (COM2 and 9600 Baud):



3 Structure of own programs with the API

The following figure shows the program flow diagram according to which programs for controlling positioning systems should be structured. The functions used are listed in the LSTEP-API description where they are described in more detail.

The LSTEP controllers are pre-configured before delivery. This configuration, however, does not cover any type of application and has to be adjusted to the relevant application. This adjustment has to be made after opening the interface. Subsequently, any user program can be written by using the LSTEP-API. The interface has to be closed upon terminating the program.



3.1 Open Interface

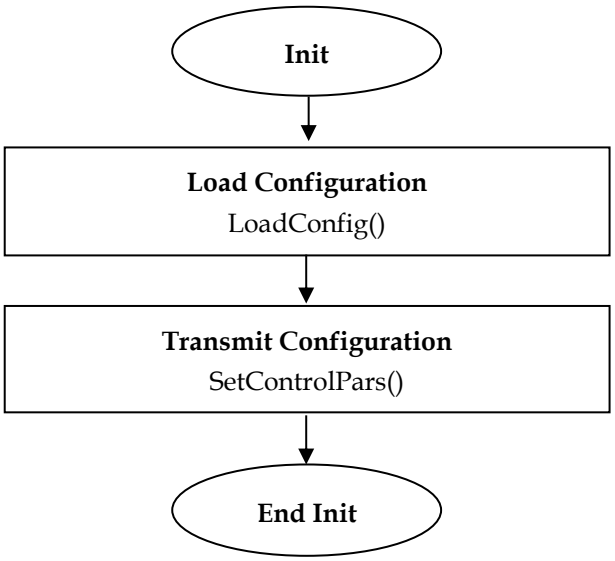
The interface is opened using one of the functions Connect, ConnectEx or ConnectSimple, with ConnectSimple being recommended for establishing the connection. The interface to be opened for connecting to a controller depend on the control type and the interface settings. The default settings of the interface may be taken from the documentation of the relevant controller. In addition, the following table may be used:

Controller series	Controller name	Command set	Supported ConnectSimple interface type (default)						
			1	2	3	4	5	6	11
	MCL	Register	x						
LSTEP Series	LSTEP-xx	Register	x						
	LSTEP-PC	Register			x				
LSTEP 2000 Series	LSTEP-PCI 32Bit-Windows	Ipreter Register	x			x			
	LSTEP-PCI 64Bit-Windows	Ipreter Register						x	
	LSTEP-xx/2	Ipreter Register	x						
	LSTEP-44	Ipreter	x						
	ECO-STEP	Ipreter Register	x						
	ECO-Mot	Ipreter Register	x						
	ECO-Drive	Ipreter Register	x						
LSTEP express Series	LSTEP-PCI express	Ipreter					x		x
	LSTEP express	Ipreter					x		x

3.2 Initialising the Controller

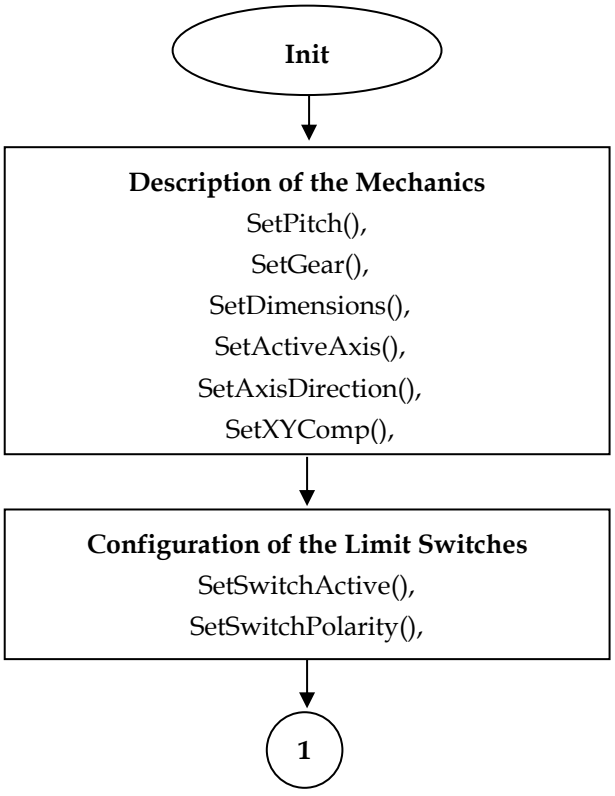
3.2.1 General

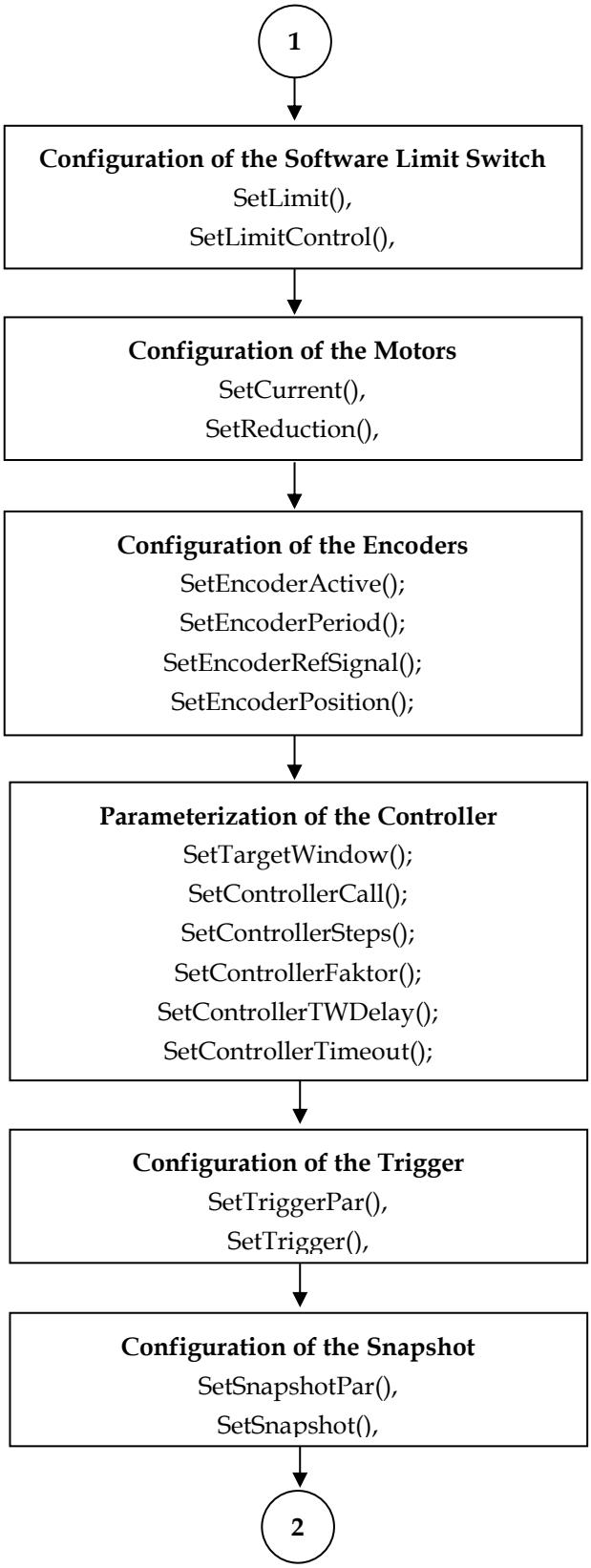
Prior to starting the own program part, the LSTEP controller should be configured. For this purpose, e.g. the commissioning software WIN-Commander may be used to create a configuration file. This file may subsequently be loaded by the API and transmitted to the controller.

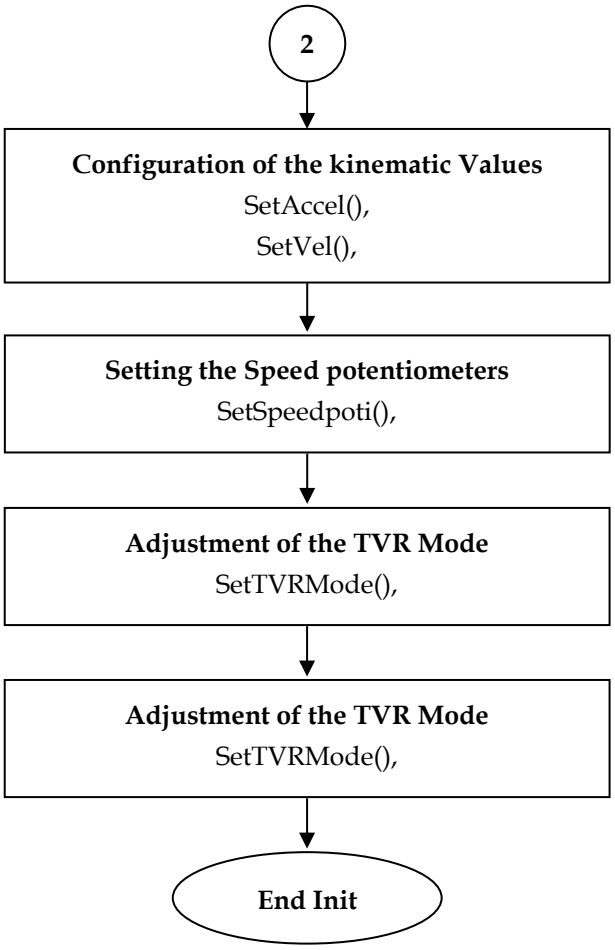


The settings in the controller may be saved if the API makes no general modifications of the control settings. You can use the commissioning software WIN-Commander for this purpose, too.

Apart from these possibilities, the API offers functions for the configuration of the controller, which are shown in the flow diagram for a LSTEP controller below.




1
Configuration of the Software Limit Switch
SetLimit(),
SetLimitControl(),
Configuration of the Motors
SetCurrent(),
SetReduction(),
Configuration of the Encoders
SetEncoderActive();
SetEncoderPeriod();
SetEncoderRefSignal();
SetEncoderPosition();
Parameterization of the Controller
SetTargetWindow();
SetControllerCall();
SetControllerSteps();
SetControllerFaktor();
SetControllerTWDelay();
SetControllerTimeout();
Configuration of the Trigger
SetTriggerPar(),
SetTrigger(),
Configuration of the Snapshot
SetSnapshotPar(),
SetSnapshot(),
2



3.2.2 LSTEP 2000 Series

A variety of API functions is available for configuring the controllers of the LSTEP 2000 series. Apart from this, WIN-Commander 4 and the menu item "Save INI file in…" in the main menu → Options can be used to create a configuration file. In addition to the controller configuration, this configuration file also includes the configuration of WIN-Commander 4. If only the controller configuration is to be saved, this is possible via "Save settings" in the main menu → Control.

The relevant configuration file may be read out by using the API command LoadConfig and sent to the controller by using the command SetControlPars. Subsequently, the API command LStepSave may be used to save the configuration in the controller so that it is immediately loaded after the next start of the controller. The settings may furthermore be saved in the controller from the WIN-Commander, which spares the manual loading of the file at the program start.

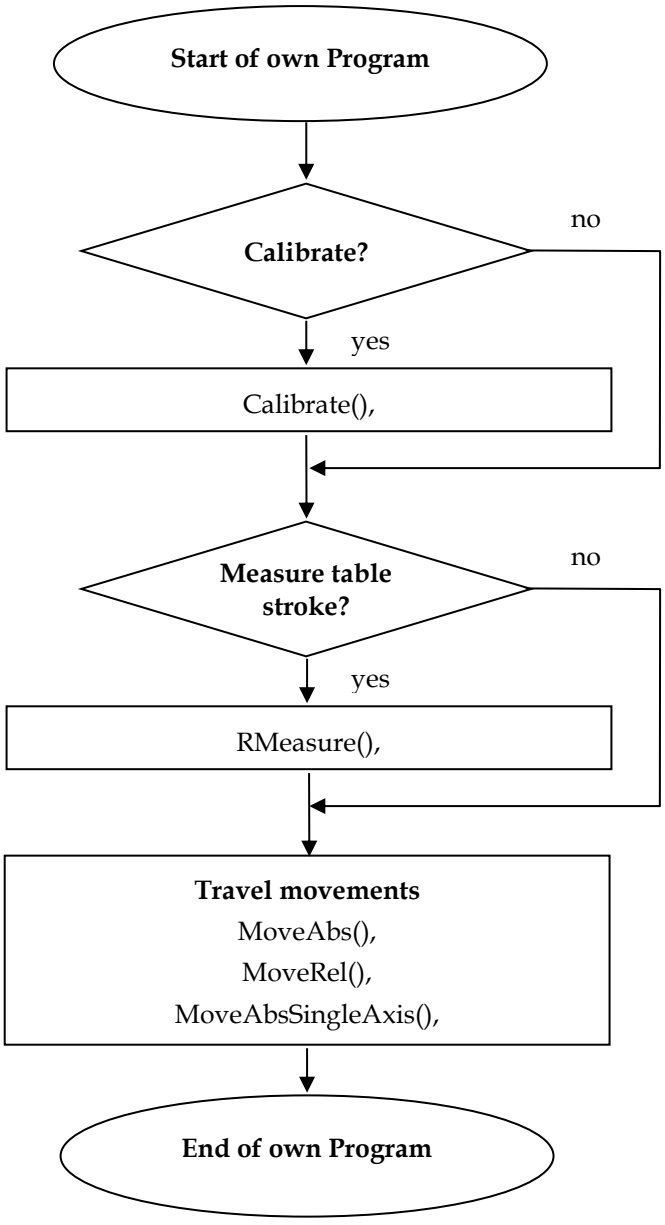
3.2.3 LSTEP express / LSTEP-PCI express controllers

LSTEP express has significantly more setting options than the controllers of the LSTEP 2000 series, so that not all elements are available as separate API function. For this reason, it is recommendable to configure the LSTEP express by using the WIN-Commander 5. This con-

figuration may be saved in a file by using the menu item “Export configuration” in the main menu → Control in the WIN-Commander 5. After export, this file may be read by using the API command LoadConfig and be sent by using the command SetControlPars. The API command LStepSave is used to save the configuration in the controller so that it is immediately loaded after the next start of the controller. The settings may also be saved in the controller from the WIN-Commander, which spares the manual loading of the file at the program start.

3.3 Own Program part

In the own program part, the user can program the desired functionality of the controller. This includes the carrying out of positioning movements dependent on the digital I/O conditions, as well as the setting of trigger signals dependent on the position, etc.



4 Functions

4.1 Brief description of the API commands

Table structure:

The tables below include brief descriptions regarding the individual API functions. For this purpose, the function name is listed in the "Command" column. In the next column, this function is described briefly. The column designated "X" indicates the controller supporting this API command.This designation has the following structure:

Values of the column "X": 1 = LSTEP; 2 = LSTEP express; 3 = both controllers

The last column includes a link or the page number of the detailed description of the function.

Attention!

Only the DLL calls are described in this brief description.

All commands not existing as API function have to be sent by using the DLL call "SendString". The functions available and their relevant descriptions can be looked up in the appropriate documentation of the controller.

4.1.1 API-Configuration / Interface Configuration

Command	Brief description	X	Page
CreateLSID	Create an ID number	3	37
FreeLSID	Releases the created ID number again	3	38
Connect	Connect with the LSTEP	3	39
ConnectEx ConnectExW	Connect with the LSTEP	3	39
ConnectSimple ConnectSimpleW	Connect with the LSTEP (recommended)	3	40
Disconnect	Disconnect LSTEP	3	41
SetExtValue	Activate extensions	3	42
SetProcessMessagesProc	Allows the substitution of the internal message dispatching procedure of the LSTEP-API	3	43
SetOsziCallBackFct	Transfers a global callback function for asynchronous results to the API. (see Section 5.1)	2	44
SetExtCallBackFct	Transfers a global callback function for asynchronous results to the API. (See 5.2)	2	45
SetLanguage SetLanguageW	Set language for the LSTEP-API (log/messages)	3	46
TranslateErrMsg	Translates an error message sent by the controller	2	46
FlushBuffer	Deletes the input buffer	3	47

4.1.2 Control and API information

Command	Brief description	X	Page
GetAPIVersion GetAPIVersionW	Shows version number of LSTEP-API	3	63
GetSerialNr GetSerialNrW	Read out controller serial number	1	63
GetVersionStr GetVersionStrW	Feeds back the current firmware version number	3	64
GetVersionStrDet GetVersionStrDetW	Read out firmware configuration	1	64
GetVersionStrInfo GetVersionStrInfoW	Read out supplement of current version number	3	65

4.1.3 Status requests

Command	Brief description	X	Page
GetError	Shows the current error number	3	65
Quit	Acknowledges that an error is rectified	2	66
GetSecurityErr	Reads all statuses and results of GAL safety monitoring (only with LS44-controllers)	1	67
GetSecurityStatus	Reads the current statuses of safety monitoring system	1	69
GetStatus GetStatusW	Shows the current status of the controller	3	70
GetStatusAxis GetStatusAxisW	Shows the present status of the individual axes	3	71
SetAutoStatus	Deactivates the AutoStatus .	3	72
GetStatusLimit GetStatusLimitW	Shows the current state of the software limits of each axis.	3	73
GetSysstat, GetSysStatus	Shows the current systemstate of the axes	2	74
GetStopStatus	Shows the state of the stop input	2	75
GetPowerAmplifierStatus	Reading the Status of power amplifiers including switching processes	2	75
GetPTemp	Reading the power amplifier temperature sensors	2	76

4.1.4 Parameter handling

Command	Brief description	X	Page
SetControlPars	Transmits the parameters loaded with LSX_LoadConfig to the LSTEP.	3	76
LStepSave	Saves current configuration in LSTEP (EEPROM)	3	77
SoftwareReset	Restarts the controller	3	77

4.1.5 Axis and motor configuration

Command	Brief description	X	Page
ValidConfig	Activates all parameters which are dependend on ValidConfig	2	78
ValidPar	Transmits the parameters to the control unit	2	78
ConfigMaxAxes	Sets the number of axes used	3	79
GetDimensions	Inquires the axis dimensions	3	79
SetDimensions	Sets the axis dimensions	3	80
GetActiveAxes	Shows the released axes	3	81
SetActiveAxes	Shows the released axes	3	82
GetAxisMap	Reading configuration of the hardware-axes to the axes of the user-interface	2	82
SetAxisMap	Mapping of the hardware-axes to the axes of the user-interface	2	83
GetAxisDirection	Inquires the reversal of direction	3	83
SetAxisDirection	Sets the reversal of direction	3	84
GetGear	Reads the gear factor	1	84
SetGear	Sets gear factor	1	85
GetGearDenominator	Inquires the denominator of the gear ratio	2	85
SetGearDenominator	Sets the denominator of the gear ratio	2	86
GetGearNumerator	Inquires the numerator of the gear ratio	2	86
SetGearNumerator	Sets the numerator of the gear ratio	2	87
GetMotorCurrent	Inquires the motor current	3	87
SetMotorCurrent	Sets the motor current	3	88
GetMotorpeakCurrent	Inquires the motor peak current	2	88
SetMotorpeakCurrent	Sets the motor peak current	2	89
GetMotortempSensor	Inquires the state of the motor temperature sensor	2	89
SetMotortempSensor	Sets the state of the motor temperature sensor	2	89
GetMotortempSensormin	Inquires the lowest acceptable motor temperature resistance	2	90
SetMotortempSensormin	Sets the lowest acceptable motor temperature resistance	2	90

4.1.6 Kinematics

Command	Brief description	X	Page
GetAccel	Inquires the acceleration	3	119
SetAccel	Sets the acceleration	3	120
SetAccelSingleAxis	Sets the acceleration of a single axis	3	120
GetAccelJerk	Inquires the jerk for acceleration	2	121
SetAccelJerk	Sets the jerk for acceleration	2	121
GetDeceleration	Inquires the deceleration	2	122
SetDeceleration	Sets the deceleration	2	122
SetDecelSingleAxis	Sets the deceleration of a single axis	2	123
GetDecelJerk	Inquires the jerk during deceleration	2	123
SetDecelJerk	Sets the jerk during deceleration	2	124
GetVel	Inquires the velocity of all axes	3	124
SetVel	Sets the velocity of all axes	3	125
SetVelSingleAxis	Sets the velocity for a single axis	3	125
GetVelFac	Inquires the velocity reduction of all axes	1	126
SetVelFac	Sets the velocity reduction of all axes	1	126
SetVelFacSingleAxis	Sets the velocity reduction of a single axis	1	127
GetVLevel	Shows the hidden speed	1	128
SetVLevel	Sets the hidden speeds at which resonances occur	1	129
GetSpeedPoti	Indicates whether the speed potentiometer is activated or deactivated	1	129
SetSpeedPoti	Activates or deactivates the speed potentiometer	1	130
GetStopAccel	Shows the braking acceleration for an active stop	1	130
SetStopAccel	Sets the braking acceleration for an active stop	1	131
GetStopDecel	Reads the value at which the axis shall decelerate in case of a Stop signal	2	131
SetStopDecel	Sets the value at which the axis shall decelerate in case of a Stop signal	2	132
GetStopDecelJerk	Inquires the jerk during system delay in case of an Emergency Stop signal	2	132
SetStopDecelJerk	Sets the jerk during system delay in case of an Emergency Stop signal	2	133
GetRoomAccelJerk	Reads Acceleration jerk (movement with room parameters)	2	133
SetRoomAccelJerk	Sets Acceleration jerk (movement with room parameters)	2	134
GetRoomDecelJerk	Reads Deceleration jerk (movement with room parameters)	2	134

Command	Brief description	X	Page
SetRoomDecelJerk	Sets Deceleration jerk (movement with room parameters)	2	135
GetRoomStopJerk	Reads Jerk at an emergency stop event (movement with room parameters)	2	135
SetRoomStopJerk	Sets Jerk at an emergency stop event (movement with room parameters)	2	136
GetRoomAccel	Reads Acceleration (movement with room parameters)	2	136
SetRoomAccel	Sets Acceleration (movement with room parameters)	2	137
GetRoomDecel	Reads Deceleration (movement with room parameters)	2	137
SetRoomDecel	Sets Deceleration (movement with room parameters)	2	137
GetRoomStopDecel	Reads Deceleration at an emergency stop event (movement with room parameters)	2	138
SetRoomStopDecel	Sets Deceleration at an emergency stop event (movement with room parameters)	2	138
GetRoomVel	Reads Velocity (movement with room parameters)	2	139
SetRoomVel	Sets Velocity (movement with room parameters)	2	139
GetSpeedEnable	Reads Velocity or speed selection release	2	139
SetSpeedEnable	Sets Velocity or speed selection release	2	140
GetSpeed	Reads Velocity or speed selection	2	140
SetSpeed	Sets Velocity or speed selection	2	140

4.1.7 Limit switches and software limits

Command	Brief description	X	Page
GetLimitControl	Reads whether if travel range control is active	3	141
SetLimitControl	Sets the travel range control	3	142
GetLimitControlMode	Shows the mode for monitoring the software limits	1	142
SetLimitControlMode	Sets the mode for controlling the software limits	1	143
GetAutoLimitAfterCalibRM	Indicates whether the internal software limits are set during calibration and table stroke measuring	3	143
SetAutoLimitAfterCalibRM	Prevents that the internal software limits are set during calibration and table stroke measuring	3	144
GetLimit	Shows travel range limits	3	145
SetLimit	Sets travel range limits	3	146
GetSwitchActive	Indicates whether limit switches are activated	3	147
SetSwitchActive	Sets the Status of activated or deactivated limit switches	3	147
GetSwitchPolarity	Reads the limit switch polarity	3	148
SetSwitchPolarity	Sets the limit switch polarity	3	149

Command	Brief description	X	Page
GetSwChange	Shows the limit switch settings	2	150
SetSwChange	Sets the value if limit switches are to be changed	2	151
GetSwitches	Reads the status of all limit switches	3	152
GetStopSwitchOffDelay	Reading Delay of poweramplifier switchoff after activating the stop-input	2	152
SetStopSwitchOffDelay	Setting Delay of poweramplifier switchoff after activating the stop-input	2	153
GetMonitoringVelFilter	Reading Time constant of actual value for velocity monitoring	2	153
SetMonitoringVelFilter	Setting Time constant of actual value for velocity monitoring	2	153
GetHaltSignalVel	Reading Velocity level for standstill signal	2	154
SetHaltSignalVel	Setting Velocity level for standstill signal	2	154
GetThresholdSignalVel	Reading Value for velocity threshold signal	2	155
SetThresholdSignalVel	Setting Value for velocity threshold signal	2	155
GetCSOffset	Reading Coordinate system offset	2	155
SetCSOffset	Setting Coordinate system offset	2	156
GetCalibRMOffsetSWAct	Reading Zero point offset to calibration limit switch	2	156
SetCalibRMOffsetSWAct	Setting Zero point offset to calibration limit switch	2	157

4.1.8 Reference travel

Command	Brief description	X	Page
GetCalibRMAccel	Inquires the acceleration during the calibration procedure	2	157
SetCalibRMAccel	Sets the acceleration during the calibration procedure	2	158
GetCalibRMJerk	Inquires the jerk during calibration	2	158
SetCalibRMJerk	Sets the jerk during calibration	2	159
GetCalibBackSpeed	Shows the speed at which limit switches are left	1	159
SetCalibBackSpeed	Sets the positioning speeds for moving out of the limit switches during the calibration procedure	1	160
GetCalibRMBackSpeed	Shows the speed at which limit switches are left	2	160
SetCalibRMBackSpeed	Sets the positioning speeds for moving out of the limit switches during the calibration procedure	2	161
GetCalibRMVel	Inquires the travel velocity during the calibration procedure	2	161
SetCalibRMVel	Sets the travel velocity during the calibration procedure	2	162
GetRefSpeed	Inquires the speed at which the reference mark is searched during calibration	1	162
SetRefSpeed	Sets the speed at which the reference mark is searched during calibration	1	163

Command	Brief description	X	Page
GetCalibOffset	Inquires the calibration offset	3	163
SetCalibOffset	Sets the calibration offset	3	164
GetRMOffset	Inquires the set RM offset	3	164
SetRMOffset	Sets the RM offset	3	165
GetCalibrateDir	Shows whether the sign reversal is set for calibration	3	165
SetCalibrateDir	Sets the sign reversal for calibration	3	166
Calibrate	Starts calibration for all active axes	3	166
CalibrateEx	Starts calibration for specific axes	3	167
RMeasure	Starts table stroke measuring for all active axes	3	167
RmeasureEx	Starts table measuring for specific axes	3	168

4.1.9 Travel commands and position administration

Command	Brief description	X	Page
GetPos	Inquires the current position of all axes	3	168
GetPosEx	Inquires the current encoder or position values of all axes	3	169
GetPosSingleAxis	Inquires the current position of a single axis	3	170
SetPos	Sets the position of all axes	3	170
ClearPos	Sets the Position values to zero (for infinite rotation axes)	1	171
GetDelay	Shows the delay of the vector start	1	171
SetDelay	Sets the delay of the vector start	1	172
GetDistance	Shows the distance started by MoveRelShort	3	172
SetDistance	Sets the distance for MoveRelShort	3	173
GetInputTrigMove	Shows the configuration of Pin1 on the MFP	1	173
SetInputTrigMove	Configures Pin 1 on the MFP	1	174
MoveAbs	Approaches an absolute position with all axes	3	175
MoveAbsSingleAxis	Moves to an absolute position with a single axis	3	176
MoveEx	Enables extended travel commands	3	177
MoveRel	Moves to a relative vector with all axes	3	178
MoveRelShort	Moves to a relative vector as a short command	3	178
MoveRelSingleAxis	Moves to a relative vector with a single axis	3	179
StopAxes	Stops all travel movements	3	179
WaitForAxisStop	Waits for the movement stop of specific axes	3	180
GetPosWindowRange	Shows the preset target window	2	181
SetPosWindowRange	Sets the target window	2	181

4.1.10 Table commands

Command	Brief description	X	Page
GetTablePos	Command for reading table positions	2	197
SetTablePos	Command for programming table positions	2	198
MoveTablePosAbs, MoveTablePosAbsW	Absolute positioning to values from table position	2	198
MoveTablePosRel, MoveTablePosRelW	Relative positioning with values from table position	2	199

4.1.11 Joystick and handwheel

Command	Brief description	X	Page
GetHandwheel	Reads the hand wheel status	1	200
SetHandWheelOff	Deactivates the handwheel	1	201
SetHandwheelOn	Activates the handwheel	1	201
GetDigJoySpeed	Reads the speed of the digital joystick	1	202
SetDigJoySpeed	Sets the speed of the digital joystick	1	202
SetDigJoyOff	Deactivates the digital joystick	1	203
GetJoystick	Reads the status of the analogue joystick	3	204
SetJoystickOff	Deactivates the analogue joystick	3	205
SetJoystickOn	Activates the analogue joystick	3	205
GetJoystickFilter	Indicates whether the filtering is active in joystick operation	1	206
SetJoystickFilter	Activates or deactivates the filtering in joystick operation	1	206
GetJoystickWindow	Reads the joystick window	3	207
SetJoystickWindow	Sets the joystick window	3	207
GetJoyChangeAxis	Reads the joystick axis assignment	1	208
JoyChangeAxis	Sets the joystick axis assignment	1	208
GetJoystickAxes	Shows the axes for which the joystick is activated	2	209
SetJoystickAxes	Activates the joystick for the specified axes	2	210
GetJoystickDir	Reads the set joystick direction	3	211
SetJoystickDir	Sets the joystick direction	3	212
GetJoyVel	Inquires the maximum positioning speed in joystick operation	2	213
SetJoyVel	Sets the max. positioning speed in joystick operation	2	213
GetJoyRedcur	Shows if the current reduction in joystick operation is active	2	214
SetJoyRedcur	Activates/Deactivates the current reduction in joystick operation	2	214
GetJoytoAxis	Shows the assigned joystick inputs to the mechanical axes	2	215

4.1.12 Control panel with trackball and joystick keys

Command	Brief description	X	Page
GetBPZ	Reads the control panel status	1	231
SetBPZ	Activates or deactivates the control panel	1	232
GetBPZJoyspeed	Reads the control panel joystick speed	1	232
SetBPZJoyspeed	Sets the control panel joystick speed	1	233
GetBPZTrackballBacklash	Reads the control panel trackball backlash	1	233
SetBPZTrackballBacklash	Sets the control panel trackball backlash	1	234
GetBPZTrackballFactor	Reads the control panel trackball factor	1	234
SetBPZTrackballFactor	Sets the control panel trackball factor	1	235
GetJoyOutPass	Reads the control panel trackball factor	2	235
SetJoyOutPass	Sets the control panel trackball factor	2	235

4.1.13 Digital and analogue inputs and outputs

Command	Brief description	X	Page
GetAnalogInput	Reads the current status of an analogue channel	3	236
GetAnalogInputs2	Reads the current statuses of the analogue channels PT100, MV and V24	1	236
GetPT100Resistance	Outputs the resistance value of a PT100 connected to the analog input	2	237
SetAnalogOutput	Sets and analogue channel	3	237
GetDigitalInputs	Reads the digital inputs (0-15)	3	238
GetDigitalInputsE	Reads the additional digital inputs (16-31)	1	238
SetDigitalOutput	Sets a digital output	3	239
SetDigitalOutputs	Reads the digital outputs (0-15)	3	239
SetDigitalOutputsE	Sets the additional digital outputs (16-31)	1	240
SetDigIO_Distance	Sets the activation of an output dependent on the set distance before/behind of the target position	1	241
SetDigIO_EmergencyStop	Sets the assignment of digital I/Os to Emergency Stop pin	1	242
SetDigIO_Off	Deactivates function of the digital inputs/outputs	1	242
SetDigIO_Polarity	Sets the polarity to the functions of the digital outputs	1	243
GetDigOutLinktoSignal	Shows if there is a signal assigned to the output	2	244
SetDigitalOutLinktoSignal	Assigns a signal to the output	2	245
GetInvertDigOutSignal	Shows if a signal will be inverted	2	246
SetInvertDigOutSignal	Inverts a signal	2	246
GetDigInStatus	Reading Status of digital inputs 0 to 15 (24 V)	2	247

Command	Brief description	X	Page
SetDigInStatus	Automatic transmission of digital inputs 0 to 15 (24 V)	2	248
GetDigInExtStatus	Reading Status of digital inputs 16 to 31 (24 V)	2	249
SetDigInExtStatus, SetDigInExtStatusW	Automatic transmission of digital inputs 16 to 31 (24 V)	2	250
GetTTLDigIn	Reads digital inputs (TTL)	2	251
GetTTLDigOut	Reads digital outputs (TTL)	2	251
SetTTLDigOut	Sets digital outputs (TTL)	2	251
GetMotorBrakeDigOut	Reads motor brake output (in "Digital output" mode)	2	252
SetMotorBrakeDigOut	Sets motor brake output (in "Digital output" mode)	2	252
GetMotorBrakeOut	Reads state of motor brake outputs	2	253
GetTVR	Reading the status of the cycle forward/back operation	2	253
SetTVR	Activates and deactivates cycle forward/back operation	2	253
GetTVRToAxis	Reads Cycle forward/ backward axis assignment	2	254
SetTVRToAxis	Sets Cycle forward/ backward axis assignment	2	254

4.1.14 Cycle forward/back In

Command	Brief description	X	Page
GetFactorTVR	Reads the Forward/ Back cycle factor	3	255
SetFactorTVR	Sets the Forward/ Back cycle factor	3	255
GetTVRMode	Reads the Cycle Forward/ Back mode	1	256
SetTVRMode	Sets the Forward/ Back cycle mode	1	257
SetTVRInPulse	Sets the Forward/Back pulse via the interface	1	258

4.1.15 Cycle forward/back outputs for additional axes

Command	Brief description	X	Page
GetAccelTVRO	Reads all adjusted acceleration values of the TVRO	1	259
SetAccelTVRO	Sets the TVRO acceleration values	1	259
SetAccelSingleAxisTVRO	Sets the individual TVRO acceleration values	1	260
GetVelTVRO	Reads all adjusted speeds of the TVRO	1	260
SetVelTVRO	Sets all speeds of the TVRO	1	261
SetVelSingleAxisTVRO	Sets the speeds of a single TVRO axis	1	261
GetPosTVRO	Shows the TVRO position values dependent on the dimension	1	262
SetPosTVRO	Sets the TVRO position	1	262
GetStatusTVRO	Shows the current status of the axes	1	263

Command	Brief description	X	Page
GetStatusTVROW			
GetTVROutMode	Reads the set TVRO mode	1	264
SetTVROutMode	Sets the TVRO offset	1	264
GetTVROutPitch	Reads the TVRO spindle pitch	1	265
SetTVROutPitch	Sets the TVRO spindle pitch	1	265
GetTVROutResolution	Shows the resolution of the TVRO power amplifier to be controlled	1	266
SetTVROutResolution	Sets the resolution of the TVRO power amplifier to be controlled	1	266
MoveAbsTVRO	Moves to an absolute position with all TVRO axes	1	267
MoveAbsTVROSingleAxis	Moves to an absolute position with a single TVRO axis	1	268
MoveRelTVRO	Moves to a relative vector with all TVRO axes	1	269
MoveRelTVROSingleAxis	Moves to a relative vector with a single TVRO axis	1	270

4.1.16 Encoder settings

Command	Brief description	X	Page
ClearEncoder	Sets the encoder position to zero	1	271
GetEncoder	Reads all encoder positions	1	271
GetEncoderActive	Reads, which encoders are activated after calibration	1	272
SetEncoderActive	Sets the encoders to be activated after calibration	1	273
GetEncoderMask	Reads the encoder statuses	3	274
SetEncoderMask	Activates or deactivates the encoders	1	275
GetEncoderPeriod	Reads the set encoder period lengths	3	276
SetEncoderPeriod	Sets the encoder period lengths	3	276
GetEncoderPosition	Shows the set position source	3	277
SetEncoderPosition	Sets the position source	3	277
GetEncoderRefSignal	Reads whether the encoder reference signal is to be evaluated during calibration	3	278
SetEncoderRefSignal	Sets whether the encoder reference signal is to be evaluated during calibration	3	278
GetEncDir	Shows if the counting direction of the position encoder was changed	2	279
SetEncDir	Sets the counting direction of the position encoder	2	279
GetEncPolePairs	Shows the number of encoder pole pairs per rotation	2	279

Command	Brief description	X	Page
SetEncPolePairs	Sets the number of encoder pole pairs per rotation	2	280
GetEnctoAxis	Shows the assigned encoder inputs to the position encoder interpretations of all axes	2	280
SetEnctoAxis	Assigns encoder inputs to the position encoder interpretations of all axes	2	281
GetEncType	Reads the position encoder types for all axes	2	282
SetEncType	Assigns a position encoder type to all axes	2	283
GetKommMode	Shows the commutation mode in servo operation	2	283
SetKommMode	Sets the commutation mode in servo operation	2	284
GetKommEncDir	Shows the commutation encoder counting direction	2	284
SetKommEncDir	Sets the commutation encoder counting direction	2	285
GetKommEncPolePairs	Shows the commutation encoder signal period	2	285
SetKommEncPolePairs	Sets the commutation encoder signal period	2	285
GetKommEnctoAxis	Shows the assigned encoder inputs to the commutation encoder interpretations of the axes.	2	286
SetKommEnctoAxis	Assigns the encoder inputs to the commutation encoder interpretations of the axes.	2	287
GetKommEncType	Shows the commutation encoder type	2	288
SetKommEncType	Sets the commutation encoder type	2	289

4.1.17 Controller settings

Command	Brief description	X	Page
GetController	Reads the set controller mode	1	290
SetController	Sets the controller mode	1	291
GetControllerCall	Reads the controller call setting	1	291
SetControllerCall	Sets the controller call	1	292
GetControllerFactor	Reads the controller factor setting	1	292
SetControllerFactor	Sets the controller factor	1	293
GetControllerSteps	Reads the controller steps	1	293
SetControllerSteps	Sets the controller steps	1	294
GetControllerTimeout	Shows the controller monitoring timeout setting	1	294
SetControllerTimeout	Sets the controller monitoring timeout	1	295
GetControllerTWDelay	Reads the controller delay	1	295
SetControllerTWDelay	Sets the controller delay	1	296
GetCtrFastMove	Reads setting of the Fast Move Function	1	296

Command	Brief description	X	Page
SetCtrFastMoveOff	Deactivates the Fast Move Function	1	297
SetCtrFastMoveOn	Sets the Fast Move function	1	297
GetCtrFastMoveCounter	Reads the number of Fast Move functions carried out	1	298
ClearCtrFastMoveCounter	Sets the number of Fast Move functions carried out to 0	1	298
GetTargetWindow	Shows the controller target window	3	299
SetTargetWindow	Sets the controller target window	3	299
GetDeviationRange	Shows the range of contouring error	2	300
SetDeviationRange	Sets the range of contouring error	2	300
GetDeviationTime	Shows the time for contouring error check	2	300
SetDeviationTime	Sets the time for the contouring error check	2	301
GetDeviationCheck	Shows if the contouring error check is active	2	301
SetDeviationCheck	Activates/Deactivates the contouring error check	2	301
GetDeviationValue	Reads the deviation value	2	302
GetPoscon	Shows the position controller setting	2	302
SetPoscon	Activates/Deactivates the position controller	2	302
GetPosConKP	Reading KP band of position controller	2	303
SetPosConKP	Setting KP band of position controller	2	303
GetPosConAdaptiveKP	Reading KP band of position controller (adaptive)	2	303
SetPosConAdaptiveKP	Setting KP band of position controller (adaptive)	2	304
GetPosConKI	Reading KI band of position controller	2	304
SetPosConKI	Setting KI band of position controller	2	304
GetPosConAdaptiveKI	Reading KI band of position controller (adaptive)	2	305
SetPosConAdaptiveKI	Setting KI band of position controller (adaptive)	2	305
GetPosConOutPass	Reading Time constant of position controller output filter	2	305
SetPosConOutPass	Setting Time constant of position controller output filter	2	306
GetPosConAdaptiveOutPass	Reading Time constant of position controller output filter (adaptive)	2	306
SetPosConAdaptiveOutPass	Setting Time constant of position controller output filter (adaptive)	2	306
GetPosConNominalSpeed	Reading Rated velocity	2	307
SetPosConNominalSpeed	Setting Rated velocity	2	307
GetPosConAdaptiveSpeed	Reading Rated velocity (adaptive)	2	308
SetPosConAdaptiveSpeed	Setting Rated velocity (adaptive)	2	308
GetPosConEnable	Reading Axis enable for position controller	2	309

Command	Brief description	X	Page
SetPosConEnable	Setting Axis enable for position controller	2	309
GetSpeedConKP	Reading KP band of speed controller	2	310
SetSpeedConKP	Setting KP band of speed controller	2	310
GetSpeedConKI	Reading KI band of speed controller	2	310
SetSpeedConKI	Setting KI band of speed controller	2	311
GetSpeedConKD	Reading KD band of speed controller	2	311
SetSpeedConKD	Setting KD band of speed controller	2	311
GetSpeedConOutPass	Reading Time constant of speed controller output filter	2	312
SetSpeedConOutPass	Setting Time constant of speed controller output filter	2	312
GetActSpeedFilterConst	Reading Time constanct of actual speed value filter	2	312
SetActSpeedFilterConst	Setting Time constanct of actual speed value filter	2	313
GetSpeedFeedForward	Reading Speed or velocity feed forward	2	313
SetSpeedFeedForward	Setting Speed or velocity feed forward	2	313
GetActAccelFilterConst	Reading Time constant of actual acceleration value filter	2	314
SetActAccelFilterConst	Setting Time constant of actual acceleration value filter	2	314
GetAccelFeedForward	Reading Acceleration feed forward	2	314
SetAccelFeedForward	Setting Acceleration feed forward	2	315
GetAccelFeedForwardOut-Pass	Reading Time constant of acceleration feed forward filter	2	315
SetAccelFeedForwardOut-Pass	Setting Time constant of acceleration feed forward filter	2	315
GetSpeedConTN	Read Reset time TN of speed controller	2	316

4.1.18 Deviation Criterion

Command	Brief description	X	Page
GetDevCritMode	Reads the time period for monitoring the deviation criterion	2	316
SetDevCritMode	Sets the time period for monitoring the deviation criterion	2	317
GetDevCritType	Reads the type for the deviation criterion	2	317
SetDevCritType	Sets the type for the deviation criterion	2	318
GetDevCritEnable	Queries if the deviation criterion measurement is active	2	318
SetDevCritEnable	Activates/ Deactivates the deviation criterion measurement	2	319
GetDevCritValue	Reads the deviation criterion	2	319

4.1.19 Trigger output

Command	Brief description	X	Page
GetTrigger	Reads the trigger setting	3	320
SetTrigger	Activates or deactivates the Trigger	3	321
GetTrigCount	Reads the trigger counter	3	321
SetTrigCount	Sets the trigger counter	3	322
GetTriggerPar	Reads the trigger parameters	3	322
SetTriggerPar	Sets the trigger parameters	3	323
GetTriggerDim	Shows the unit for parametertransfer	2	323
SetTriggerDim	Sets the unit for parametertransfer	2	324
GetTriggerSource	Shows the trigger source	2	324
SetTriggerSource	Sets the trigger source	2	325
GetTriggerHysterese	Shows the trigger hysteresis	2	325
SetTriggerHysterese	Sets the trigger hysteresis	2	326
GetTriggerPLength	Shows the trigger signal period length	2	326
SetTriggerPLength	Sets the trigger signal period length	2	327
GetTriggerPulsCount	Shows the number of pulses to be output in burst mode	2	327
SetTriggerPulsCount	Sets the number of pulses in the burst mode	2	327
GetTrigger_Two	Reads the trigger setting for the second trigger	2	328
SetTrigger_Two	Activates or deactivates the second trigger	2	328
GetTrigger_TwoCount	Reads the trigger counter for the second trigger	2	329
SetTrigger_TwoCount	Sets the trigger counter for the second trigger	2	329
GetTrigger_TwoPar	Reads the trigger parameters for the second trigger	2	330
SetTrigger_TwoPar	Sets the trigger parameters for the second trigger	2	331
GetTrigger_TwoDim	Shows the unit for parametertransfer for the second trigger	2	331
SetTrigger_TwoDim	Sets the unit for parameter transfer for the second trigger	2	332
GetTrigger_TwoSource	Shows the trigger source for the second trigger	2	332
SetTrigger_TwoSource	Sets the trigger source for the second trigger	2	333
GetTrigger_TwoHyste-rese	Shows the trigger hysteresis for the second trigger	2	333
SetTrigger_TwoHysterese	Sets the trigger hysteresis for the second trigger	2	334
GetTrigger_TwoPLength	Shows the trigger signal period length for the second trigger	2	334
SetTrigger_TwoPLength	Sets the trigger signal period length for the second trigger	2	335
GetTrigger_TwoPuls-Count	Shows the number of pulses to be output in burst mode for the second trigger	2	335

Command	Brief description	X	Page
SetTrigger_TwoPuls-Count	Sets the number of pulses in the burst mode Sets the number of pulses in the burst mode for the second trigger	2	335

4.1.20 Snapshot input

Command	Brief description	X	Page
GetSnapshot	Reads the snapshot setting	3	336
SetSnapshot	Activates or deactivates the Snpashot	3	336
GetSnapshotFilter	Reads the input filter	3	337
SetSnapshotFilter	Sets the input filter	3	337
GetSnapshotPar	Reads the snapshot parameters	3	338
SetSnapshotPar	Sets the snapshot parameters	3	339
GetSnapshotCount	Reads the snapshot counter	3	339
GetSnapshotPos	Reads the snapshot position	3	340
GetSnapshotPosArray	Reads the snapshot position array	3	340
GetSnapshotSource	Shows the position value source	2	341
SetSnapshotSource	Sets the position value source	2	341

4.2 Detailed functional description

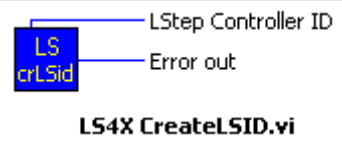
The detailed functional descriptions below contain a descriptive text for the function, a description of the function parameters, an example call of the function as well as the prototypes for the programming languages Delphi and C++. In addition, the "Other" column includes notes as to which controller is supported by the function, which command is transmitted to the controller and whether it is necessary to activate the command.

The "Activation" field currently only applies to the LSTEP express controller series.

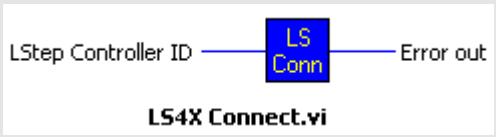
A section of the table for the area "Other" is listed below for better clarity:

Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	(Compatible with X, see Section 4.1)	(Used command from the controller command set)	(Activation possible via the indicated "SendSting" commands)

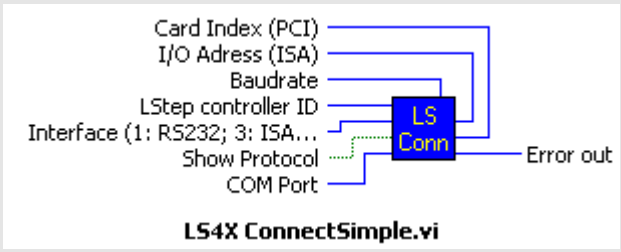
4.2.1 API-Configuration/Interface configuration

LSX_CreateLSID			
Description:	Creates an LSTEP ID number. This is used as an additional parameter in the LSTEP-API commands in order to select the LSTEP to which the command shall refer out of several connected LSTEP controllers.		
Delphi:	function LSX_CreateLSID(var LSID: Integer): Integer;		
C++:	-		
LabView:			
Parameter:	LSID	Contains a new LSTEPID number after calling CreateLSID; this number may then be used for connect, moving and other commands	
Example:	var LStep1: Integer; ... LSX_CreateLSID(&LStep1);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

<table>
<tr><th colspan="4">LSX_FreeLSID</th></tr>
<tr><td>Description:</td><td colspan="3">Releases a created LSTEP ID number. This is used as an additional parameter in the LSTEP-API commands in order to select the LSTEP to which the command shall refer out of several connected LSTEP controllers. FreeLSID should only be called after Disconnect.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_FreeLSID(LSID: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">-</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>LSID</td><td colspan="2">LSTEP ID number to be released. This may no longer be used after FreeLSID.</td></tr>
<tr><td>Example:</td><td colspan="3">var LStep1: Integer;
...
LSX_CreateLSID(&LStep1);
LSX_ConnectSimple(LStep1, ...);
...
LSX_Disconnect(LStep1);
LSX_FreeLSID(LStep1);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>-</td><td>-</td></tr>
</table>

LSX_Connect			
Description:	This function establishes a connection to an LSTEP controller. For this purpose, the interface parameters loaded from the INI file via LSX_LoadConfig are used. (One of the functions LSX_Connect, LSX_ConnectSimple or LSX_ConnectEx must be called for initialising of the interface so that the communication with the LSTEP is possible.)		
Delphi:	function LSX_Connect(LSID: Integer): Integer;		
C++:	int Connect();		
LabView:			
Parameter:	-		
Example:	LSX.LoadConfig("C:\LStepTest\LStep.INI"); LSX.Connect();		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

LSX_ConnectEx, LSX_ConnectExW			
Description:	This function establishes a connection to an LSTEP controller. A pointer is transferred to a data structure containing the interface parameters. This record also returns information on the identified controller (version number...). (One of the functions LSX_Connect, LSX_ConnectSimple or LSX_ConnectEx must be called for initialising of the interface so that the communication with the LSTEP is possible.)		
Delphi:	function LSX_ConnectEx(LSID: Integer; var AControlInitPar: TLS_ControlInitPar): Integer; function LSX_ConnectExW(LSID: Integer; var AControlInitPar: TLS_ControlInitParW): Integer;		
C++:	int ConnectEx (TLS_ControlInitPar *pAControlInitPar); int ConnectExW (TLS_ControlInitParW *pAControlInitPar);		
LabView:	-		
Parameter:	AControlInitPar	Pointer on a record of type TLS_ControlInitPar or TLS_ControlInitParW	
Example:	LSX.ConnectEx(&ControlInitPar1);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

<table>
<tr><th colspan="4">LSX_ConnectSimple, LSX_ConnectSimpleW</th></tr>
<tr><td>Description:</td><td colspan="3">This function establishes a connection to an LSTEP controller. The settings of the interface are transferred as parameters.
(One of the functions LSX_Connect, LSX_ConnectSimple or LSX_ConnectEx must be called for initialising of the interface so that the communication with the LSTEP is possible.)</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_ConnectSimple(LSID: Integer; AnInterfaceType: Integer; AComName: PAnsiChar; ABaudRate: Integer; AShowProt: LongBool): Integer;
function LSX_ConnectSimpleW(LSID: Integer; AnInterfaceType: Integer; AComName: PWideChar; ABaudRate: Integer; AShowProt: LongBool): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int ConnectSimple (int lAnInterfaceType, char *pcAComName, int lABR, BOOL AShowProt);
int ConnectSimpleW (int lAnInterfaceType, TCHAR *pcAComName, int lABR, BOOL AShowProt);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="4">Parameter:</td><td>AnInterfaceType</td><td colspan="2">Interface type
1 = RS232
2 = ArcNet
3 = DPRAM / ISA-Bus
4 = DPRAM / PCI-Bus
5 = RS232 with RTS/CTS and extended log
11= RS232 with RTS/CTS</td></tr>
<tr><td>AComName</td><td colspan="2">Name of the COM interface, e.g. 'COM2'; to be set to ZERO with ArcNet or DPRAM</td></tr>
<tr><td>ABR</td><td colspan="2">The meaning is independent of the interface type
RS232 = baud rate, e.g. 9600
ArcNet = 0 for coax, 1 for twisted pair
DPRAM/ISA-Bus = Basic I/O address, e.g. 0x0340
DPRAM/PCI-Bus = 0 for first card, 1 for second card</td></tr>
<tr><td>AShowProt</td><td colspan="2">Show interface log
True = indicate
False = Hide</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.ConnectSimple(1, “COM2“, 9600, true); // RS232, 9600 Baud
LS_ConnectSimple(4, nil, 0, true); //LSTEP PCI card 0;</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>-</td><td>-</td></tr>
</table>

LSX_Disconnect			
Description:	Disconnects the connection to the LSTEP controller.After calling this function, commands can no longer be transmitted to the LSTEP. This function should be called shortly before the program is terminated.		
Delphi:	function LSX_Disconnect(LSID: Integer): Integer;		
C++:	int Disconnect ();		
LabView:			
Parameter:	-		
Example:	LSX.Disconnect();		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

<table>
<tr><th colspan="4">LSX_SetExtValue</th></tr>
<tr><td>Description:</td><td colspan="3">Activates API extension; these are partially experimental modes for debugging purposes.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_SetExtValue(LSID: Integer; AName, AValue: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int SetExtValue (int lAName, int lAValue);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="6">Parameter:</td><td>AName</td><td colspan="2">Number of extended function</td></tr>
<tr><td>AValue</td><td colspan="2">Parameter</td></tr>
<tr><td>AName=2
(IFSleepTime)</td><td colspan="2">Sets the polling interval for the DPRAM of the LSTEPPCI.
AValue: Time interval in [ms], default is 10</td></tr>
<tr><td>AName=3
(ProtMoveOnly)</td><td colspan="2">Activates the filter for the log file so that only moves & errors are recorded
AValue=1: Filter On
AValue=0: Filter Off</td></tr>
<tr><td>AName=4
(Max_LogLn)</td><td colspan="2">Limits the length of the log file, the older log file will be re-named in .old
AValue=maximum number of lines</td></tr>
<tr><td>AName=5
(ThreadPriority)</td><td colspan="2">Changes the priority of threads of the LSTEP-API. After Connect, the threads are always set to normal priority; they may be changed subsequently using SetExtValue(5, ...).
AValue:the Windows API constant for thread priority such as THREAD_PRIORITY_ABOVE_NORMAL</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.SetExtValue(3, 1); // Filter for move commands On
LSX.SetExtValue(4, 10000); // maximum length of log file = 10000 lines
LSX.SetExtValue(5, THREAD_PRIORITY_HIGHEST);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>-</td><td>-</td></tr>
</table>

<table>
<tr><th colspan="5">LSX_SetProcessMessagesProc</th></tr>
<tr><td>Description:</td><td colspan="4">Enables the replacement of the internal message dispatching procedure of the LSTEP-API.
The LSTEP-API processes messages while waiting for acknowledgements of the LStep in Main-Thread. If you want to turn off message dispatching or replace it by your own code, you can use SetProcessMessagesProc to set a callback procedure.</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_SetProcessMessagesProc(LSID: Integer; Proc: Pointer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int SetProcessMessagesProc(void* pProc);</td></tr>
<tr><td>LabView:</td><td colspan="4"> 
</td></tr>
<tr><td>Parameter:</td><td>pProc</td><td colspan="3">Pointer to substitute function
This must be a pointer to a stdcall procedure without a parameter: void MyProcessMessages ()</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.SetProcessMessagesProc(&MyProcessMessages);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>-</td><td colspan="2">-</td></tr>
</table>

<table>
<tr><th colspan="4">LSX_SetOsziCallBackFct</th></tr>
<tr><td>Description:</td><td colspan="3">Transfers the address of a CallBack function to the API to be called if an asynchronous event occurs. The set callback function is global and does not differentiate between the LSTEPs. If a controller-specific CallBack function is to be called, the function LSX_SetExtCallBackFct has to be used. As long as no controller-related CallBack function has been assigned via the function LSX_SetExtCallBackFct the function assigned by LSX_SetOsziCallBackFct is used.
This functionality is dependent on the interface type and the controller. A list of the interface types and controllers supported is included in “5.4 Supported controller or interface types”.
The description for using the CallBack function of the LSTEP-API as well as its structure is included in “5 CallBack functions of the LSTEP-API”.</td></tr>
<tr><td>Delphi:</td><td colspan="3">LSX_SetOsziCallBackFct(LSID: Integer; Fct: Pointer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int SetOsziCallBackFct (void* pFct)</td></tr>
<tr><td>LabView:</td><td colspan="3">-</td></tr>
<tr><td>Parameter:</td><td>pFct</td><td colspan="2">Pointer to CallBack function. For function declaration, please refer to “5 CallBack functions of the LSTEP-API”.</td></tr>
<tr><td>Example:</td><td colspan="3">void CALLBACK OsziCallBackFct(char *pcData, int lMaxLen, int lChannelID)
{
…
}
…
LSX.SetOsziCallBackFct(OsziCallBackFct);
//activate extended log:
LSX.SendString("!errorchannel 2\r", 0, 0, false, 1000);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>2</td><td>-</td><td>-</td></tr>
</table>

LSX_SetExtCallBackFct			
Description:	Transfers the address of a CallBack function to the API to be called if an asynchronous event occurs. The set CallBack function refers to the LSTEP entities and allows for transferring an object, e.g. for controller identification. As long as no controller-related CallBack function has been assigned via the function LSX_SetExtCallBackFct the function assigned by LSX_SetOsziCallBackFct is used. This functionality is dependent on the interface type and the controller. A list of the interface types and controllers supported is included in "5.4 Supported controller or interface types". The description for using the CallBack function of the LSTEP-API as well as its structure is included in "5 CallBack functions of the LSTEP-API".		
Delphi:	LSX_SetExtCallBackFct(LSID: Integer; Fct: Pointer; pprivate: Pointer): Integer;		
C++:	int SetExtCallBackFct (void* pFct, void* ppprivate)		
LabView:	-		
Parameter:	pFct	Pointer to CallBack function. For function declaration, please refer to "5 CallBack functions of the LSTEP-API".	
	pprivate	Pointer to object which is transferred when the CallBack function is called.	

Example:	int CALLBACK LSTEPExtCallBackFct(char *pcData, int lMaxLen, int lChannelID, void* pObject)
	{
	…
	}
	…
	LSX.SetExtCallBackFct(LSTEPExtCallBackFct);
	//activate extended log:
	LSX.SendString("!errorchannel 2\r", 0, 0, false, 1000);		

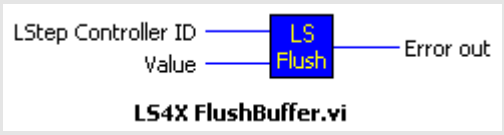
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	-	-

LSX_SetLanguage, LSX_SetLanguageW			
Description:	Set language for LSTEP-API (log/messages)		
Delphi:	function LSX_SetLanguage(LSID: Integer; PLN: PAnsiChar): Integer; function LSX_SetLanguageW(LSID: Integer; PLN: PWideChar): Integer;		
C++:	int SetLanguage (char *pcPLN); int SetLanguageW (TCHAR *pcPLN);		
LabView:			
Parameter:	PLN	Language (abbrev., e.g. "DEU" [German] or "ENG" [English] The appropriate ANSI or UNICODEtext file (LSTEP4deu.txt or LSTEP4eng.txt) must be included in the program directory.	
Example:	LSX.SetLanguage('ENG');		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

LSX_TranslateErrMsg			
Description:	Translates error message transferred by a CallBack function.		
Delphi:	LSX_TranslateErrMsg(LSID: Integer; MsgIn: PWideChar; MsgOut: PWideChar; MaxLen: Integer): Integer;		
C++:	int TranslateErrMsg (TCHAR *pcMsgIn, TCHAR *pcMsgOut, int lMaxLen)		
LabView:	-		
Parameter:	MsgIn	Error message transferred to the CallBack function, converted into UNICODE, zero-terminated.	
	pcMsgOut	Buffer containing the error message translation.	
	MaxLen	Maximum number of characters, which may be copied into the buffer.	

Example:	TCHAR InputString[255];
	TCHAR TranslatedString [255];
	
	// translate ASCII string to UNICODE
	wcscpy_s(InputString, CString(pcData, lMaxLen));
	LSX.TranslateErrMsg(InputString, TranslatedString, 255);
	
	… // process TranslatedString		

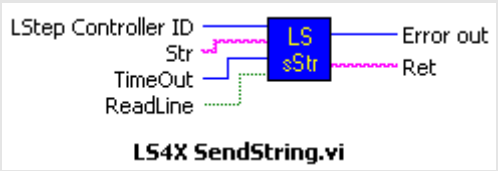
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

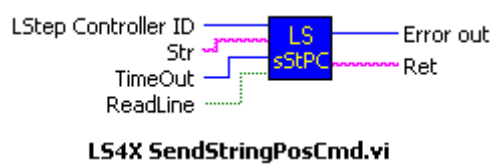
LSX_FlushBuffer			
Description:	Deletes the communication input buffer at RS-232 or PCI interface. This is useful in error situations for eliminating acknowledgements no longer needed from the input buffer.		
Delphi:	function LSX_FlushBuffer(LSID: Integer; AValue: Integer): Integer;		
C++:	int FlushBuffer (int lAValue);		
LabView:			
Parameter:	AValue	Currently not used, can be set to 0	
Example:	LSX.FlushBuffer(0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

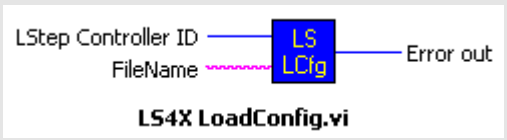
LSX_SetAbortFlag			
Description:	Set flag so that communication with LSTEP is disrupted. A function that is still waiting for a feedback from the controller (e.g. movement commands) when LSX_SetAbortFlag is called, then returns an error message. This function is especially useful in programs with message handling routines or several threads, e.g. if a movement is to be aborted quickly.		
Delphi:	function LSX_SetAbortFlag(LSID: Integer): Integer;		
C++:	int SetAbortFlag ();		
LabView:			
Parameter:	-		
Example:	LSX.SetAbortFlag(); LSX.StopAxes(); (terminate communication with the LSTEP and send the command to stop all axes)		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

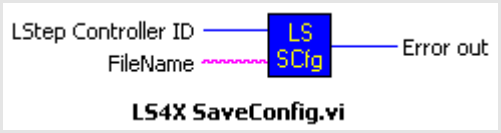
LSX_EnableCommandRetry			
Description:	This function is used to activate/deactivate the repeated sending of commands in case errors (activated by default).		
Delphi:	function LSX_EnableCommandRetry(LSID: Integer; AValue: LongBool): Integer;		
C++:	int EnableCommandRetry (BOOL bAValue);		
LabView:			
Parameter:	AValue	Activate or deactivate repeated sending True = activate repeated sending in case of errors False = deactivate repeated sending in case of errors	
Example:	LSX.EnableCommandRetry(false) ;		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

LSX_SetCommandTimeout			
Description:	Sets timeouts for waiting foracknowledgement with regard to general API calls, positioning and calibration. The default value for the timeout is 1000 ms.		
Delphi:	LSX_SetCommandTimeout(LSID: Integer; AtoRead, AtoMove, AtoCalibrate: Integer): Integer;		
C++:	int SetCommandTimeout (int lAtoRead, int lAtoMove, int lAtoCalibrate) ;		
LabView:			
Parameter:	AtoRead	Timeout for waiting for general acknowledgement of an API call in ms	
	AtoMove	Timeout for positioning in ms	
	AtoCalibrate	Timeout for calibration in ms	
Example:	LSX. SetCommandTimeout (int lAtoRead, int lAtoMove, int lAtoCalibrate) ;		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

<table>
<tr><th colspan="4">LSX_SendString, LSX_SendStringW</th></tr>
<tr><td>Description:</td><td colspan="3">Sends a string to an LSTEP controller. This function is used to transmit commands to a controller if not API function exists for them. The command structure and the terminating characters are to be observed. Please also refer to the relevant controller documentation.
Beware: The command “!configmaxaxis” is notto be send via this function. Use the corresponding API command “LSX_ConfigMaxAxis”. Refer to the Chapter “Axis and motor configuration” for more information.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_SendString(LSID: Integer; Str, Ret: PAnsiChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer;
function LSX_SendString(LSID: Integer; Str, Ret: PWideChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int SendString (char *pcStr,char *pcRet,int lMaxLen,BOOL ReadLine,int lTimeOut);
int SendStringW (TCHAR *pcStr,TCHAR *pcRet,int lMaxLen,BOOL ReadLine,int lTimeOut);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="5">Parameter:</td><td>Str</td><td colspan="2">Zero-terminated string that is to be sent to the controller.</td></tr>
<tr><td>Ret</td><td colspan="2">Buffer containing the acknowledgement of the LSTEP, if ReadLine = true</td></tr>
<tr><td>MaxLen</td><td colspan="2">Maximum number of characters that can be copied into the buffer for acknowledgement.</td></tr>
<tr><td>ReadLine</td><td colspan="2">Waiting for acknowledgement from LSTEP controller.
True = Acknowledgement expected
False = No acknowledgement</td></tr>
<tr><td>TimeOut</td><td colspan="2">Maximum waiting time for acknowledgement in ms.</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.SendString(“?ver\r“, pcLStepVer, 256, true, 1000);
// Read version number, timeout 1s</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>-</td><td>-</td></tr>
</table>

LSX_SendStringPosCmd, LSX_SendStringPosCmdW			
Description:	Sends a string as a moving command to the controller, which may await the axis, mask (see LSX_GetStatusAxis, LSX_GetStatusAxisW) as acknowledgement from the controller.		
Delphi:	function LSX_SendStringPosCmd(LSID: Integer; Str, Ret: PAnsiChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer; function LSX_SendStringPosCmdW(LSID: Integer; Str, Ret: PWideChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer;		
C++:	int SendStringPosCmd (char *pcStr, char *pcRet, int lMaxLen, BOOL bReadLine, int lTimeOut); int SendStringPosCmdW(TCHAR*pcStrTCHAR *pcRet, int lMaxLen, BOOL bReadLine, int lTimeOut);		
LabView:			
Parameter:	Str	Zero-terminated string that is to be sent to the controller.	
	Ret	Buffer containing the acknowledgement of the LSTEP, if ReadLine = True	
	MaxLen	Maximum number of characters, which may be copied into the buffer.	
	ReadLine	Read acknowledgement of the LSTEP:	
	TimeOut	Maximum waiting time for acknowledgement in [ms].	
Example:	LSX.SendStringPosCmd("!moa 1 2\r", pcLStepVer, 256, true, 100000);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

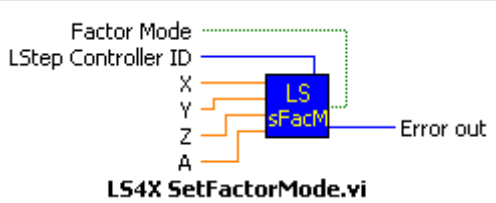
<table>
<tr><th colspan="4">LSX_LoadConfig, LSX_LoadConfigW</th></tr>
<tr><td>Description:</td><td colspan="3">Loads LSTEP configuration (interface, axis settings, controllers) from an INI file. The loaded configuration is used in the functions LSX_Connect and LSX_SetControlPars.
The format of the INI file for LSTEP controllers is compatible with the WIN-Commander INI file, i.e. the settings can be taken over from the WIN-Commander (Wincom4.ini). This does not apply to WIN-Commander 5 and the controllers of the LSTEP express series
Attention!
For LSTEP-PCI express or LSTEP express, you can save the settings with WIN-Commander 5 in the Controller/Export configuration menu, and can load them with LoadConfig. LSX_SetControlPars is used to transmit this configuration to the controller.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_LoadConfig(LSID: Integer; FileName: PAnsiChar): Integer;
function LSX_LoadConfigW(LSID: Integer; FileName: PWideChar): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int LoadConfig (char *pcFileName);
int LoadConfigW (TCHAR *pcFileName);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>FileName</td><td colspan="2">File name of INI file as zero-terminated string</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.LoadConfig(“C:\LStepTest\LStep.INI“);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>-</td><td>-</td></tr>
</table>

LSX_SaveConfig, LSX_SaveConfigW			
Description:	Saves LSTEP configuration (interface, axis settings, controllers) in an INI file. The format of the INI file is compatible with the WIN-Commander 4 INI file. (only for LSTEP express)		
Delphi:	function LSX_SaveConfig(LSID: Integer; FileName: PAnsiChar): Integer; function LSX_SaveConfig(LSID: Integer; FileName: PWideChar): Integer;		
C++:	int SaveConfig (char *pcFileName); int SaveConfigW (TCHAR *pcFileName);		
LabView:			
Parameter:	FileName	File name of INI file as zero-terminated string	
Example:	LSX.SaveConfig("C:\LStepTest\LStep.INI");		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	-	-

<table>
<tr><th colspan="4">LSX_SetFactorMode</th></tr>
<tr><td>Description:</td><td colspan="3">Position value conversion for 'odd' spindle pitches

SetPitch can only be called with the actual, physical spindle pitch after SetFactorMode.

All moving commands use a factor for conversion after calling SetFactorMode and SetPitch, so that the LSTEP is positioned correctly. The resulting vector is as follows:

Sent position vector = sent position vector * spindle pitch LSTEP / physical spindle pitch</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_SetFactorMode(LSID: Integer; AFactorMode: LongBool; X, Y, Z, A: Double): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int SetFactorMode (BOOL bAFactorMode, double dX, double dY, double dZ, double dA);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="2">Parameter:</td><td>AFactorMode</td><td colspan="2">This parameter activates an API-internal conversion of the position values/spindle pitch in order to avoid rounding errors with 'odd' spindle pitches.</td></tr>
<tr><td>X, Y, Z, A</td><td colspan="2">Spindle pitch values transferred to the LSTEP (if possible values such as 1.0 or 4.0 so that a microstep will correspond to a non-periodic decimal fraction)</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.SetFactorMode(true, 1, 1, 1, 0);
LSX.SetPitch(1.234, 1.234, 2.345, 0);
LSX.MoveAbs(1.234, 2.468, 2.345, 0, true);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>1</td><td>-</td><td>-</td></tr>
</table>

LSX_Initialize, LSX_InitializeW			
Description	Function for setting the path in which the settings for the log window are saved. This path also includes the log files, if no path was set for them using SetWriteLogTextFN.		
Delphi	function LSX_Initialize(LSID: Integer; Workpath: PAnsiChar): Integer; function LSX_InitializeW(LSID: Integer; Workpath: PWideChar): Integer;		
C++	int Initialize(char *pcWorkpath); int InitializeW(TCHAR *pcWorkpath);		
LabView:	-		
Parameter	Work path – zero-terminated string		
Example	LSX.Initialize("C:\Users\All Users\MyProg\LS1");		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

LSX_SetShowCmdList			
Description:	Display or hide LSTEP-API command list		
Delphi:	function LSX_SetShowCmdList(LSID: Integer; ShowCmdList: LongBool): Integer;		
C++:	int SetShowCmdList (BOOL bShowCmdList);		
LabView:	-		
Parameter:	ShowProt	Indicates whether or not the window "LSTEP-API command list" is to be shown	
Example:	LSX.SetShowCmdList(true);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

LSX_SetShowProt			
Description:	Display or hide interface protocol		
Delphi:	function LSX_SetShowProt(LSID: Integer; ShowProt: LongBool): Integer;		
C++:	int SetShowProt (BOOL ShowProt);		
LabView:			
Parameter:	ShowProt	Specifies whether or not the window "Interface Protocol" is to be shown	
Example:	LSX.SetShowProt(true);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

LSX_EnableGlobalLogging			
Description:	Deactivate every logging-functionality of the API		
Delphi:	function LSX_EnableGlobalLogging(LSID: Integer; Enable: LongBool): Integer;		
C++:	int EnableGlobalLogging (BOOL Enable);		
LabView:	Not supported		
Parameter:	Enable	Activate every logging-functionality of the API	
Example:	LSX.EnableGlobalLogging(false); //deactivating the logging completely		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

LSX_EnableGuiLogging			
Description:	Deactivate only the logging into the protocol window oft he API. Deactivating logging, reduces the generated CPU-usage of the API drastically.		
Delphi:	function LSX_EnableGuiLogging(LSID: Integer; Enable: LongBool): Integer;		
C++:	int EnableGuiLogging (BOOL Enable);		
LabView:	Not supported		
Parameter:	Enable	Activate logging into the protocol window.	
Example:	LSX.EnableGuiLogging(false); //deactivate logging to the protocol window		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

LSX_SetWriteLogText			
Description:	Writing of log file LSTEP4.log On/Off (writing is deactivated in LSTEP4.log by default)		
Delphi:	function LSX_SetWriteLogText(LSID: Integer; AWriteLogText: LongBool): Integer;		
C++:	int SetWriteLogText (BOOL AWriteLogText);		
LabView:			
Parameter:	-		
Example:	LSX.SetWriteLogText (true);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

LSX_SetWriteLogTextFN, LSX_SetWriteLogTextFNW			
Description:	Activation/deactivation of writing the interface log in a specific file (writing is deactivated by default)		
Delphi:	function LSX_SetWriteLogTextFN(LSID: Integer; AWriteLogText: LongBool; ALogFN: PAnsiChar): Integer; function LSX_SetWriteLogTextFNW(LSID: Integer; AWriteLogText: LongBool; ALogFN: PWideChar): Integer;		
C++:	int SetWriteLogTextFN (BOOL bAWriteLogText, char *pcALogFN); int SetWriteLogTextFNW (BOOL bAWriteLogText, TCHAR *pcALogFN);		
LabView:	Not supported		
Parameter:	AWriteLogText	If True, then write log file	
	ALogFN	File name of log file	
Example:	LSX.SetWriteLogTextFN(true, "C:\Temp\prot.txt");		
Other:	Compatibility	"SendString" command	Aktivierung (LSTEP express Serie)
	3	-	

LSX_GetEqepConfig			
Description:	Shows the function of pins 20 to 25 of the 50-pole multifunction port (MFP).		
Delphi:	function LSX_GetEqepConfig (LSID: Integer; var GetWert: Integer): Integer;		
C++:	int GetEqepConfig (int *plGetWert);		
LabView:	Not supported		
Parameter:	Value:		
	0 = TTL inputs		
	1 = Inching operation		
	2 = Trackball operation		
	3 = Handwheel operation (intended/ not yet implemented)		
	4 = TVR inputs		
	5 = QEP encoder systems		
	6 = Axis-enable in joystick-operation		
Example:	LSX.GetEqepConfig(&GetWert);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?eqepconfig	

LSX_SetEqepConfig			
Description:	Configures the function of pins 20 to 25 of the 50-pole multifunction port (MFP).		
Delphi:	function LSX_SetEqepConfig(LSID: Integer; Wert: Integer): Integer;		
C++:	int GetEqepConfig (int *plWert);		
LabView:	Not supported		
Parameter:	Value:		
	0 = TTL inputs		
	1 = Inching operation		
	2 = Trackball operation		
	3 = Handwheel operation (intended/ not yet implemented)		
	4 = TVR inputs		
	5 = QEP encoder systems		
	6 = Axis-enable in joystick-operation		
Example:	LSX.SetEqepConfig (0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!eqepconfig	!validconfig

LSX_GetTTLOutConfig			
Description:	Shows the function of pins 16 to 19 of the 50-pole multifunction port (MFP).		
Delphi:	function LSX_GetTTLOutConfig(LSID: Integer; TTLOut: Integer): Integer;		
C++:	int GetTTLOutConfig (int *plTTLOut);		
LabView:	Not supported		
Parameter:	TTLOut	0 = TTL output	
		1 = TVR Out operation	
Example:	LSX.GetTTLOutConfig(&TTLOut);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?ttloutconfig	-

LSX_SetTTLOutConfig			
Description:	Configures the function of pins 20 to 25 of the 50-pole multifunction port (MFP).		
Delphi:	function LSX_SetTTLOutConfig(LSID: Integer; TTLOut: Integer): Integer;		
C++:	int SetTTLOutConfig (int lTTLOut);		
LabView:	Not supported		
Parameter:	TTLOut	0 = TTL output	
		1 = TVR Out operation	
Example:	LSX.SetTTLOutConfig(1); //Pins in TVR out operation		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!ttloutconfig	ValidConfig

LSX_GetStopMode			
Description:	Shows the mode of the active stop input.		
Delphi:	function LSX_GetStopMode (LSID: Integer; var Mode: Integer): Integer;		
C++:	int GetStopMode (int *plMode);		
LabView:	Not supported		
Parameter:	Mode	0 = Stop of movments will be activated	
		1 = Stop movment and deactivating of power amplifiers will be activated	
Example:	LSX.GetStopMode(&Mode);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?stopmode	-

LSX_SetStopMode			
Description:	Sets the mode of the active stop input		
Delphi:	function LSX_SetStopMode (LSID: Integer;Mode: Integer): Integer;		
C++:	int SetStopMode (int lMode);		
LabView:	Not supported		
Parameter:	Mode	0 = Stop of movments will be activated	
		1 = Stop movment and deactivating of power amplifiers will be activated	
Example:	LSX.SetStopMode(1);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!stopmode	ValidConfig

LSX_GetConfigurated			
Description:	Shows if the 'Configured' ID is set. Parameter set of individual axes is declared valid in the controller. Axes without 'Configured' ID cannot be operated.		
Delphi:	function LSX_GetConfigurated (LSID: Integer; var X,Y,Z,A: LongBool): Integer;		
C++:	int GetConfigurated (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A		
Example:	LSX.GetConfigurated(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?configurated	-

LSX_SetConfigurated			
Description:	Sets the 'Configured' ID. Parameter set of individual axes is declared valid in the controller. Axes without 'Configured' ID cannot be operated.		
Delphi:	function LSX_SetConfigurated (LSID: Integer; X,Y,Z,A: LongBool): Integer;		
C++:	int SetConfigurated (BOOL bX,BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A		
Example:	LSX.SetConfigurated(True,True,True,True);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!configurated	ValidConfig

LSX_GetSysSampleRate			
Description:	Command for reading the system sampling rate respectively sample time of the controller.		
Delphi:	function LSX_GetSysSampleRate (LSID: Integer; val rate : integer): Integer;		
C++:	int GetSysSampleRate (int *plrate);		
LabView:	Not supported		
Parameter:	rate	samplerate: 0: 40,0 µs / 25,0 kHz 1: 50,0 µs / 20,0 kHz 2: 62,5 µs / 16,0 kHz 3: 66,6… µs / 15,0 kHz 4: 75,0 µs / 13,3… kHz 5: 80,0 µs / 12,5 kHz	
Example:	LSX.GetSysSampleRate (&Rate);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?syssamplerate	-

LSX_SetSysSampleRate			
Description:	Command for customizing the system sampling rate respectively sample time of the controller.		
Delphi:	function LSX_SetSysSampleRate (LSID: Integer; rate : integer): Integer;		
C++:	int SetSysSampleRate (int lrate);		
LabView:	Not supported		
Parameter:	rate	samplerate: 0: 40,0 µs / 25,0 kHz 1: 50,0 µs / 20,0 kHz 2: 62,5 µs / 16,0 kHz 3: 66,6… µs / 15,0 kHz 4: 75,0 µs / 13,3… kHz 5: 80,0 µs / 12,5 kHz	
Example:	LSX.SetSysSampleRate (0);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!syssamplerate	!save and controller restart

LSX_GetBaud			
Description:	Command for reading the baud rate, maximum value depends on hardware interface		
Delphi:	function LSX_GetBaud (LSID: Integer; val rate : integer): Integer;		
C++:	int GetBaud (int *plrate);		
LabView:	Not supported		
Parameter:	rate	Possible values: 9600, 19200, 38400, 57600, 115200, 230400, 460800 USB: 460.800 baud Ethernet: 115.200 baud RS232: 115.200 baud PCIexpress: 115.200 baud	
Example:	LSX.GetBaud (&Rate);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?baud	-

LSX_SetBaud			
Description:	Command for setting the baud rate, maximum value depends on hardware interface		
Delphi:	function LSX_SetBaud (LSID: Integer; rate : integer): Integer;		
C++:	int SetBaud (int lrate);		
LabView:	Not supported		
Parameter:	rate	Possible values: 9600, 19200, 38400, 57600, 115200, 230400, 460800 USB: 460.800 baud Ethernet: 115.200 baud RS232: 115.200 baud PCIexpress: 115.200 baud	
Example:	LSX.SetBaud (115200);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!baud	Immediately

4.2.2 Control and API information

LSX_GetAPIVersion, LSX_GetAPIVersionW			
Description:	Shows the version number of LSTEP-API.		
Delphi:	LSX_GetAPIVersion(LSID: Integer; APIVer: PAnsiChar; MaxLen: Integer): Integer; LSX_GetAPIVersionW(LSID: Integer; APIVer: PWideChar; MaxLen: Integer): Integer;		
C++:	int GetAPIVersion (char *pcAPIVer, int lMaxLen) int GetVersionStrDetW (TCHAR *pcVersDet, int lMaxLen)		
LabView:			
Parameter:	APIVer	Buffer including the version number of LSTEP-API	
	MaxLen	Maximum number of characters, which may be copied into the buffer. The version number may have a length of approx. 20 characters.	
Example:	LSX.GetAPIVersion(pcVers, 100);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

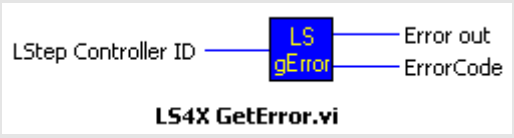
LSX_GetSerialNr, LSX_GetSerialNrW			
Description:	Reads serial number of controller. (not for LSTEP express)		
Delphi:	function LSX_GetSerialNr(LSID: Integer; SerialNr: PAnsiChar; MaxLen: Integer): Integer; function LSX_GetSerialNrW(LSID: Integer; SerialNr: PWideChar; MaxLen: Integer): Integer;		
C++:	int GetSerialNr (char *pcSerialNr,int lMaxLen); int GetSerialNr W(TCHAR *pcSerialNr,int lMaxLen);		
LabView:			
Parameter:	SerialNr	Pointer to a buffer in which the serial number is returned	
	MaxLen	Maximum number of characters, which may be copied into the buffer.	
Example:	LSX.GetSerialNr(pcSerialNr, 256);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?readsn	-

LSX_GetVersionStr, LSX_GetVersionStrW			
Description:	Returns the current firmware version number. For additional information, GetVersionStrDet and GetVersionStrInfo should also be used.		
Delphi:	function LSX_GetVersionStr(LSID: Integer; Vers: PAnsiChar; MaxLen: Integer): Integer; function LSX_GetVersionStrW(LSID: Integer; Vers: PWideChar; MaxLen: Integer): Integer;		
C++:	int GetVersionStr (char *pcVers,int lMaxLen);		
LabView:			
Parameter:	Vers	Pointer to a buffer in which the version string is returned	
	MaxLen	Maximum number of characters, which may be copied into the buffer.	
Example:	LSX.GetVersionStr(pcVers, 64);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?ver	-

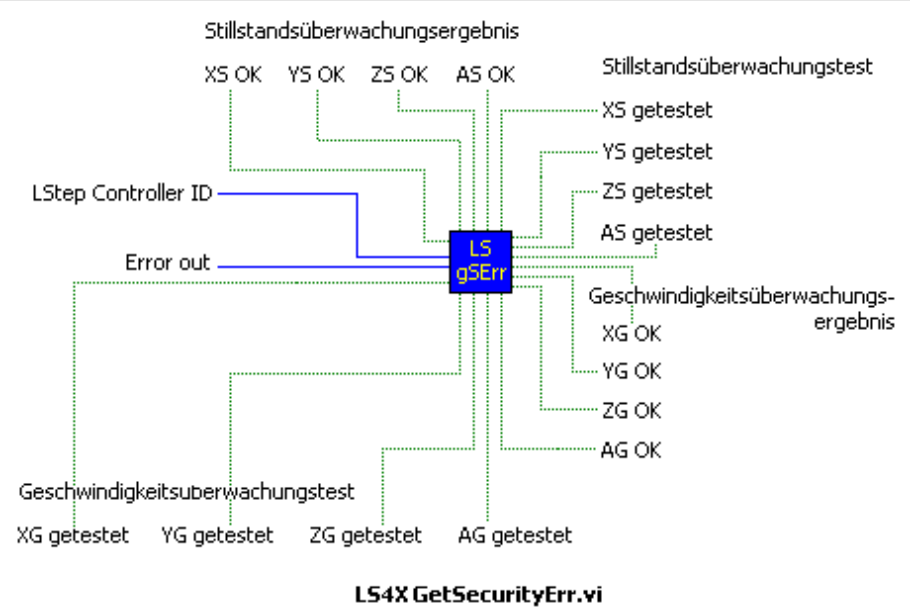
LSX_GetVersionStrDet, LSX_GetVersionStrDetW			
Description:	Function for reading the firmware configuration. For additional information, GetVersionStr and GetVersionStrInfo should also be used. (not for LSTEP express)		
Delphi:	function LSX_GetVersionStrDet(LSID: Integer; VersDet: PAnsiChar; MaxLen: Integer): Integer; function LSX_GetVersionStrDetW(LSID: Integer; VersDet: PWideChar; MaxLen: Integer): Integer;		
C++:	int GetVersionStrDet (char *pcVersDet, int lMaxLen); int GetVersionStrDetW (TCHAR *pcVersDet, int lMaxLen);		
LabView:			
Parameter:	VersDet	Pointer to a buffer in which the detailed version string is returned	
	MaxLen	Maximum number of characters, which may be copied into the buffer.	
Example:	LSX.GetVersionStrDet(pcVersDet, 64);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?det	-

LSX_GetVersionStrInfo, LSX_GetVersionStrInfoW			
Description:	Provides detailed information about version number. For additional information, GetVersionStr and GetVersionStrDet should also be used.		
Delphi:	function LSX_ GetVersionStrInfo (LSID: Integer; VersInfo: PAnsiChar; MaxLen: Integer): Integer; function LSX_ GetVersionStrInfoW (LSID: Integer; VersInfo: PWideChar; MaxLen: Integer): Integer;		
C++:	int GetVersionStrInfo (char *pcVersInfo, int lMaxLen); int GetVersionStrInfoW (TCHAR *pcVersInfo, int lMaxLen);		
LabView:			
Parameter:	VersInfo	Pointer to a buffer in which the detailed version number is saved. e.g.: T04.35.02-0004	
	MaxLen	Maximum number of characters, which may be copied into the buffer.	
Example:	LSX.GetVersionStrInfo (pcVersInfo, 64);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?iver	-

4.2.3 Status requests

LSX_GetError			
Description:	This function requests the current error number from the controller.		
Delphi:	function LSX_GetError(LSID: Integer; var ErrorCode: Integer): Integer;		
C++:	int GetError(int *plErrorCode);		
LabView:			
Parameter:	ErrorCode	Error number	
Example:	LSX.GetError(&ErrCode);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!err	-

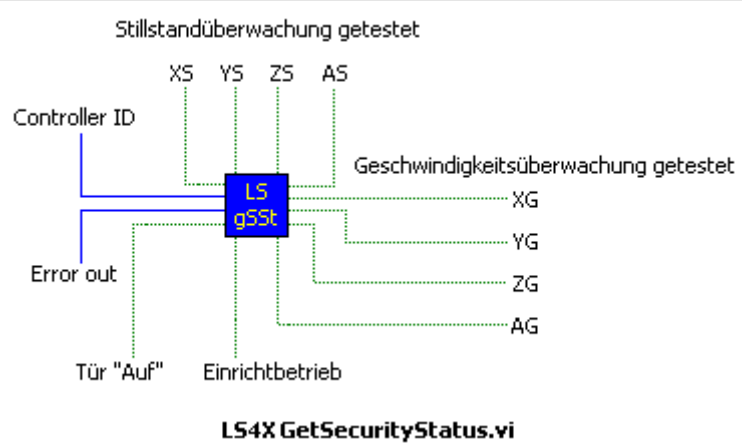
LSX_Quit			
Description:	This function acknowledges mistakes and resets the error number.		
Delphi:	function LSX_Quit(LSID:Integer):Integer;		
C++:	int Quit ();		
LabView:	Not supported		
Parameter:	-		
Example:	LSX.Quit();		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!quit	-

LSX_GetSecurityErr	
Description:	Reads all statuses and results of the GAL-safety monitoring. (only with LS44 controllers).
Delphi:	function LSX_GetSecurityErr(LSID: Integer; var Value: LongWord): Integer;
C++:	int GetSecurityErr(LongWord *pValue);
LabView:	

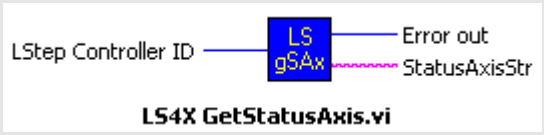
<table>
<tr><td>Parameter:</td><td>Value</td><td colspan="3">32-Bit LongWord without sign; contains the bit mask in the bits 0-15 after activating the function.

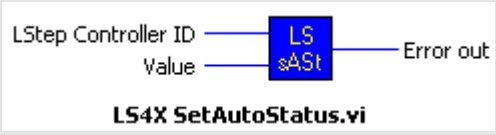
Value 0 for monitoring result = not OK
Value 1 for monitoring result = OK
Value 0 for monitoring test = not tested
Value 1 for monitoring test = tested

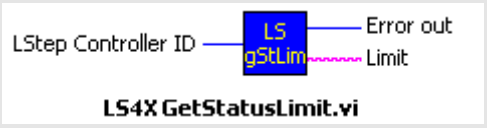
Bit 0 = X-axis standstill monitoring result
Bit 1 = Y-axis standstill monitoring result
Bit 2 = Z-axis standstill monitoring result
Bit 3 = A-axis standstill monitoring result
Bit 5 = X-axis standstill monitoring test
Bit 6 = Y-axis standstill monitoring test
Bit 7 = Z-axis standstill monitoring test
Bit 8 = A-axis standstill monitoring test
Bit 9 = X-axis speedl monitoring result
Bit 10 = Y-axis speed monitoring result
Bit 11 = Z-axis speed monitoring result
Bit 12 = A-axis speed monitoring result
Bit 13 = X-axis speed monitoring test
Bit 14 = Y-axis speed monitoring test
Bit 15 = Z-axis speed monitoring test</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.GetSecurityErr(&Value);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>1</td><td>?securityerror</td><td colspan="2">-</td></tr>
</table>

<table>
<tr><th colspan="5">LSX_GetSecurityStatus</th></tr>
<tr><td>Description:</td><td colspan="4">Shows the current status of safety monitoring.
(only with LS44 controllers).</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_GetSecurityStatus(LSID: Integer; var Value: LongWord): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int GetSecurityStatus(LongWord *pValue);</td></tr>
<tr><td>LabView:</td><td colspan="4"> 
</td></tr>
<tr><td>Parameter:</td><td>Value</td><td colspan="3">32-Bit LongWord without sign; contains the bit mask in the bits 0-15 after activating the function.
Bit 0-3 = internal flag
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 =Setup mode status (setup mode = 1)
Bit 15 = Door status (door “Open“ = 1)</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.GetSecurityStatus(&Value);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>1</td><td>?securitystatus</td><td colspan="2">-</td></tr>
</table>

LSX_GetStatus, LSX_GetStatusW			
Description:	Shows the current controller status.		
Delphi:	function LSX_GetStatus(LSID: Integer; Stat: PAnsiChar; MaxLen: Integer): Integer; function LSX_GetStatusW(LSID: Integer; Stat: PWideChar; MaxLen: Integer): Integer;		
C++:	int GetStatus(char *pcStat,int lMaxLen); int GetStatusW(TCHAR *pcStat,int lMaxLen);		
LabView:			
Parameter:	Stat	Pointer to a buffer in which the status string is returned	
	MaxLen	Maximum number of characters, which may be copied into the buffer.	
Example:	LSX.GetStatus(pcStat, 256);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?status	-

LSX_GetStatusAxis, LSX_GetStatusAxisW			
Description:	Shows the present status of the individual axes		
Delphi:	function LSX_GetStatusAxis(LSID: Integer; StatusAxisStr: PAnsiChar; MaxLen: Integer): Integer; function LSX_GetStatusAxisW(LSID: Integer; StatusAxisStr: PWideChar; MaxLen: Integer): Integer;		
C++:	int GetStatusAxis(char *pcStatusAxisStr,int lMaxLen); int GetStatusAxisW(TCHAR *pcStatusAxisStr,int lMaxLen);		
LabView:			
Parameter:	StatusAxisStr	Pointer to a buffer in which the status string is returned. The status string includes a character for each axis and ends with ".". - = Axis is not enabled @ = Axis stands and is ready M = Axis is moving (Motion) J = Joystick is activated C = Axis is being controlled * S = Axis has reached limit switch A = Axis is calibrated and ready E = Error during calibration (limit switch no released correctly) * F = Axis is in error status D = Acknowledgement after table stroke measuring U = Set-up mode * T = Timeout * (* only LSTEP 2000 series)	
Parameter:	MaxLen	Maximum number of characters, which may be copied into the buffer.	
Example:	LSX.GetStatusAxis(pcStatAxis, 256);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?statusaxis	-

<table>
<tr><th colspan="5">LSX_SetAutoStatus</th></tr>
<tr><td>Description:</td><td colspan="4">This function activates or deactivates the autoatatus.
Note: The AutoStatus mode should normally not be changed, as the LSTEP-API sets the correct mode for travel commands etc.; changing to 0 or 2 could result in errors.</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_SetAutoStatus(LSID: Integer; Value: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int SetAutoStatus(int lValue);</td></tr>
<tr><td>LabView:</td><td colspan="4"> 
</td></tr>
<tr><td>Parameter:</td><td>Value</td><td colspan="3">AutoStatus-Modus
0 = No status is transmitted by the controller.
1 = “Position reached” signals are sent automatically by the controller.
2 = “Position reached” signals and status messages are sent automatically by the controller. *
3 = Only a Carriage Return is fed back for “Position reached”. *
(* only LSTEP 2000 series)</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.SetAutoStatus(3);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>3 (see Value)</td><td>!autostatus</td><td colspan="2">-</td></tr>
</table>

<table>
<tr><th colspan="5">LSX_GetStatusLimit, LSX_GetStatusLimitW</th></tr>
<tr><td>Description:</td><td colspan="4">Shows the current state of the software limits of each axis.</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_GetStatusLimit(LSID: Integer; Limit: PAnsiChar; MaxLen: Integer): Integer;
function LSX_GetStatusLimitW(LSID: Integer; Limit: PWideChar; MaxLen: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int GetStatusLimit(char *pcLimit, int lMaxLen);
int GetStatusLimitW (TCHAR *pcLimit, int lMaxLen);</td></tr>
<tr><td>LabView:</td><td colspan="4"> 
</td></tr>
<tr><td rowspan="2">Parameter:</td><td>pc Limit</td><td colspan="3">Pointer to a buffer in which the condition of the axis is returned. E.g.: AA- A- - DD - LL- L- - L
A = Axis was calibrated
D = table stroke was measured
L = Software-limit was set
- = Software-limit was not changed</td></tr>
<tr><td>MaxLen</td><td colspan="3">Maximum number of characters which may be copied into the buffer</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.GetStatusLimit(pc Limit, 64);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>?statuslimit</td><td colspan="2">-</td></tr>
</table>

<table>
<tr><th colspan="5">LSX_GetSysstat, LSX_GetSysStatus</th></tr>
<tr><td>Description:</td><td colspan="4">Shows the current state of the axes.</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_GetSysStat(LSID: Integer; var X,Y,Z,A: Integer): Integer;
function LSX_GetSysStatus(LSID: Integer; Status :PWideChar; MaxLen: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int GetSysStat (int *plX, int *plY, int *plZ, int *plA);
int GetSysStatus(TCHAR *pcStatus, int MaxLen);</td></tr>
<tr><td>LabView:</td><td colspan="4">Not supported</td></tr>
<tr><td rowspan="16">Parameter:</td><td>SysStat</td><td colspan="2">SysStatus</td><td>Description</td></tr>
<tr><td>0</td><td colspan="2">Just turned On</td><td>Status after controller activation</td></tr>
<tr><td>1</td><td colspan="2">not configurated</td><td>Axis is not configurated</td></tr>
<tr><td>2</td><td colspan="2">Configurated</td><td>Axis is configurated</td></tr>
<tr><td>3</td><td colspan="2">Ready for Komm</td><td>Axis is ready for auto-commutation</td></tr>
<tr><td>4</td><td colspan="2">Autokommutating</td><td>Axis is auto-commutating</td></tr>
<tr><td>5</td><td colspan="2">Ready for Power</td><td>Axis is ready for power amplifier activation</td></tr>
<tr><td>6</td><td colspan="2">Positioncontrol</td><td>Axis is in position control mode or stepping motor operation</td></tr>
<tr><td>7</td><td colspan="2">Speedcontrol</td><td>Axis is in speed control mode</td></tr>
<tr><td>8</td><td colspan="2">Manual Mode</td><td>Axis is in manual mode (e.g. joystick)</td></tr>
<tr><td>9</td><td colspan="2">Calibrating</td><td>Axis is being calibrated</td></tr>
<tr><td>10</td><td colspan="2">Error-Power-Off</td><td>Driver voltage, intermediate circuit voltage, motor temperature, power amplifier temperature, autocommutation or power amplifier hardware error</td></tr>
<tr><td>12</td><td colspan="2">Error-Stop-On</td><td>Error due to deviation of position controller (contouring error) or I²T motor monitoring</td></tr>
<tr><td>13</td><td colspan="2">System Error</td><td>System error → contact manufacturer</td></tr>
<tr><td>15</td><td colspan="2">Changing</td><td>System status change active</td></tr>
<tr><td colspan="3">Note:</td><td>State 11 and 14 are not implementet</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.GetSysStat(&X,&Y,&Z,&A);
LSX.GetSysStatus(&Status);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td colspan="2">“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>2</td><td colspan="2">?sysstat
?sysstatus</td><td>-</td></tr>
</table>

LSX_GetStopStatus			
Description:	Shows the state of the stop input while taking the stop-polarity into consideration (see Command "stoppol").		
Delphi:	function LSX_GetStopStatus(LSID: Integer; StopStatus: Boolean): Integer;		
C++:	int GetStopStatus (BOOL *pbStopStatus);		
LabView:	Not supported		
Parameter:	Stop input status: False = Stop inactive True = Stop active		
Example:	LSX.GetStopStatus(&StopStatus);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?stopstatus	-

LSX_GetPowerAmplifierStatus			
Description:	Command for reading the power amplifier status including switching on and switching off processes (with taking into consideration motor brake outputs).		
Delphi:	function LSX_GetPowerAmplifierStatus (LSID: Integer; var X, Y, Z, R: Integer): Integer;		
C++:	int GetPowerAmplifierStatus (int *pbX, int *pbY, int *pbZ, int *pbR);		
LabView:	Not supported		
Parameter:	Status of the Input: 000: Power amplifiere is switched off, no process is active 010: Power amplifier is switched off and switching off process (with taking into consideration motor brake outputs) is active 100: Power amplifier is switched off and switching on process (with taking into consideration motor brake outputs) is active 001: Power amplifier is switched on, no process is active 011: Power amplifier is switched on and switching off process (with taking into consideration motor brake outputs) is active 101: Power amplifier is switched on and switching on process (with taking into consideration motor brake outputs) is active		
Example:	LSX.GetPowerAmplifierStatus(&X, &Y, &Z, &R);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?pastatus	-

LSX_GetPTemp			
Description:	Command for reading the power amplifier temperatures		
Delphi:	function LSX_GetPTemp (LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int GetPTemp (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Nicht unterstützt		
Parameter:	X, Y, Z, A	Temperature for every Axis in [°C]	
Example:	LSX.GetPTemp (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?ptemp	-

4.2.4 Parameter handling

LSX_SetControlPars			
Description:	Transmits the parameters loaded by LSX_LoadConfig to the LSTEP.		
Delphi:	function LSX_SetControlPars(LSID: Integer): Integer;		
C++:	int SetControlPars ();		
LabView:			
Parameter:	-		
Example:	LSX.SetControlPars();		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	-	-

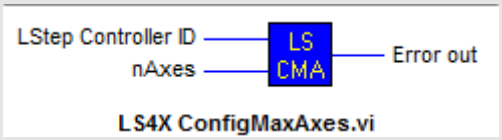
LSX_LStepSave			
Description	This function triggers the saving of the current configuration in the non-volatile storage of the LSTEP controller. The controller feedback showing that saving is completed is awaited.		
Delphi	function LSX_LStepSave(LSID: Integer): Integer;		
C++	int LStepSave();		
LabView:			
Parameter	-		
Example	LSX.LStepSave() ;		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!save	-

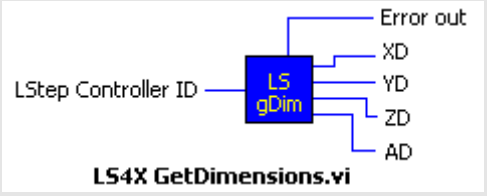
LSX_SoftwareReset			
Description:	The controller is restarted. The controller loads the data saved last.		
Delphi:	function LSX_SoftwareReset(LSID: Integer): Integer;		
C++:	int SoftwareReset();		
LabView:			
Parameter:	-		
Example:	LSX.SoftwareReset();		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!reset	-

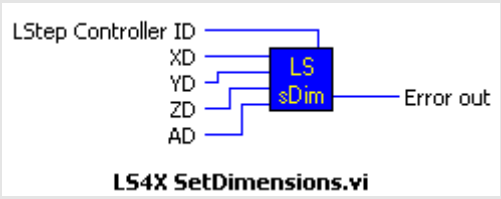
4.2.5 Axis and motor configuration

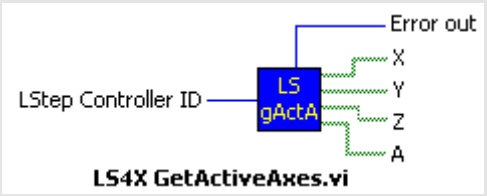
LSX_ValidConfig			
Description:	Activates the parameters which are dependend on ValidConfig for the chosen axis. Positions get reset. New calibration is mandatory.		
Delphi:	function LSX_ValidConfig(LSID: Integer; AxisInt: Integer): Integer;		
C++:	int ValidConfig (int lAxisint);		
LabView:	Not supported		
Parameter:	AxisInt	0 = Use all axes 1 = Use x-axis 2 = Use y-axis 3 = Use z-axis 4 = Use a-axis	
Example:	LSX.ValidConfig(0); // ValidConfig command on all axis		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!validconfig	-

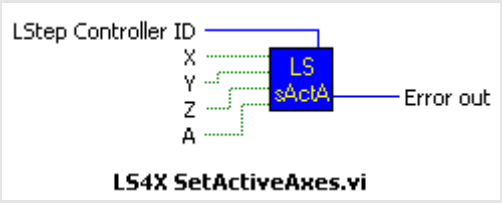
LSX_ValidPar			
Description:	Transmits changed parameters of a certain axis or all axes to the control unit. There is no need for a new calibration.		
Delphi:	function LSX_ValidPar(LSID: Integer; AxisInt: Integer): Integer;		
C++:	int ValidPar(int lAxisInt);		
LabView:	Not supported		
Parameter:	AxisInt	0 = Transmit all axes 1 = Transmit x-axis 2 = Transmit y-axis 3 = Transmit z-axis 4 = Transmit a-axis	
Example:	LSX.ValidPar(0); // Transmit all axes		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!validpar	-

LSX_ConfigMaxAxes			
Description:	Configures the number of axes.		
Delphi:	function LSX_ConfigMaxAxes(LSID: Integer; nAxes: Integer): Integer;		
C++:	int ConfigMaxAxes(int nAxes);		
LabView:			
Parameter:	nAxes	Number of axes	
Example:	LSX.ConfigMaxAxes(2); // controller has two axes		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!configmaxaxis	-

LSX_GetDimensions		
Description:	Inquiry of set axes dimensions.	
Delphi:	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 = Millimetres 3 = Degrees 4 = Revolutions
Example:	LSX.GetDimensions(&XD, &YD, &ZD, &AD);	
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?dim	-

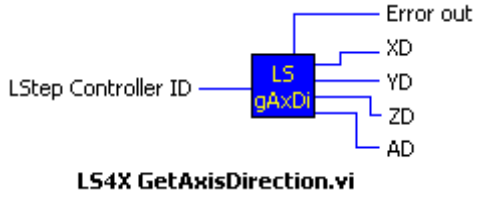
<table>
<tr><th colspan="4">LSX_SetDimensions</th></tr>
<tr><td>Description:</td><td colspan="3">Sets axes dimensions.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_SetDimensions(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int SetDimensions(int lXD,int lYD,int lZD,int lAD);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>XD, YD, ZD, AD</td><td colspan="2">Dimension values
0 = Microsteps
1 = µm
2 = Millimetres
3 = Degrees
4 = Revolutions</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.SetDimensions(3, 2, 2);
// X-axis in degrees; Y and Z in mm</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>!dim</td><td>-</td></tr>
</table>

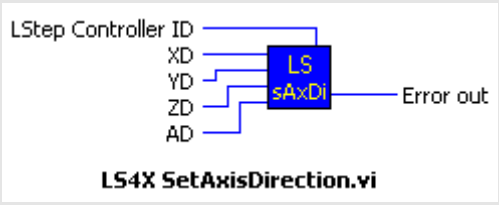
<table>
<tr><th colspan="5">LSX_GetActiveAxes</th></tr>
<tr><td>Description:</td><td colspan="4">Shows the released axes.</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_GetActiveAxes(LSID: Integer; var Flags: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int GetActiveAxes(int *plFlags);</td></tr>
<tr><td>LabView:</td><td colspan="4"> 
</td></tr>
<tr><td>Parameter:</td><td>Flags</td><td colspan="3">32-bit integer containing a bit mask after calling the function in the bits 0-4.
Bit 0 = X-axis
Bit 1 = Y-axis
…
Value 0 = Axis deactivated
Value 1 = Axis activated</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.GetActiveAxes(&Flags);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>?axis</td><td colspan="2">-</td></tr>
</table>

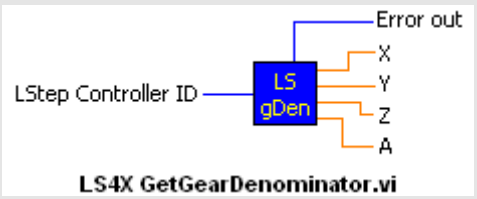
LSX_SetActiveAxes			
Description:	Sets axes release.		
Delphi:	function LSX_SetActiveAxes(LSID: Integer; Flags: Integer): Integer;		
C++:	int SetActiveAxes(int Flags);		
LabView:			
Parameter:	Flags	Bit mask for axis release Bit 0 = X-axis Bit 1 = Y-axis … Value 0 = Axis deactivated Value 1 = Axis activated	
Example:	LSX.SetActiveAxes(3); /* X and Y-axis are released (bits 0 and 1 set), Z-axis not released (bit 2 = 0) */		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!axis	-

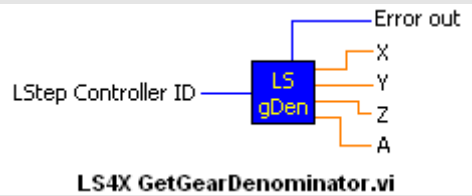
LSX_GetAxisMap			
Description:	Command for reading the configuration of the hardware-axes to the axes of the user-interface		
Delphi:	function LSX_GetAxisMap (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetAxisMap (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Numbering of the axis: 1, 2, 3 or 4	
Example:	LSX.GetAxisMap(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?axismap	-

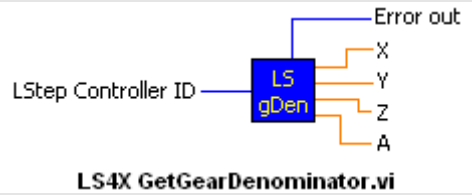
LSX_SetAxisMap			
Description:	Command for mapping of the hardware-axes to the axes of the user-interface		
Delphi:	function LSX_SetAxisMap (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetAxisMap (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Numbering of the axis: 1, 2, 3 or 4	
Example:	LSX.SetAxisMap(2, 1, 4, 3);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!axismap	ValidConfig

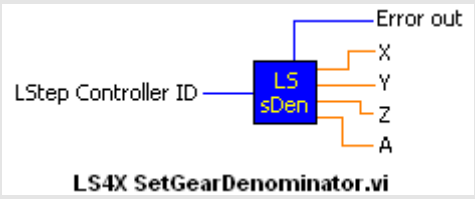
LSX_GetAxisDirection			
Description	Function for change of direction inquiry.		
Delphi	function LSX_GetAxisDirection(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;		
C++	int GetAxisDirection(int *plXD, int *plYD, int *plZD, int *plAD);		
LabView:			
Parameter	XD, YD, ZD, AD	32-bit integer with indication of direction of rotation. 0 = normal direction of rotation 1 = reverse direction of rotation	
Example	LSX.GetAxisDirection(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?axisdir	-

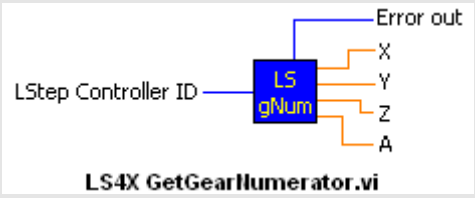
LSX_SetAxisDirection			
Description	Function for setting the change of direction.		
Delphi	function LSX_SetAxisDirection(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;		
C++	int SetAxisDirection (int lXD, int lYD, int lZD, int lAD);		
LabView:			
Parameter	XD, YD, ZD, AD	32-bit integer with indication of direction of rotation. 0 = normal direction of rotation 1 = reverse direction of rotation	
Example	LSX.SetAxisDirection(1, 0, 0, 0); // reverse direction of X-axis		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!axisdir	ValidConfig

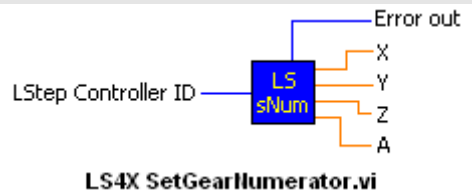
LSX_GetGear			
Description:	Function for gear ratio inquiry. (not for LSTEP express)		
Delphi:	function LSX_GetGear(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetGear (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	X, Y, Z, A	Read-out gear ratio (motor/actuator side)	
Example:	LSX.GetGear(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?gear	-

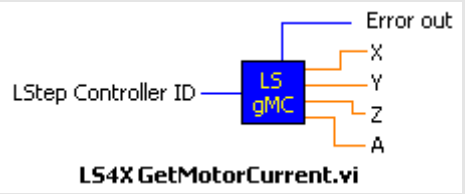
LSX_SetGear			
Description:	Function for gear ratio setting. (not for LSTEP express)		
Delphi:	function LSX_SetGear(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int SetGear (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	X, Y, Z, A	Gear ratio to be set (motor/actuator side)	
Example:	LSX.SetGear (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!gear	-

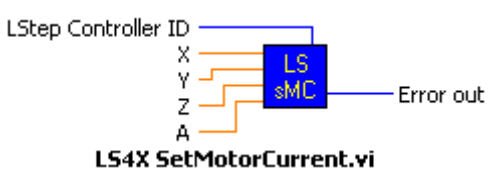
LSX_GetGearDenominator			
Description:	Inquires denominator of gear ratio. (only for LSTEP express)		
Delphi:	function LSX_GetGearDenominator(LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetGearDenominator(int *plX, int *plY, int *plZ, int *plA);		
LabView:			
Parameter:	X, Y, Z, A	Denominator of gear ratio (motor side)	
Example:	LSX.GetGearDenominator(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?geardenominator	-

LSX_SetGearDenominator			
Description:	Adjust denominator of gear ratio. (only for LSTEP express)		
Delphi:	function LSX_SetGearDenominator(LSID: Integer; Integer): Integer; Integer;		
C++:	int SetGearDenominator(int lX, int lY, int lZ, int lA);		
LabView:			
Parameter:	X, Y, Z, A	Value of gear ratio denominator (motor side)	
Example:	LSX.SetGearDenominator(2, 1, 1, 1); // gear ratio 2/γ with x		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!geardenominator	ValidConfig

LSX_GetGearNumerator			
Description:	Inquiry of gear ratio numerator. (only for LSTEP express)		
Delphi:	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	Gear ratio numerator (actuator side).	
Example:	LSX.GetGearNumerator(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?gearnumerator	-

LSX_SetGearNumerator			
Description	Adjust numerator of gear ratio. (only for LSTEP express)		
Delphi	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	Value of gear ratio numerator (motor side)	
Example	LSX.SetGearNumerator(4, 1, 1, 1);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?pos	-

LSX_GetMotorCurrent			
Description:	Function for inquiring the set motor current.		
Delphi:	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 in A	
Example:	LSX.GetMotorCurrent(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	LSTEP 2000: ?cur LSTEP express: ?motorcurrent	-

LSX_SetMotorCurrent			
Description:	Function for setting the motor current.		
Delphi:	function LSX_SetMotorCurrent(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetMotorCurrent (double dX,double dY,double dZ,double dA);		
LabView:			
Parameter:	X, Y, Z, A	Motor current in A	
Example:	LSX.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		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	LSTEP2000: !cur LSTEP express: !motorcurrent	ValidPar / ValidConfig

LSX_GetMotorpeakCurrent			
Description:	Function for inquiring the set motor peak current.		
Delphi:	function LSX_GetMotorpeakCurrent(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetMotorpeakCurrent(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Motor current in A	
Example:	LSX.GetMotorpeakCurrent(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorpeakcurrent	-

LSX_SetMotorpeakCurrent			
Description:	Function for setting the motor peak current.		
Delphi:	function LSX_SetMotorpeakCurrent(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetMotorpeakCurrent (double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Motor current in A	
Example:	LSX.SetMotorpeakCurrent(1.5, 1.5, 1.0, 1.0); // Motor peak current for X and Y is 1.5 amperes; for Z and A 1.0 amperes		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!motorpeakcurrent	ValidPar / ValidConfig

LSX_GetMotortempSensor			
Description:	Function for inquiring the state of the motor temperature sensor.		
Delphi:	function LSX_GetMotortempSensor(LSID: Integer; var X, Y, Z, A: longBool): Integer;		
C++:	int Getmotortempsensor (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Motor temperature resistore state	
Example:	LSX.GetMotortempSensor(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motortempsensor	-

LSX_SetMotortempSensor			
Description:	Function for activating/deactivating the motor temperature sensor.		
Delphi:	function LSX_SetMotortempSensor(LSID: Integer; X, Y, Z, A: longBool): Integer;		
C++:	int SetMotortempSensor (double dX,double dY,double dZ,double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Motortemperature resistore state	
Example:	LSX.SetMotortempSensor(True,True,False,False); //The temperature sensors for X and Y are activated; for Z and A are deactivated		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!motortempSensor	ValidPar / ValidConfig

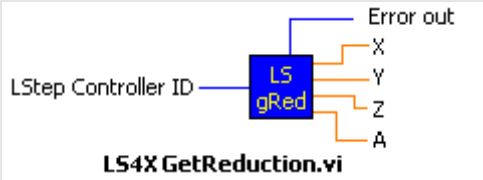
LSX_GetMotortempSensormin			
Description:	Function for inquiring the lowest acceptable motor temperature resistance. (only for LSTEP express)		
Delphi:	function LSX_GetMotortempSensormin(LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetMotortempSensormin (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Motor temperature resistance in Ω	
Example:	LSX.GetMotortempSensormin(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motortempSensormin-imum	-

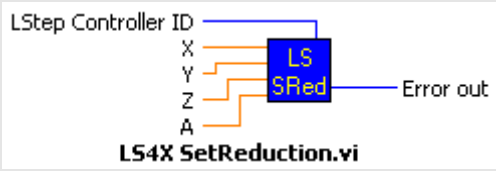
LSX_SetMotortempSensormin			
Description:	Function for setting the lowest acceptable motor temperature resistance.		
Delphi:	function LSX_SetMotortempSensormin (LSID: Integer; X, Y, Z, A:Integer): Integer;		
C++:	int SetMotortempSensormin (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Motor temperature resistance in Ω	
Example:	LSX.SetMotortempSensormin(150, 150, 100, 100); // Motor temperature sensor resistance for X and Y is 150 Ω; for Z and A 100 Ω		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!motortempsensormini-mum	ValidPar / ValidConfig

LSX_GetMotortempSensormax			
Description:	Function for inquiring the highest acceptable motor temperature resistance.		
Delphi:	function LSX_GetMotortempSensormax(LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetMotortempSensormax (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Motor temperature resistance in Ω	
Example:	LSX.GetMotortempSensormin(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motortempSensormax-imum	-

LSX_SetMotortempSensormax			
Description:	Function for setting the highest acceptable motor temperature resistance.		
Delphi:	function LSX_SetMotortempSensormax (LSID: Integer; X, Y, Z, A:Integer): Integer;		
C++:	int SetMotortempSensormax (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Motor temperature resistance in Ω	
Example:	LSX.SetMotortempSensormax(150, 150, 100, 100); //Highest acceptable Motor temperature resistance for X and Y is 150 Ω; for Z and A 100 Ω		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!motortempsensormax-imum	ValidPar / ValidConfig

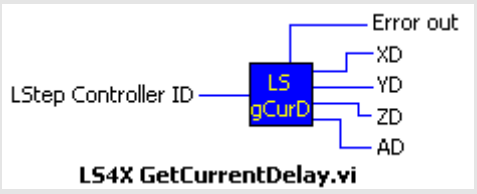
LSX_GetMotortempSensorValue			
Description:	Function for inquiring the motor temperature resistance.		
Delphi:	function LSX_GetMotortempSensorValue(LSID: Integer; var X,Y,Z,A: Integer): Integer;		
C++:	int GetMotortempSensorValue (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Motor temperature resistance in Ω	
Example:	LSX.GetMotortempSensorValue(X, Y, Z, A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motortempsensor-value	-

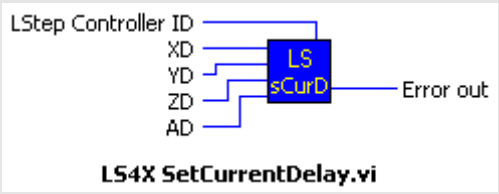
LSX_GetReduction			
Description:	Inquiry of set current reduction.		
Delphi:	function LSX_GetReduction(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetReduction(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	X, Y, Z, A	Set current reduction in %	
Example:	LSX.GetReduction(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?reduction	-

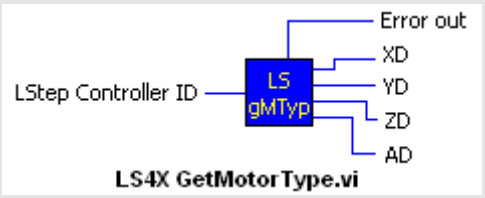
LSX_SetReduction			
Description:	Setting the ratio by which the motor rated current is reduced, if current reduction is active.		
Delphi:	function LSX_SetReduction(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetReduction(double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	X, Y, Z, A	Current reduction value. LSTEP Series = 0.0 – 1.0 LSTEP express series = 0 – 100 in %	

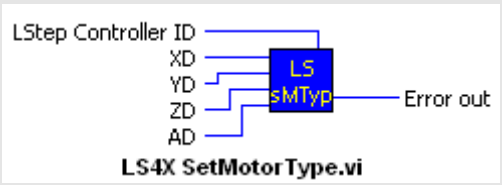
Example:	LSX.SetReduction(0.1, 0.7, 0.5, 0.5); //LSTEP 2000 series
	/* X-axis bias current = <0.1*rated current; Y-axis = 0.7*rated current … */		

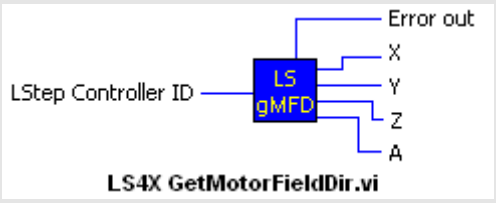
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!reduction	-

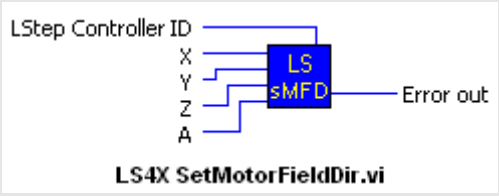
LSX_GetCurrentDelay			
Description:	Indicates the time delay until the current reduction is activated.		
Delphi:	function LSX_GetCurrentDelay(LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetCurrentDelay(int *plX, int *plY, int *plZ, int *plA);		
LabView:			
Parameter:	X, Y, Z, A:	Time delay in ms	
Example:	LSX.SetCurrentDelay(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?curdelay	-

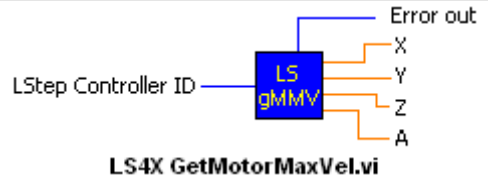
LSX_SetCurrentDelay			
Description:	Indicates the time delay for activating the current reduction.		
Delphi:	function LSX_SetCurrentDelay(LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetCurrentDelay(int lX, int lY, int lZ, int lA);		
LabView:			
Parameter:	X, Y, Z, A	Time delay in ms	
Example:	LSX.SetCurrentDelay(100, 300, 1000, 0) ;		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!curdelay	-

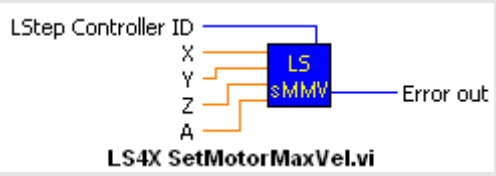
LSX_GetMotorType			
Description:	Function for reading the set motor type. (only for LSTEP express)		
Delphi:	function LSX_GetMotorType(LSID: Integer; var X: Integer; var Y: Integer; var Z: Integer; var A: Integer): Integer;		
C++:	Int GetMotorType(int *plX, int *plY, int *plZ, int *plA)		
LabView:			
Parameter:	X, Y, Z, A	Motor type 0 = rotary 2-phase stepper motor 1 = linear 2-phase stepper motor 2 = rotary 3-phase stepper motor 3 = linear 3-phase stepper motor 4 = rotary 2-phase servomotor 5 = linear 2-phase servomotor 6 = rotary 3-phase servomotor 7 = linear 3-phase servomotor	
Example:	LSX.GetMotorType(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motortype	-

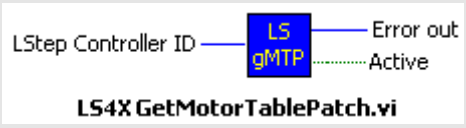
<table>
<tr><th colspan="4">LSX_SetMotorType</th></tr>
<tr><td>Description:</td><td colspan="3">Function for setting the motor type.
(only for LSTEP express)</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_SetMotorType(LSID: Integer; X: Integer; var Y: Integer; var Z: Integer; var A: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int SetMotorType(int lX, int lY, int lZ, int lA);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>X, Y, Z, A</td><td colspan="2">Motor type
0 = rotary 2-phase stepper motor
1 = linear 2-phase stepper motor
2 = rotary 3-phase stepper motor
3 = linear 3-phase stepper motor
4 = rotary 2-phase servomotor
5 = linear 2-phase servomotor
6 = rotary 3-phase servomotor
7 = linear 3-phase servomotor</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.SetMotorType(5, 4, 4, 4);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>2</td><td>!motortype</td><td>ValidConfig</td></tr>
</table>

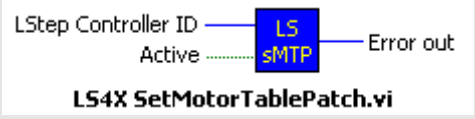
LSX_GetMotorFieldDir			
Description:	Function for inquiring the field direction of the motor. (only for LSTEP express)		
Delphi:	function LSX_GetMotorFieldDir(LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetMotorFieldDir(int *plX, int *plY, int *plZ, int *plA);		
LabView:			
Parameter:	X, Y, Z, A	Currently set direction 0 = normal field direction 1 = reversed field direction	
Example:	LSX.GetMotorFieldDir(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorfielddir	-

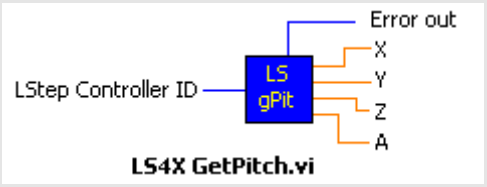
LSX_SetMotorFieldDir			
Description:	Function for setting the field direction of the motor. (only for LSTEP express)		
Delphi:	function LSX_SetMotorFieldDir(LSID: Integer; Integer): Integer; Integer;		
C++:	int SetMotorFieldDir(int lX, int lY, int lZ, int lA);		
LabView:			
Parameter:	X, Y, Z, A	Field direction currently to be set 0 = normal field direction 1 = reverse field direction	
Example:	LSX.SetMotorFieldDir(1, 0, 0, 0); // reverse direction of X-axis		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!motorfielddir	ValidConfig

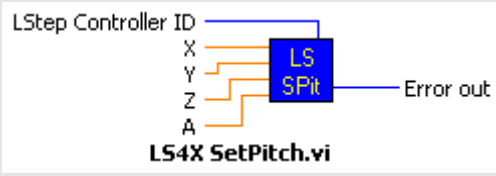
LSX_GetMotorMaxVel			
Description:	Reading of maximum motor Speed or velocity. (only for LSTEP express)		
Delphi:	function LSX_GetMotorMaxVel(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;		
C++:	int GetMotorMaxVel(double *pdXD, double *pdYD, double *pdZD, double *pdAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Motor speed or velocity. In rpm for rotary motor. In mm/s for linear motor.	
Example:	LSX.GetMotorMaxVel(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motormaxvel	-

LSX_SetMotorMaxVel			
Description:	Setting of maximum Motor Speed or velocity. (only for LSTEP express)		
Delphi:	function LSX_SetMotorMaxVel(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetMotorStandForce(double dXD, double dYD, double dZD, double dAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Motor speed or velocity. In rpm for rotary motor. In mm/s for linear motor.	
Example:	LSX.GetMotorMaxVel(1000, 1000, 500, 250);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!motormaxvel	ValidConfig

LSX_GetMotorTablePatch			
Description:	Indicates whether the correction table is activated. (not for LSTEP express)		
Delphi:	function LSX_GetMotorTablePatch (LSID: Integer; var bActive: LongBool): Integer;		
C++:	int GetMotorTablePatch (BOOL *pbActive);		
LabView:			
Parameter:	bActive	Correction table status True = Table is activated False = Table is deactivated	
Example:	LSX.GetMotorTablePatch(&Active);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?motorpatch	-

LSX_SetMotorTablePatch			
Description:	The correction table is activated. The correction table for a special motor was determined by measurement. Correction tables can be determined upon the customer's request. (not for LSTEP express)		
Delphi:	function LSX_SetMotorTablePatch (LSID: Integer; bActive: LongBool): Integer;		
C++:	int SetMotorTablePatch (BOOL bActive);		
LabView:			
Parameter:	bActive	Activation of correction table True = Table is activated False = Table is deactivated	
Example:	LSX.SetMotorTablePatch(True);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!motorpatch	-

LSX_GetPitch			
Description:	Shows the set value of the spindle pitch.		
Delphi:	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 pitches in mm/revolution	
Example:	LSX.GetPitch(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?pitch	-

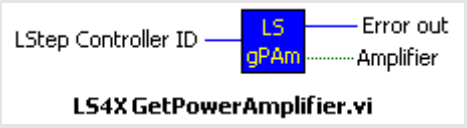
LSX_SetPitch			
Description:	Sets the axis spindle pitch.		
Delphi:	function LSX_SetPitch(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetPitch(double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	X, Y, Z, A	Spindle pitches in mm/revolution	
Example:	LSX.SetPitch(4, 4, 4, 4); // Set spindle pitches to 4 mm for all axes		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!pitch	ValidConfig

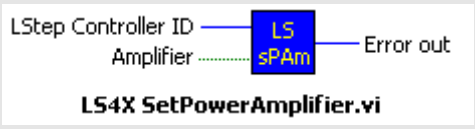
LSX_GetXYAxisComp			
Description	Inquiry of XY-axis overlay (not for LSTEP express)		
Delphi	function LSX_GetXYAxisComp(LSID: Integer; var Value: Integer): Integer;		
C++	int GetXYAxisComp (int *plValue);		
LabView:			
Parameter	Value	Mode of axis overlay (see LSTEPdocumentation)	
Example	LSX.SetXYAxisComp(&mode) ;		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?xycomp	-

LSX_SetXYAxisComp			
Description	Activate XY-axis overlay (not for LSTEP express)		
Delphi	function LSX_SetXYAxisComp(LSID: Integer; Value: Integer): Integer;		
C++	int SetXYAxisComp(int lValue);		
LabView:			
Parameter	Value	Mode of axis overlay (see LSTEPdocumentation)	
Example	LSX.SetXYAxisComp(1) ;		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!xycomp	-

LSX_GetStopPolarity			
Description:	Reading of stop input polarity.		
Delphi:	function LSX_GetStopPolarity(LSID: Integer; var bHighActiv: LongBool): Integer;		
C++:	int GetStopPolarity(BOOL *pbHighActiv);		
LabView:			
Parameter:	bHighActiv	Value of set polarity True = Stop input high-active False = Stop input low-active	
Example:	LSX.GetStopPolarity(&HighActiv);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?stoppol	-

LSX_SetStopPolarity			
Description:	Function for setting the stop input polarity.		
Delphi:	function LSX_SetStopPolarity(LSID: Integer; bHighActiv: LongBool): Integer;		
C++:	int SetStopPolarity(BOOL bHighActiv);		
LabView:			
Parameter:	bHighActiv	Value of set polarity True = Stop input high-active False = Stop input low-active	
Example:	LSX.SetStopPolarity(False); //The stop input is low active.		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!stoppol	-

LSX_GetPowerAmplifier			
Description:	Indicates whether the power amplifiers of the LS44 are activated or deactivated. (only for LS44 controll and for LSTEP express)		
Delphi:	function LSX_GetPowerAmplifier(LSID: Integer; var bAmplifier: LongBool): Integer;		
C++:	int GetPowerAmplifier (BOOL *pbAmplifier);		
LabView:			
Parameter:	bAmplifier	Power amplifier status True = All power amplifiers are activated False = All power amplifiers are deactivated	
Example:	LSX.GetPowerAmplifier(&Amplifier);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3 (see description)	?pa	-

LSX_SetPowerAmplifier			
Description:	Activates or deactivates the power amplifiers of the controller. (only for LS44 controll and for LSTEP express)		
Delphi:	function LSX_SetPowerAmplifier (LSID: Integer; bAmplifier: LongBool): Integer;		
C++:	int SetPowerAmplifier(BOOL bAmplifier);		
LabView:			
Parameter:	bAmplifier	Intended power amplifier status. True = All power amplifiers are activated False = All power amplifiers are deactivated	
Example:	LSX.SetPowerAmplifier(True); // Activate power amplifiers		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3 (see description)	!pa	-

LSX_GetPowerAmplifierMode			
Description:	Shows the target state of the power amplifiers		
Delphi:	function LSX_GetPowerAmplifierMode(LSID: Integer; var X, Y, Z, R : integer): Integer;		
C++:	int GetPowerAmplifierMode (int *plX, int *plY, int *plZ, int *plR);		
LabView:	-		
Parameter:	X,Y,Z,R	Target state of the power amplifiers 0 = All power amplifiers are deactivated 1 = All power amplifiers are activated 2 = Activates with restart after switching off and switching on again the driver voltage respectively the intermediate circuit voltage	
Example:	LSX.GetPowerAmplifierMode(&X,&Y,&Z,&R);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?pamode	-

LSX_GetModulo			
Description:	Function for inquiring if the axes are in modulo operation mode.		
Delphi:	function LSX_GetModulo(LSID: Integer; var X,Y,Z,A: LongBool): Integer;		
C++:	int GetModulo (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	True = Activate the modulo operation mode for this axis False = Deactivate the modulo operation mode for this axis	
Example:	LSX.GetModulo(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?modulo	-

LSX_SetModulo			
Description:	Activates/ Deactivates the modulo operation mode.		
Delphi:	function LSX_SetModulo(LSID: Integer; X,Y,Z,A: LongBool): Integer;		
C++:	int SetModulo (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	True = Activate the modulo operation mode for this axis False = Deactivate the modulo operation mode for this axis	
Example:	LSX.SetModulo(True,True,True,True); //All axes in modulo operation mode.		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!modulo	ValidConfig

LSX_GetModulomode			
Description:	Function for inquiring in which modulo operation mode the axes currently are.		
Delphi:	function LSX_GetModuloMode(LSID: Integer; var X,Y,Z,A: Integer): Integer;		
C++:	int GetModuloMode (int*plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	0 = Move is controlled without optimisation . 1 = Target position is always aproached in positive direction of rotation. 2 = Target position is always approached in negative direction of rotation. 3 = The shortest connection between start and target position is always used.	
Example:	LSX.GetModulomode(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?modulomode	-

LSX_SetModulomode			
Description:	Sets the modulo operation mode for each axis.		
Delphi:	function LSX_SetModuloMode(LSID: Integer; X,Y,Z,A: Integer): Integer;		
C++:	int SetModulomode (int lX, int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	0 = Move is controlled without optimisation . 1 = Target position is always aproached in positive direction of rotation. 2 = Target position is always approached in negative direction of rotation. 3 = The shortest connection between start and target position is always used.	
Example:	LSX.SetModulomode(0,0,0,0); //All axes operate without optimisation		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!modulomode	-

LSX_GetMotorPoleScale			
Description:	Function for inquiring the pole scale of linear motors.		
Delphi:	function LSX_GetMotorPoleScale (LSID: Integer; X,Y,Z,A: double): Integer;		
C++:	Int GetMotorPoleScale(int lLSID, double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Pole scale in mm	
Example:	LSX.GetMotorPoleScale(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString"command	Activation (LSTEP express series)
	2	?motorpolescale	-

LSX_SetMotorPoleScale			
Description:	Sets the pole scale of linear motors.		
Delphi:	function LSX_SetMotorPoleScale(LSID: Integer; X,Y,Z,A : double) : Integer;		
C++:	Int SetMotorPoleScale (int lLSID, double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	PoleScale in mm	
Example:	LSX.SetMotorPoleScale(10,10,10,10);		
Other:	Compatibility	"SendString"command	Activation (LSTEP express series)
	2	!motorpolescale	-

LSX_GetMotorPolePairs			
Description:	Function for inquiring the pole pair number of rotative motors.		
Delphi:	function LSX_GetMotorPolePairs (LSID: Integer; X,Y,Z,A: integer): Integer;		
C++:	Int GetMotorPolePairs(int lLSID, int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Pole pair number	
Example:	LSX.GetMotorPolePairs(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorpolpairs	-

LSX_SetMotorPolePairs			
Description:	Sets the pole pair number of rotative motors		
Delphi:	function LSX_SetMotorPolePairs (LSID: Integer; X,Y,Z,A : integer): Integer;		
C++:	Int SetMotorPolePairs (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Pole pair number	
Example:	LSX.SetPolePairs(10,10,10,10);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!motorpolepairs	-

LSX_GetMotorPolePairRes			
Description:	Function for inquiring the step resolution of pole pairs and/or pole scale.		
Delphi:	function LSX_GetMotorPolePairRes (LSID: Integer; X,Y,Z,A: integer): Integer;		
C++:	Int GetMotorPolePairRes(int lLSID, int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Step resolution in mm	
Example:	LSX.GetMotorPolePairRes(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorpolepairres	-

LSX_SetMotorPolePairRes			
Description:	Sets the step resolution of pole pairs and / or pole scale.		
Delphi:	function LSX_SetMotorPolePairRes(LSID: Integer; X,Y,Z,A : Integer): Integer;		
C++:	Int SetMotorPolePairRes (int lLSID, int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Step resolution in mm	
Example:	LSX.SetMotorPolePairRes(50,50,50,50);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorpolepairres	ValidConfig

LSX_GetMotorBrake			
Description:	Command for reading the motor brake output mode.		
Delphi:	function LSX_GetMotorBrake (LSID: Integer; X, Y, Z, A: integer): Integer;		
C++:	Int GetMotorBrake(int lLSID, int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Mode of the motor brakes: 0: Output switches to GND 1: Output switches dependent on power amplifier 2: Output switches to 24V or 12V with PC variant without 24V (I/O) 3: Use as digital output, see command	
Example:	LSX.GetMotorBrake(&X, &Y, &Z, &A);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorbrake	-

LSX_SetMotorBrake			
Description:	Command for setting the motor brake output mode.		
Delphi:	function LSX_SetMotorBrake(LSID: Integer; X,Y,Z,A : Integer): Integer;		
C++:	Int SetBrake (int lLSID, int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Mode of the motor brakes: 0: Output switches to GND 1: Output switches dependent on power amplifier 2: Output switches to 24V or 12V with PC variant without 24V (I/O) 3: Use as digital output, see command	
Example:	LSX.SetMotorBrake(0,0,0,0);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorbrake	ValidConfig

LSX_GetMotorBrakeSwitchOnDelay			
Description:	Command for reading the delay between the power amplifier switch-on and the motor brake output (mode "MotorBrake 1"). Negative values delay the switch-on of the power amplifier. Positive values delay the switch-on of the output Outputs may for instance be used for switching motor brakes or door locks.		
Delphi:	function LSX_GetMotorBrakeSwitchOnDelay (LSID: Integer; X,Y,Z,A: integer): Integer;		
C++:	Int GetMotorBrakeSwitchOnDelay (int lLSID, int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Delay in ms, range: -5000 ms to 5000 ms	
Example:	LSX.GetMotorBrakeSwitchOnDelay (&X,&Y,&Z,&A);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorbrakeswitchondelay	-

LSX_SetMotorBrakeSwitchOnDelay			
Description:	Command for setting the delay between the power amplifier switch-on and the motor brake output (mode "MotorBrake 1"). Negative values delay the switch-on of the power amplifier. Positive values delay the switch-on of the output Outputs may for instance be used for switching motor brakes or door locks.		
Delphi:	function LSX_SetMotorBrakeSwitchOnDelay (LSID: Integer; X,Y,Z,A : Integer): Integer;		
C++:	Int SetBrakeSwitchOnDelay (int lLSID, int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Delay in ms, range: -5000 ms to 5000 ms	
Example:	LSX.SetMotorBrakeSwitchOnDelay (0,0,0,0);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorbrakeswitchondelay	ValidConfig

LSX_GetMotorBrakeSwitchOffDelay			
Description:	Command for reading the delay between the power amplifier switch-off and the motor brake output (mode "MotorBrake 1"). Negative values delay the switch-off of the power amplifier. Positive values delay the switch-off of the output. Outputs may for instance be used for switching motor brakes or door locks.		
Delphi:	function LSX_GetMotorBrakeSwitchOffDelay (LSID: Integer; X,Y,Z,A: integer): Integer;		
C++:	Int GetMotorBrakeSwitchOffDelay (int lLSID, int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Delay in ms, range: -5000 ms to 5000 ms	
Example:	LSX.GetMotorBrakeSwitchOffDelay (&X,&Y,&Z,&A);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorbrakeswitchoffdelay	-

LSX_SetMotorBrakeSwitchOffDelay			
Description:	Command for setting the delay between the power amplifier switch-off and the motor brake output (mode "MotorBrake 1"). Negative values delay the switch-off of the power amplifier. Positive values delay the switch-off of the output. Outputs may for instance be used for switching motor brakes or door locks.		
Delphi:	function LSX_SetMotorBrakeSwitchOffDelay (LSID: Integer; X,Y,Z,A : Integer): Integer;		
C++:	Int SetMotorBrakeSwitchOffDelay (int lLSID, int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Delay in ms, range: -5000 ms to 5000 ms	
Example:	LSX.SetMotorBrakeSwitchOffDelay (0,0,0,0);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorbrakeswitchoffdelay	ValidConfig

LSX_GetMotorAutoKommDelay			
Description:	Command for reading the delay between the power amplifier switch-on and starting the auto-commutation.		
Delphi:	function LSX_GetMotorAutoKommDelay (LSID: Integer; X,Y,Z,A: integer): Integer;		
C++:	Int GetMotorAutoKommDelay (int lLSID, int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	delay in ms, range: 0 to 5000 ms	
Example:	LSX.GetMotorAutoKommDelay (&X,&Y,&Z,&A);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorautokommdelay	-

LSX_SetMotorAutoKommDelay			
Description:	Command for setting the delay between the power amplifier switch-on and starting the auto-commutation.		
Delphi:	function LSX_SetMotorAutoKommDelay (LSID: Integer; X,Y,Z,A : Integer): Integer;		
C++:	Int SetMotorAutoKommDelay (int lLSID, int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	delay in ms, range: 0 to 5000 ms	
Example:	LSX.SetMotorAutoKommDelay (5,5,5,5);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorautokommdelay	ValidConfig

LSX_GetMotorAutoKommSpeedScale			
Description:	Command for reading the auto-commutation speed. Can be used for adjusting the movement process when auto-commutation to the motor force respectively the motor torque.		
Delphi:	function LSX_GetMotorAutoKommSpeedScale (LSID: Integer; X,Y,Z,A: double): integer;		
C++:	Int GetMotorAutoKommSpeedScale (int lLSID, double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	scaling factor, range: 0.01 to 100	
Example:	LSX.GetMotorAutoKommSpeedScale (&X,&Y,&Z,&A);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorautokommspeedscale	-

LSX_SetMotorAutoKommSpeedScale			
Description:	Command for setting the auto-commutation speed. Can be used for adjusting the movement process when auto-commutation to the motor force respectively the motor torque.		
Delphi:	function LSX_SetMotorAutoKommSpeedScale (LSID: Integer; X,Y,Z,A : double): Integer;		
C++:	Int SetMotorAutoKommSpeedScale (int lLSID, double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	scaling factor, range: 0.01 to 100	
Example:	LSX.SetMotorAutoKommSpeedScale (10.25, 10.25, 10.25, 10.25);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorautokommspeedscale	ValidConfig

LSX_GetMotorForceConstant			
Description:	Command for reading the force constant of a linear motor in [N/A]		
Delphi:	function LSX_GetMotorForceConstant (LSID: Integer; X, Y, Z, A: double): Integer;		
C++:	Int GetMotorForceConstant (int lLSID, double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	force constant, range: 0 to 500 N/A	
Example:	LSX.GetMotorForceConstant (&X, &Y, &Z, &A);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorforceconstant	-

LSX_SetMotorForceConstant			
Description:	Command for setting the force constant of a linear motor in [N/A]		
Delphi:	function LSX_SetMotorForceConstant (LSID: Integer; X, Y, Z, A: double): Integer;		
C++:	Int SetMotorForceConstant (int lLSID, double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	force constant, range: 0 to 500 N/A	
Example:	LSX.SetMotorForceConstant (2.1, 2.1, 2.1, 2.1);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorforceconstant	ValidPar / ValidConfig

LSX_GetMotorLoad			
Description:	Command for reading the motor load of axes with linear motors in [kg]. Parameter is used for servo controller rating, the entire mass (axes, mounted parts, loads, ...) is to be transmitted.		
Delphi:	function LSX_GetMotorLoad (LSID: Integer; X, Y, Z, A: double): Integer;		
C++:	Int GetMotorLoad (int lLSID, double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	motor load, range: 0 to 100 kg	
Example:	LSX.GetMotorLoad (&X, &Y, &Z, &A);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorload	-

LSX_SetMotorLoad			
Description:	Command for setting the motor load of axes with linear motors in [kg]. Parameter is used for servo controller rating, the entire mass (axes, mounted parts, loads, ...) is to be transmitted.		
Delphi:	function LSX_SetMotorLoad (LSID: Integer; X,Y,Z,A : double): Integer;		
C++:	Int SetMotorLoad (int lLSID, double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	motor load, range: 0 to 100 kg	
Example:	LSX.SetMotorLoad (5.1, 2.4, 3.0, 4.2);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorload	ValidPar / ValidConfig

LSX_GetMotorMomentConstant			
Description:	Command for reading the moment constant in [N/A]		
Delphi:	function LSX_GetMotorMomentConstant (LSID: Integer; X,Y,Z,A: double): Integer;		
C++:	Int GetMotorMomentConstant (int lLSID, double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	moment constant, range: 0 to 50 Nm/A	
Example:	LSX.GetMotorMomentConstant (&X,&Y,&Z,&A);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motormomentconstant	-

LSX_SetMotorMomentConstant			
Description:	Command for setting the moment constant in [N/A]		
Delphi:	function LSX_SetMotorMomentConstant (LSID: Integer; X,Y,Z,A : double): Integer;		
C++:	Int SetMotorMomentConstant (int lLSID, double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	moment constant, range: 0 to 50 Nm/A	
Example:	LSX.SetMotorMomentConstant (0.75, 0.75, 0.75, 0.75);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motormomentconstant	ValidPar / ValidConfig

LSX_GetMotorMomentOfInertia			
Description:	Command for reading the moment of inertia in [kgcm²] on the motor shaft. Parameter is used for servo controller rating; the sum total of all moments (rotor moment of inertia, (counted back) moment of inertia of connected mechanical devices, ...) is to be transmitted.		
Delphi:	function LSX_GetMotorMomentOfInertia (LSID: Integer; X, Y, Z, A: double): Integer;		
C++:	Int GetMotorMomentOfInertia (int lLSID, double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	moment of inertia, range: 0 to 100000 kgcm²	
Example:	LSX.GetMotorMomentOfInertia (&X, &Y, &Z, &A);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motormomentofinertia	-

LSX_SetMotorMomentOfInertia			
Description:	Command for setting the moment of inertia in [kgcm²] on the motor shaft. Parameter is used for servo controller rating; the sum total of all moments (rotor moment of inertia, (counted back) moment of inertia of connected mechanical devices, ...) is to be transmitted.		
Delphi:	function LSX_SetMotorMomentOfInertia (LSID: Integer; X, Y, Z, A: double): Integer;		
C++:	Int SetMotorMomentOfInertia (int lLSID, double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	moment of inertia, range: 0 to 100000 kgcm²	
Example:	LSX.SetMotorMomentOfInertia (0.25, 0.25, 0.25, 0.25);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motormomentofinertia	ValidPar / ValidConfig

LSX_GetMotorIITCheck			
Description:	Command for the status of the motor I²t monitoring		
Delphi:	function LSX_GetMotorIITCheck (LSID: Integer; X, Y, Z, A: LongBool): Integer;		
C++:	Int GetMotorIITCheck (int lLSID, bool *pbX, bool *pbY, bool *pbZ, bool *pbA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Status of motor I²t monitoring: True / False	
Example:	LSX.GetMotorIITCheck (&X, &Y, &Z, &A);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorisquaretcheck	-

LSX_SetMotorIITCheck			
Description:	Command for activating and deactivating the motor I²t monitoring		
Delphi:	function LSX_SetMotorIITCheck (LSID: Integer; X, Y, Z, A: LongBool): Integer;		
C++:	Int SetMotorIITCheck (int lLSID, bool bX, bool bY, bool bZ, bool bA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Status of motor I²t monitoring: True / False	
Example:	LSX.SetMotorIITCheck (False, True, False, False);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorisquaretcheck	ValidPar / ValidConfig

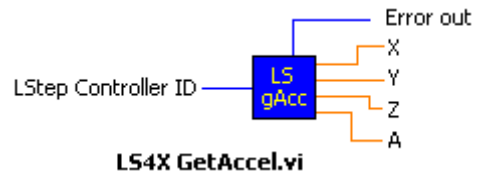
LSX_GetMotorpeakCurrentTime			
Description:	Command for reading the maximum motor peak current time in [ms]		
Delphi:	function LSX_GetMotorpeakCurrentTime (LSID: Integer; X, Y, Z, A: LongBool): Integer;		
C++:	Int GetMotorpeakCurrentTime (int lLSID, bool *pbX, bool *pbY, bool *pbZ, bool *pbA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Maximum motor peak current time, range: 0 bis 10000ms	
Example:	LSX.GetMotorpeakCurrentTime (&X, &Y, &Z, &A);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorpeakcurrenttime	-

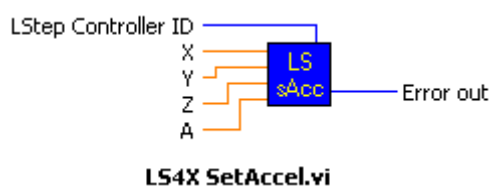
LSX_SetMotorpeakCurrentTime			
Description:	Command for setting the maximum motor peak current time in [ms]		
Delphi:	function LSX_SetMotorpeakCurrentTime (LSID: Integer; X, Y, Z, A: LongBool): Integer;		
C++:	Int SetMotorpeakCurrentTime (int lLSID, bool bX, bool bY, bool bZ, bool bA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Maximum motor peak current time, range: 0 bis 10000ms	
Example:	LSX.SetMotorpeakCurrentTime (3000, 3000, 2000, 2000);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorpeakcurrenttime	ValidPar / ValidConfig

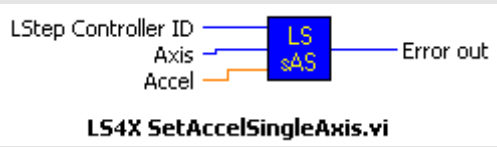
LSX_GetMotorAutoKommCurrent			
Description:	Command for reading the motor current for auto commutation (servo operation only), which has to be less than or equal to the maximum device current (see maxcur) and the motor rated current.		
Delphi:	function LSX_GetMotorAutoKommCurrent (LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	Int GetMotorAutoKommCurrent (int lLSID, double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Motor rated current or current for auto-commutation	
Example:	LSX.GetMotorAutoKommCurrent (&X, &Y, &Z, &A);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?motorautokommcurrent	-

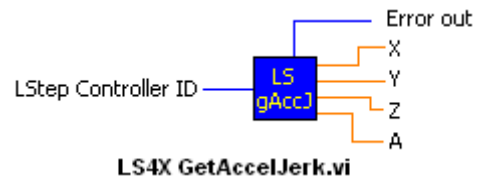
LSX_SetMotorAutoKommCurrent			
Description:	Command for setting the motor current for auto commutation (servo operation only), which has to be less than or equal to the maximum device current (see maxcur) and the motor rated current.		
Delphi:	function LSX_SetMotorAutoKommCurrent (LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	Int SetMotorAutoKommCurrent (int lLSID, double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Motor rated current or current for auto-commutation	
Example:	LSX.SetMotorAutoKommCurrent (1.5, 1.5, 1.5, 1.5);		
Others:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!motorautokommcurrent	ValidPar / ValidConfig

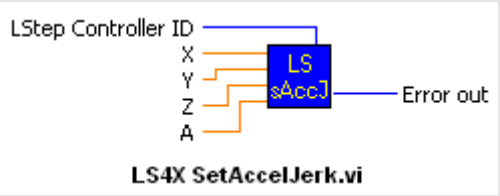
4.2.6 Kinematics

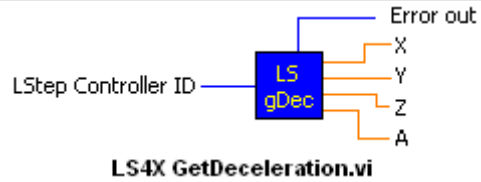
LSX_GetAccel			
Description:	Function for inquiring the acceleration for positioning processes.		
Delphi:	function LSX_GetAccel(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetAccel(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	X, Y, Z, A	Read acceleration values LSTEP 2000 series = m/s² LSTEP express series = Set dimension/s²	
Example:	LSX.GetAccel(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?Accel	-

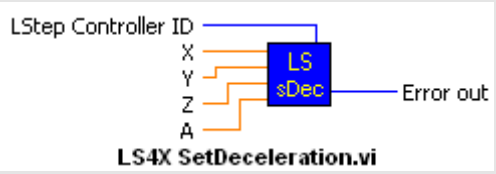
LSX_SetAccel			
Description:	Function for setting the acceleration for positioning processes.		
Delphi:	function LSX_SetAccel(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetAccel(double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	X, Y, Z, A	Axis acceleration values. LSTEP 2000 series = m/s², limited to 20 m/s² LSTEP express series = set dimension/s², maximum value limited by controller	
Example:	LSX.SetAccel(1.0, 1.5, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!accel	-

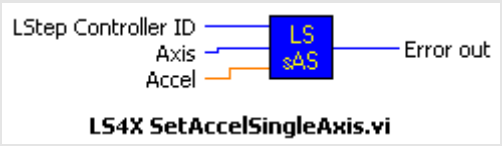
LSX_SetAccelSingleAxis			
Description:	Function for setting the acceleration of a single axis for positioning processes.		
Delphi:	function LSX_SetAccelSingleAxis(LSID: Integer; Axis: Integer; Accel: Double): Integer;		
C++:	int SetAccelSingleAxis(int lAxis,double dAccel);		
LabView:			
Parameter:	Axis	Axis whose acceleration is to be set X = 1 Y = 2 …	
	Accel	Acceleration to be adjusted LSTEP 2000 series = m/s², limited to 20 m/s² LSTEP express series = set dimension/s², maximum value limited by controller	
Example:	LSX.SetAccelSingleAxis(4, 1.0); // Accelerate A-axis 1.0 m/s²		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!accel	-

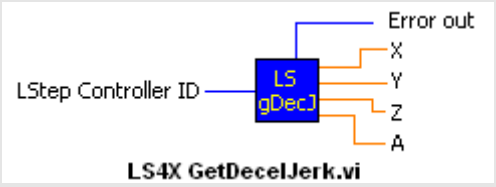
LSX_GetAccelJerk			
Description:	Function for inquiring the jerk used during the acceleration phase of positioning processes. (only for LSTEP express)		
Delphi:	function LSX_GetAccelJerk(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;		
C++:	int GetAccelJerk(double *pdXD, double *pdYD, double *pdZD, double *pdAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Jerk values in the set dimension/s3	
Example:	LSX.GetAccelJerk(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?acceljerk	-

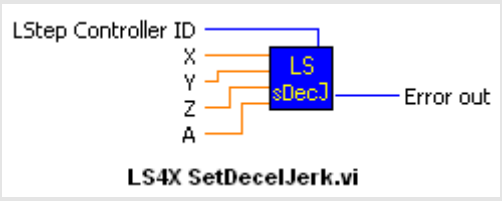
LSX_SetAccelJerk			
Description:	Function for setting the jerk used during the acceleration phase of positioning processes. (only for LSTEP express)		
Delphi:	function LSX_SetAccelJerk(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetAccelJerk(double dXD, double dYD, double dZD, double dAD);		
LabView:			
Parameter:	X, Y, Z and A	Jerk in the set dimension/s3	
Example:	LSX.SetAccelJerk(100.0, 100.5, 200, 300);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!acceljerk	-

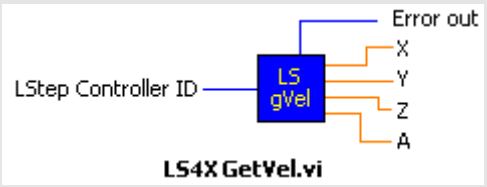
LSX_GetDeceleration			
Description:	Inquiry of deceleration applied for movements. (only for LSTEP express)		
Delphi:	function LSX_GetDeceleration(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetDeceleration(double *pdXD, double *pdYD, double *pdZD, double *pdAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Deceleration values in the set dimension/s3	
Example:	LSX.GetDeceleration(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?decel	-

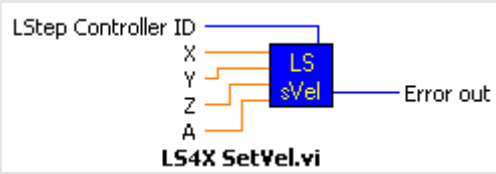
LSX_SetDeceleration			
Description:	Setting of deceleration to be applied for movements. (only for LSTEP express)		
Delphi:	function LSX_SetDeceleration(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetDeceleration(double dXD, double dYD, double dZD, double dAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Deceleration values in the set dimension/s3	
Example:	LSX.SetDeceleration(1.0, 1.5, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!decel	-

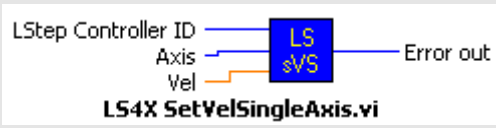
LSX_SetDecelSingleAxis			
Description:	Sets the deceleration applied for the movement of a single axis. (only for LSTEP express)		
Delphi:	function LSX_SetDecelSingleAxis(LSID: Integer; Axis: Integer; Decel: Double): Integer;		
C++:	int SetDecelSingleAxis(int lAxis,double dDecel);		
LabView:			
Parameter:	Axis	Axis whose deceleration is to be set X = 1 Y = 2 …	
	Decel	Deceleration in the set dimension/s²	
Example:	LSX.SetDecelSingleAxis(1, 1000.0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!decel	-

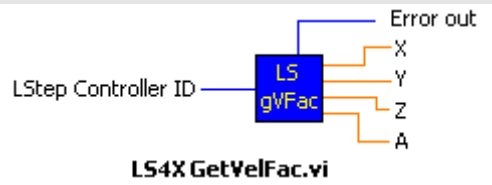
LSX_GetDecelJerk			
Description:	Inquiry of the jerk to be used during the deceleration phase of a movement. (only for LSTEP express)		
Delphi:	function LSX_GetDecelJerk(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;		
C++:	int GetDecelJerk(double *pdXD, double *pdYD, double *pdZD, double *pdAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Jerk values in the set dimension/s³	
Example:	LSX.GetDecelJerk(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?deceljerk	-

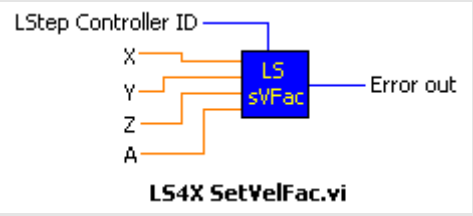
LSX_SetDecelJerk			
Description:	Setting of the jerk to be used during the deceleration phase of a movement. (only for LSTEP express)		
Delphi:	function LSX_SetDecelJerk(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetDecellJerk(double dXD, double dYD, double dZD, double dAD);		
LabView:			
Parameter:	X, Y, Z, A	Jerk in the set dimension/s³	
Example:	LSX.SetDecelJerk(1.0, 1.5, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!deceljerk	-

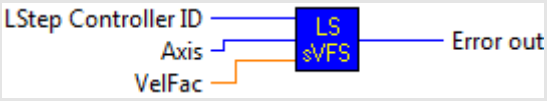
LSX_GetVel			
Description:	Inquiry of the set velocity used for positioning processes.		
Delphi:	function LSX_GetVel(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetVel(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	X, Y, Z, A	Read velocity values LSTEP 2000 series = m/s LSTEP express series = Set dimension/s	
Example:	LSX.GetVel(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?vel	-

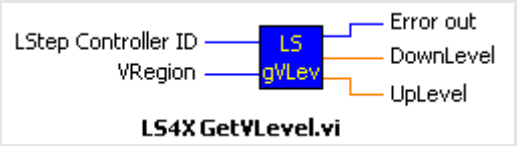
LSX_SetVel			
Description:	Setting of velocity used for positioning processes.		
Delphi:	function LSX_SetVel(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetVel(double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	X, Y, Z, A:	Velocity values to be set LSTEP 2000 series = m/s LSTEP express series = Set dimension/s	
Example:	LSX.SetVel(1.0, 15.0, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!vel	-

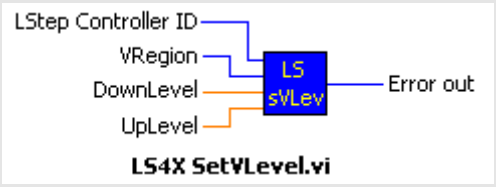
LSX_SetVelSingleAxis			
Description:	Function for setting the velocity for a single axis.		
Delphi:	function LSX_SetVelSingleAxis(LSID: Integer; Axis: Integer; Vel: Double): Integer;		
C++:	int SetVelSingleAxis(int lAxis,double dVel);		
LabView:			
Parameter:	Axis	Axis whose velocity is to be set X = 1 Y = 2 …	
	Vel	Velocity value to be set LSTEP 2000 series = m/s LSTEP express Serie = Set dimension/s	
Example:	LSX.SetVelSingleAxis(1, 10.0) // Speed of X-axis 10 rps		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!vel	-

LSX_GetVelFac			
Description:	Function for inquiring the velocity reduction (not for LSTEP express)		
Delphi:	function LSX_GetVelFac(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetVelFac(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	X, Y, Z, A:	Set factor for velocity reduction.	
Example:	LSX.GetVelFac(X, Y, Z, A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?velfac	-

LSX_SetVelFac			
Description:	Function for setting the velocity reduction. (not for LSTEP express)		
Delphi:	function LSX_SetVelFac(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetVelFac(double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	X, Y, Z, A:	Factor to be set for velocity reduction. The resulting velocity is v=Vel*VelFac.	
Example:	LSX.SetVelFac(1, 1, 0.1, 0.1);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!VelFac	-

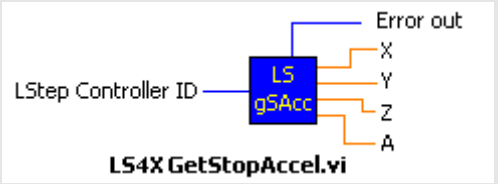
<table>
<tr><th colspan="4">LSX_SetVelFacSingleAxis</th></tr>
<tr><td>Description:</td><td colspan="3">Function for setting the velocity reduction of a single axis.
(not for LSTEP express)</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_SetVelFacSingleAxis(LSID: Integer; Axis: Integer; Value: Double): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int SetVelFacSingleAxis(int lAxis, double dValue);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="2">Parameter:</td><td>Axis</td><td colspan="2">Axis whose velocity reduction is to be set
X = 1
Y = 2
…</td></tr>
<tr><td>VelFac</td><td colspan="2">Factor to be set for velocity reduction.
The resulting velocity is v=Vel*VelFac.</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.SetVelFacSingleAxis(1, 0.1)</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>1</td><td>!velfac</td><td>-</td></tr>
</table>

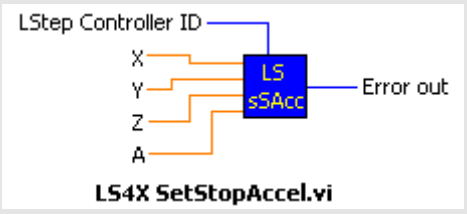
LSX_GetVLevel			
Description:	Delivers the speed limits of the indicated speed range. (not for LSTEP express)		
Delphi:	function LSX_GetVLevel(LSID: Integer; lVRegion: Integer; var dDownLevel, dUpLevel: Double): Integer;		
C++:	int GetVLevel(int lVRegion, double *pdDownLevel, double *pdUpLevel);		
LabView:			
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, the correction table is used.	
	dDownLevel	Lower limit of range (for lVRegion = 4 speed limit) in rps	
	dUpLevel	Upper limit of range (for lVRegion = 4 has no meaning) in rps	
Example:	LSX.GetVLevel(2, &DownLevel, &UpLevel); // DownLevel = Lower limit of the second speed range, UpLevel = Upper limit of the second speed range.		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?vlevel	-

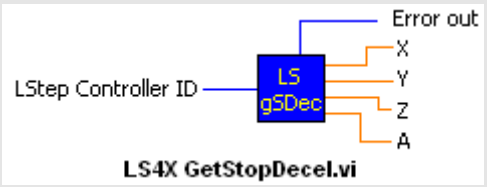
LSX_SetVLevel			
Description:	Exclude speed ranges in which the system shows resonances. (not for LSTEP express)		
Delphi:	function LSX_SetVLevel(LSID: Integer; lVRegion: Integer; dDownLevel, dUppLevel: Double): Integer;		
C++:	int SetVLevel(int lVRegion, double dDownLevel, double dUppLevel);		
LabView:			
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, the correction table is used.	
	dDownLevel	Lower limit of range (for lVRegion = 4 speed limit) in rps	
	dUppLevel	Upper limit of range (for lVRegion = 4 has no meaning) in rps	
Example:	LSX.SetVLevel(4, 10.0, 0.0); //The correction table is active up to a speed of 10 rps.		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!vlevel	-

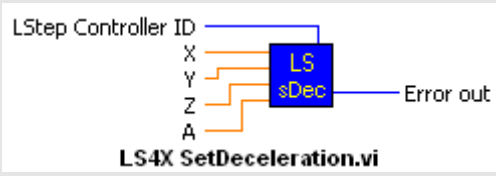
LSX_GetSpeedPoti			
Description:	Inquires whether the potentiometer is activated or deactivated.		
Delphi:	function LSX_GetSpeedPoti(LSID: Integer; var SpePoti: LongBool): Integer;		
C++:	int GetSpeedPoti(BOOL *pbSpePoti);		
LabView:			
Parameter:	SpePoti	Potentiometer status True = All power amplifiers are activated False = All power amplifiers are deactivated	
Example:	LSX.GetSpeedPoti(&flag);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?pot	-

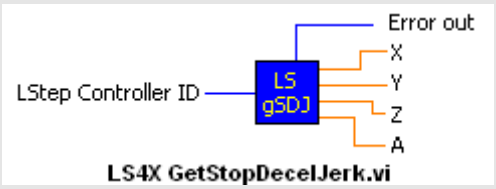
LSX_SetSpeedPoti			
Description:	Activation or deactivation of the potentiometer. If this is activated, the speed of travel is a percentage dependent on the potentiometer setting. If it is deactivated, the speed specified before is used. In manual joystick operation, potentiometer evaluation is always active.		
Delphi:	function LSX_SetSpeedPoti(LSID: Integer; SpeedPoti: LongBool): Integer;		
C++:	int SetSpeedPoti(BOOL SpeedPoti);		
LabView:			
Parameter:	SpePoti	Potentiometer status True = All power amplifiers are activated False = All power amplifiers are deactivated	
Example:	LSX.SetSpeedPoti(true);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!pot	-

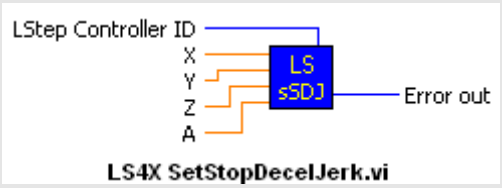
LSX_GetStopAccel			
Description:	Shows the brake acceleration, if the stop input is .active. (not for LSTEP express)		
Delphi:	function LSX_GetStopAccel(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetStopAccel(double *pdXD, double *pdYD, double *pdZD, double *pdAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Deceleration in m/s2	
Example:	LSX.GetStopAccel(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?stopaccel	-

LSX_SetStopAccel			
Description:	Function for setting the brake acceleration, if the stop input is active. This value only applies to vector operation, not for: joystick, calibration and stroke measurement. Apart from this, it is not stored with a Save in the controllers of the LSTEP 2000 series. (not for LSTEP express)		
Delphi:	function LSX_SetStopAccel(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetStopAccel(double dXD, double dYD, double dZD, double dAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Deceleration in m/s²	
Example:	LSX.SetStopAccel(1.0, 1.5, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!stopaccel	-

LSX_GetStopDecel			
Description:	Sets the brake deceleration for an active stop input. (only for LSTEP express)		
Delphi:	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	Deceleration values in the set dimension/s²	
Example:	LSX.GetStopDecel(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?stopdecel	-

LSX_SetStopDecel			
Description:	Function for setting the brake deceleration at which the axes shall brake in case of a stop signal. In order that this value be actively used, it has to be higher than the value set by LSX_SetDecelJerk. In order that this value be actively used, it has to be higher than the value set by LSX_SetDecel. (only for LSTEP express)		
Delphi:	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, AD	Deceleration values in the set dimension/s²	
Example:	LSX.SetStopDecel(1.0, 1.5, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!stopdecel	-

LSX_GetStopDecelJerk			
Description:	Function to stop the jerk used in order to build up or relieve the deceleration in case of an emergency stop signal. (only for LSTEP express)		
Delphi:	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 in the set dimension/s²	
Example:	LSX.GetStopDecelJerk(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?stopdeceljerk	-

LSX_SetStopDecelJerk			
Description:	Function to set the jerk used in order to build up or relieve the deceleration in case of an emergency stop signal. In order that this value be actively used, it has to be higher than the value set by LSX_SetDecelJerk. If it is lower, a stop with the brake jerk set by LSX_SetDecelJerk is effected. (only for LSTEP express)		
Delphi:	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, AD	Jerk values in the set dimension/s²	
Example:	LSX.SetStopDecelJerk(1000, 1500, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!stopdeceljerk	-

LSX_GetRoomAccelJerk			
Description:	Command for reading the jerk during acceleration: Jerk limitation is recommended for oscillation-critical machines. Values should be set to the required minimum for oscillation suppression, but to the possible maximum, since these parameters have a considerable effect on the positioning times. Input values depend on the dimension.		
Delphi:	function LSX_GetRoomAccelJerk (LSID: Integer; var value: Double): Integer;		
C++:	int GetRoomAccelJerk (double *pdvalue);		
LabView:	Not Supported		
Parameter:	value	Jerk during acceleration	
Example:	LSX.GetRoomAccelJerk (&value);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?roomacceljerk	-

LSX_SetRoomAccelJerk			
Description:	Command for adjusting the jerk during acceleration: Jerk limitation is recommended for oscillation-critical machines. Values should be set to the required minimum for oscillation suppression, but to the possible maximum, since these parameters have a considerable effect on the positioning times. Input values depend on the dimension.		
Delphi:	function LSX_SetRoomAccelJerk (LSID: Integer; value: Double): Integer;		
C++:	int SetRoomAccelJerk (double dvalue);		
LabView:	Not Supported		
Parameter:	value	Jerk during acceleration	
Example:	LSX.SetRoomAccelJerk (4500);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!roomacceljerk	Immediately

LSX_GetRoomDecelJerk			
Description:	Command for reading the jerk during deceleration: Jerk limitation is recommended for oscillation-critical machines. Values should be set to the required minimum for oscillation suppression, but to the possible maximum, since these parameters have a considerable effect on the positioning times. Input values depend on the dimension.		
Delphi:	function LSX_GetRoomDecelJerk (LSID: Integer; var value: Double): Integer;		
C++:	int GetRoomDecelJerk (double *pdvalue);		
LabView:	Not Supported		
Parameter:	value	Jerk during deceleration	
Example:	LSX.GetRoomDecelJerk (&value);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?roomdeceljerk	-

LSX_SetRoomDecelJerk			
Description:	Command for adjusting the jerk during deceleration: Jerk limitation is recommended for oscillation-critical machines. Values should be set to the required minimum for oscillation suppression, but to the possible maximum, since these parameters have a considerable effect on the positioning times. Input values depend on the dimension.		
Delphi:	function LSX_SetDecelJerk (LSID: Integer; value: Double): Integer;		
C++:	int SetRoomDecelJerk (double dvalue);		
LabView:	Not Supported		
Parameter:	value	Jerk during deceleration	
Example:	LSX.SetRoomDecelJerk (4800);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!roomdeceljerk	Immediately

LSX_GetRoomStopJerk			
Description:	Command for reading the jerk during braking upon occurrence of an emergency stop event. Jerk limitation is recommended for oscillation-critical machines. Values should be set to the required minimum for oscillation suppression, but to the possible maximum, since this parameter has a considerable effect on the duration of the braking process. Input values depend on the dimension. If "!stopdeceljerk" falls below "!deceljerk", the ramp is computed on basis of "!deceljerk".		
Delphi:	function LSX_GetRoomStopJerk (LSID: Integer; var value: Double): Integer;		
C++:	int GetRoomStopJerk (double *pdvalue);		
LabView:	Not Supported		
Parameter:	value	Jerk during deceleration upon occurrence of an emergency stop event	
Example:	LSX.GetRoomStopJerk (&value);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?roomstopjerk	-

LSX_SetRoomStopJerk			
Description:	Command for setting the jerk during braking upon occurrence of an emergency stop event. Jerk limitation is recommended for oscillation-critical machines. Values should be set to the required minimum for oscillation suppression, but to the possible maximum, since this parameter has a considerable effect on the duration of the braking process. Input values depend on the dimension. If "!stopdeceljerk" falls below "!deceljerk", the ramp is computed on basis of "!deceljerk".		
Delphi:	function LSX_SetStopJerk (LSID: Integer; value: Double): Integer;		
C++:	int SetRoomStopJerk (double dvalue);		
LabView:	Not Supported		
Parameter:	value	Jerk during deceleration upon occurrence of an emergency stop event	
Example:	LSX.SetRoomStopJerk (5800);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!roomstopjerk	Immediately

LSX_GetRoomAccel			
Description:	Command for reading the acceleration		
Delphi:	function LSX_GetRoomAccel (LSID: Integer; var value: Double): Integer;		
C++:	int GetRoomAccel (double *pdvalue);		
LabView:	Not Supported		
Parameter:	value	Acceleration	
Example:	LSX.GetRoomAccel (&value);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?roomaccel	-

LSX_SetRoomAccel			
Description:	Command for setting the acceleration		
Delphi:	function LSX_SetRoomAccel (LSID: Integer; value: Double): Integer;		
C++:	int SetRoomAccel (double dvalue);		
LabView:	Not Supported		
Parameter:	Value	Acceleration	
Example:	LSX.SetRoomAccel (9500);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!roomaccel	Immediately

LSX_GetRoomDecel			
Description:	Command for reading the deceleration		
Delphi:	function LSX_GetRoomDecel (LSID: Integer; var value: Double): Integer;		
C++:	int GetRoomDecel (double *pdvalue);		
LabView:	Not Supported		
Parameter:	Value	Deceleration	
Example:	LSX.GetRoomDecel (&value);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?roomdecel	-

LSX_SetRoomDecel			
Description:	Command for setting the deceleration		
Delphi:	function LSX_SetRoomDecel (LSID: Integer; value: Double): Integer;		
C++:	int SetRoomDecel (double dvalue);		
LabView:	Not Supported		
Parameter:	Value	Deceleration	
Example:	LSX.SetRoomDecel (2900);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!roomdecel	Immediately

LSX_GetRoomStopDecel			
Description:	Commands for reading the deceleration after occurrence of an emergency stop event. If !stopdecel or !roomstopdecel fall below the value of roomdecel, the braking ramp is computed on basis of their values.		
Delphi:	function LSX_GetRoomStopDecel (LSID: Integer; var value: Double): Integer;		
C++:	int GetRoomStopDecel (double *pdvalue);		
LabView:	Not Supported		
Parameter:	Value	Preset deceleration value in case of an emergency stop event	
Example:	LSX.GetRoomStopDecel (&value);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?roomstopdecel	-

LSX_SetRoomStopDecel			
Description:	Commands for setting the deceleration after occurrence of an emergency stop event. If !stopdecel or !roomstopdecel fall below the value of roomdecel, the braking ramp is computed on basis of their values.		
Delphi:	function LSX_SetRoomStopDecel (LSID: Integer; value: Double): Integer;		
C++:	int SetRoomStopDecel (double dvalue);		
LabView:	Not Supported		
Parameter:	Value	Preset deceleration value in case of an emergency stop event	
Example:	LSX.SetRoomStopDecel (2900);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!roomstopdecel	Immediately

LSX_GetRoomVel			
Description:	Commands for reading the travel velocity. The value must be lower than or equal to the value set by "!motormaxvel". The lowest velocity is possible by using the dimension microsteps and the max. step resolution of 32768 MI/pole pair.		
Delphi:	function LSX_GetRoomVel (LSID: Integer; var value: Double): Integer;		
C++:	int GetRoomVel (double *pdvalue);		
LabView:	Not Supported		
Parameter:	Value	Velocity	
Example:	LSX.GetRoomVel (&value);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?roomvel	-

LSX_SetRoomVel			
Description:	Commands for setting the travel velocity. The value must be lower than or equal to the value set by "!motormaxvel". The lowest velocity is possible by using the dimension microsteps and the max. step resolution of 32768 MI/pole pair.		
Delphi:	function LSX_SetRoomVel (LSID: Integer; value: Double): Integer;		
C++:	int SetRoomVel (double dvalue);		
LabView:	Not Supported		
Parameter:	Value	Velocity	
Example:	LSX.SetRoomVel (3200);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!roomvel	Immediataly

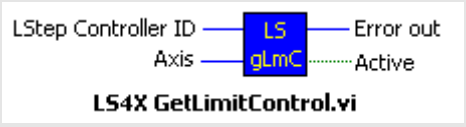
LSX_GetSpeedEnable			
Description:	Command for reading the status of the velocity or speed selection.		
Delphi:	function LSX_GetSpeedEnable (LSID: Integer; var X, Y, Z, A: LongBool): Integer;		
C++:	int GetSpeedEnable (BOOL *plX, BOOL *plY, BOOL *plZ, BOOL *plA);		
LabView:	Not Supported		
Parameter:	X, Y, Z, A	Status of the axes	
Example:	LSX.GetSpeedEnable (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?senable	-

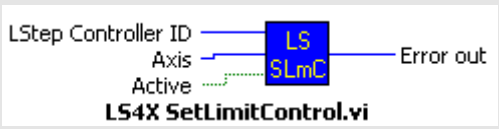
LSX_SetSpeedEnable			
Description:	Command for enabling the velocity or speed selection.		
Delphi:	function LSX_SetSpeedEnable (LSID: Integer; X, Y, Z, A: LongBool): Integer;		
C++:	int SetSpeedEnable (BOOL lX, BOOL lY, BOOL lZ, BOOL lA);		
LabView:	Not Supported		
Parameter:	X, Y, Z, A	Status of the axes	
Example:	LSX.SetSpeedEnable (1, 1, 1, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!senable	Immediately

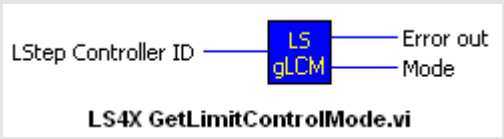
LSX_GetSpeed			
Description:	Command for reading the target velocity or speed. Value is transferred actively without ramp and with 3 decimal places. The value gets shown in hexadecimal(20 Bit signed)		
Delphi:	function LSX_GetSpeed (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetSpeed (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not Supported		
Parameter:	X, Y, Z, A	Velocity	
Example:	LSX.GetSpeed (&value);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?s	-

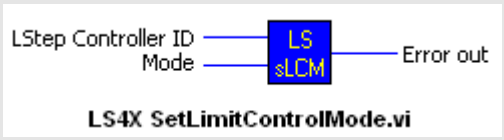
LSX_SetSpeed			
Description:	Command for setting the target velocity or speed. Value is transferred actively without ramp and with 3 decimal places. It is possible to give decimal or hexadecimal values into the function (20 Bit signed)		
Delphi:	function LSX_SetSpeed (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetSpeed (int lX, int lY, int lZ, int lA);		
LabView:	Not Supported		
Parameter:	X, Y, Z, A	Velocity	
Example:	LSX.SetSpeed (300, $12C, -300, $FFED4);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!s	SpeedEnable

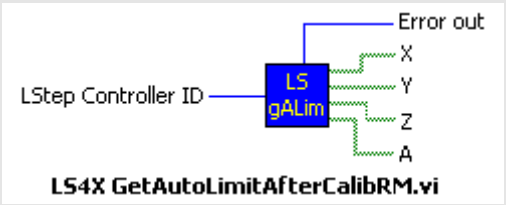
4.2.7 Limit switches and software limits

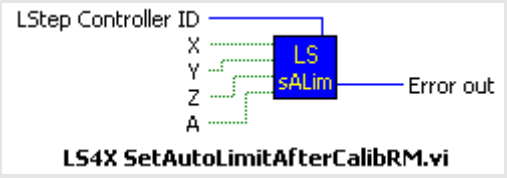
LSX_GetLimitControl			
Description:	This function reads whether the range monitoring is activated or deactivated from the controller.		
Delphi:	function LSX_GetLimitControl(LSID: Integer; Axis: Integer; var Active: Long-Bool): Integer;		
C++:	int GetLimitControl(int lAxis, BOOL *pbActive);		
LabView:			
Parameter:	Axis	Axis from where the range monitoring is to be read 1 = X-axis 2 = Y-axis …	
	Active	Set value of range monitoring True = Range monitoring is active False = Range monitoring is not active	
Example:	LSX.GetLimitControl(2, &Active); // Active = False means: Range monitoring of axis y is deactivated.		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?limctr	-

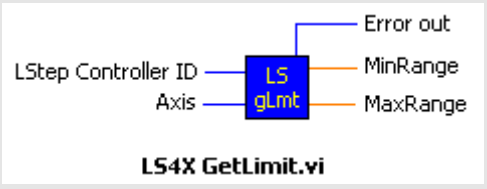
LSX_SetLimitControl			
Description:	Activates or deactivates the range monitoring of the controller.		
Delphi:	function LSX_SetLimitControl(LSID: Integer; Axis: Integer; Active: LongBool): Integer;		
C++:	int SetLimitControl(int lAxis,BOOL Active);		
LabView:			
Parameter:	Axis	Axis where the range monitoring is to be written 1 = X-axis 2 = Y-axis …	
	Active	Value of range monitoring to be set True = Range monitoring is active False = Range monitoring is not active	
Example:	LSX.SetLimitControl(2, true); // range monitoring of Y-axis is active		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!limctr	-

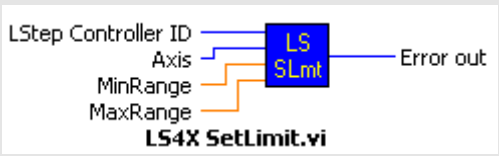
LSX_GetLimitControlMode			
Description:	Shows the mode for controlling the software limits. (not for LSTEP express)		
Delphi:	function LSX_GetLimitControlMode (LSID: Integer; var Mode: Integer): Integer;		
C++:	int GetLimitControlMode(int *plMode);		
LabView:			
Parameter:	Mode	Preset mode 0 = Moves outside of the positioning range will only be executed up to the limits of the positioning range. 1 = Moves outside the positioning range will not be executed.	
Example:	LSX.GetLimitControlMode(&lMode);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?limmode	-

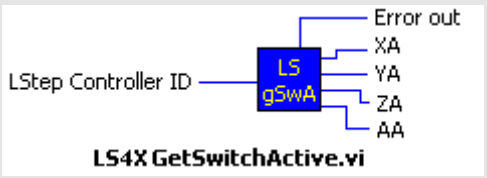
LSX_SetLimitControlMode			
Description:	Sets the mode for monitoring the software limits. (not for LSTEP express)		
Delphi:	function LSX_SetLimitControlMode (LSID: Integer; Mode: Integer): Integer;		
C++:	int SetLimitControlMode (int lMode);		
LabView:			
Parameter:	Mode	Mode to be set 0 = Moves outside of the positioning range will only be executed up to the limits of the positioning range. 1 = Moves outside the positioning range will not be executed.	
Example:	LSX.SetLimitControlMode(1);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!limmode	-

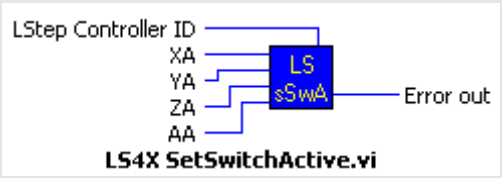
LSX_GetAutoLimitAfterCalibRM			
Description:	Indicates whether the internal software limits are set during calibration and table stroke measuring.		
Delphi:	function LSX_GetAutoLimitAfterCalibRM(LSID: Integer; var lFlags: Integer): Integer;		
C++:	int GetAutoLimitAfterCalibRM(int *plFlags);		
LabView:			
Parameter:	lFlags	32-bit integer containing a bit mask after calling the function in the bits 0-4. Bit 0 = X-axis Bit 1 = Y-axis … Value 0 = Automatic limits are used Value 1 = No automatic limits are set	
Example:	LSX.SetAutoLimitAfterCalibRM(&lFlags);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?nosetlimit	-

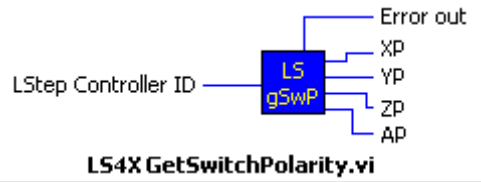
<table>
<tr><th colspan="5">LSX_SetAutoLimitAfterCalibRM</th></tr>
<tr><td>Description:</td><td colspan="4">Prevents that the internal software limits are set during calibration and table stroke measuring.</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_SetAutoLimitAfterCalibRM(LSID: Integer; lFlags: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int SetAutoLimitAfterCalibRM(int lFlags);</td></tr>
<tr><td>LabView:</td><td colspan="4"> 
</td></tr>
<tr><td>Parameter:</td><td>lFlags</td><td colspan="3">32-bit integer containing a bit mask after calling the function in the bits 0-4.
Bit 0 = X-axis
Bit 1 = Y-axis
…
Value 0 = Automatic limits are used
Value 1 = No automatic limits are set</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.SetAutoLimitAfterCalibRM(lFlags);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>!nosetlimit</td><td colspan="2">-</td></tr>
</table>

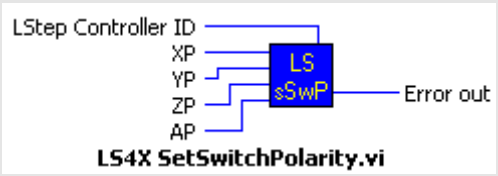
<table>
<tr><th colspan="4">LSX_GetLimit</th></tr>
<tr><td>Description:</td><td colspan="3">Reads the travel range limits of the axes.
As regards controllers of the LSTEP express series, the error code 4032 is returned for invalid range limits.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_GetLimit(LSID: Integer; Axis: Integer; var MinRange, MaxRange: Double): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int GetLimit(int lAxis, double *pdMinRange, double *pdMaxRange);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="3">Parameter:</td><td>Axis</td><td colspan="2">Axis from where the range limits are to be read.
1 = X-axis
2 = Y-axis
…</td></tr>
<tr><td>MinRange</td><td colspan="2">Lower range limit in the set axis dimension</td></tr>
<tr><td></td><td colspan="2">Upper range limit in the set axis dimension</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.GetLimit(1, &MinRange, &MaxRange);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>?lim</td><td>-</td></tr>
</table>

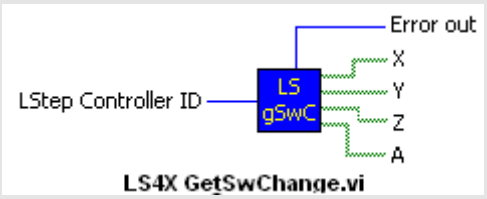
<table>
<tr><th colspan="4">LSX_SetLimit</th></tr>
<tr><td>Description:</td><td colspan="3">Sets the travel range limits of an axis.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_SetLimit(LSID: Integer; Axis: Integer; MinRange, MaxRange: Double): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int SetLimit(int lAxis,double dMinRange,double dMaxRange);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="3">Parameter:</td><td>Axis</td><td colspan="2">Axis from where the range limits are to be read.
1 = X-axis
2 = Y-axis
…</td></tr>
<tr><td>MinRange</td><td colspan="2">Lower range limit in the set axis dimension</td></tr>
<tr><td>MaxRange</td><td colspan="2">Upper range limit in the set axis dimension</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.SetLimit(1, -10.0, 20.0);
// Allocate -10 as bottom and 20 as top limit, respectively, for the X-axis</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>!lim</td><td>-</td></tr>
</table>

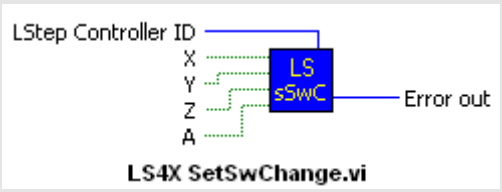
LSX_GetSwitchActive			
Description:	This function reads which limit switches were configured for monitoring.		
Delphi:	function LSX_GetSwitchActive(LSID: Integer; var XA, YA, ZA, AA: Integer): Integer;		
C++:	int GetSwitchActive(int *plXA, int *plYA, int *plZA, int *plAA);		
LabView:			
Parameter:	XA, YA, ZA, AA	Bit mask over limit switch configuration Bit 0 = Zero limit switch configuration Bit 1 = Reference limit switch configuration (always 0 for LSTEP express) Bit 2 = End limit switch configuration	
Example:	LSX.GetSwitchActive(&XA, &YA, &ZA, &AA);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?swact	-

LSX_SetSwitchActive			
Description:	Activates limit switch monitoring		
Delphi:	function LSX_SetSwitchActive(LSID: Integer; XA, YA, ZA, AA: Integer): Integer;		
C++:	int SetSwitchActive(int lXA,int lYA,int lZA,int lAA);		
LabView:			
Parameter:	XA, YA, ZA, AA	Bit mask over limit switch configuration Bit 0 = Zero limit switch configuration Bit 1 = Reference limit switch configuration (always 0 for LSTEP express) Bit 2 = End limit switch configuration	
Example:	LSX.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		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!swact	-

LSX_GetSwitchPolarity			
Description:	Function for reading the set limit switch polarity.		
Delphi:	function LSX_GetSwitchPolarity(LSID: Integer; var XP, YP, ZP, AP: Integer): Integer;		
C++:	int GetSwitchPolarity(int *plXP, int *plYP, int *plZP, int *plAP);		
LabView:			
Parameter:	XP, YP, ZP, AP	Bit mask over configured limit switch polarity Bit 0 = Zero limit switch polarity Bit 1 = Reference limit switch polarity (always 0 for LSTEP express) Bit 2 = End limit switch polarity Value 0 = Reacts on positive limit switch edge Value 1 = Reacts on negative limit switch edge	
Example:	LSX.GetSwitchPolarity(&XP, &YP, &ZP, &AP);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?swpol	-

<table>
<tr><th colspan="4">LSX_SetSwitchPolarity</th></tr>
<tr><td>Description:</td><td colspan="3">Function for setting the set limit switch polarity.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_SetSwitchPolarity(LSID: Integer; XP, YP, ZP, AP: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int SetSwitchPolarity(int lXP,int lYP,int lZP,int lAA);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>XP, YP, ZP, AP</td><td colspan="2">Bit mask over configured limit switch polarity
Bit 0 = Zero limit switch polarity
Bit 1 = Reference limit switch polarity (no effect with LSTEP express)
Bit 2 = End limit switch polarity
Value 0 = Reacts on positive limit switch edge
Value 1 = Reacts on negative limit switch edge</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.SetSwitchPolarity(7, 0, 0, 0);
// All X-axis limit switches high-active, all Y-axis limit switches low-active...</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>!swpol</td><td>-</td></tr>
</table>

LSX_GetSwChange			
Description:	Function for reading the setting limit switch change. (only for LSTEP express)		
Delphi:	function LSX_GetSwChange(LSID: Integer; var Flags: Integer): Integer;		
C++:	int GetSwChange(int *plFlags);		
LabView:			
Parameter:	Flags	32-bit integer containing a bit mask after calling the function in the bits 0-4. Bit 0 = X-axis Bit 1 = Y-axis … Value 0 = Do not change limit switch Value 1 = Change limit switch	
Example:	LSX.GetSwChange(&Flags);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?swchange	-

LSX_SetSwChange			
Description:	Function for setting limit switch change. (only for LSTEP express)		
Delphi:	function LSX_SetSwChange(LSID: Integer; Flags: Integer): Integer;		
C++:	int SetSwChange(int Flags);		
LabView:			
Parameter:	Flags	32-bit integer, with one bit mask in the bits 0-4 Bit 0 = X-axis Bit 1 = Y-axis … Value 0 = Do not change limit switch Value 1 = Change limit switch	
Example:	LSX.SetSwChange(3); /* X and Y-axis – Change limit switch (bits 1 and 1 set), Z and A-axis – No change of limit switch (bit 2 = 0) */		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!swchange	ValidConfig

LSX_GetSwitches			
Description:	This function reads the status of all limit switches.		
Delphi:	function LSX_GetSwitches(LSID: Integer; var Flags: Integer): Integer;		
C++:	int GetSwitches(int *plFlags);		
LabView:			
Parameter:	Value	Pointer to an integer value containing the status of all limit switches as a bit mask after calling the function. The limit switch statuses are encoded 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:	LSX.GetSwitches(&Flags);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?readsw	-

LSX_GetStopSwitchOffDelay			
Description:	Command for reading the delay between activating the stop input and switching off the power amplifiers and the motor brake outputs (only in stopmode 1).		
Delphi:	function LSX_GetStopSwitchOffDelay (LSID: Integer; var delay: Integer): Integer;		
C++:	int GetStopSwitchOffDelay(int *pldelay);		
LabView:	Not supported		
Parameter:	delay	delay, range: 0 to 5000ms	
Example:	LSX.GetStopSwitchOffDelay(&delay);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?stopswitchoffdelay	-

LSX_SetStopSwitchOffDelay			
Description:	Command for setting the delay between activating the stop input and switching off the power amplifiers and the motor brake outputs (only in stopmode 1).		
Delphi:	function LSX_SetStopSwitchOffDelay (LSID: Integer; delay: Integer): Integer;		
C++:	int SetStopSwitchOffDelay(int ldelay);		
LabView:	Not supported		
Parameter:	delay	delay, range: 0 to 5000ms	
Example:	LSX.SetStopSwitchOffDelay(100);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!stopswitchoffdelay	LSX_ValidConfig

LSX_GetMonitoringVelFilter			
Description:	Command for reading the time constant of actual value filter for velocity monitoring (see following commands "haltsignalvel" and "thresholdsignalvel")		
Delphi:	function LSX_GetMonitoringVelFilter (LSID: Integer; var X, Y, Z, R: Integer): Integer;		
C++:	int GetMonitoringVelFilter (int *plX, int *plY, int *plZ, int *plR);		
LabView:	Not supported		
Parameter:	X, Y, Z, R	time constant: 0 - 100 ms	
Example:	LSX.GetMonitoringVelFilter (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?monitoringvelfilter	-

LSX_SetMonitoringVelFilter			
Description:	Command for setting the time constant of actual value filter for velocity monitoring (see following commands "haltsignalvel" and "thresholdsignalvel")		
Delphi:	function LSX_SetMonitoringVelFilter (LSID: Integer; X, Y, Z, R: Integer): Integer;		
C++:	int SetMonitoringVelFilter (int lX, int lY, int lZ, int lR);		
LabView:	Not supported		
Parameter:	X, Y, Z, R	time constant: 0 - 100 ms	
Example:	LSX.SetMonitoringVelFilter (20, 50, 30, 10);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!monitoringvelfilter	LSX_ValidConfig, LSX_ValidPar

LSX_GetHaltSignalVel			
Description:	Command for reading the velocity level for the standstill signal. The standstill signal is switched off by exceeding one of the velocity levels and by a movement in one of the manual modes or started by a command. (For signal connector assignment see command "digoutlinktosignal", for control cabinet variant only: see Section 5 Connector assignment "Additional digital outputs")		
Delphi:	function LSX_GetHaltSignalVel (LSID: Integer; var XD, YD, ZD, RD: Double): Integer;		
C++:	int GetHaltSignalVel (double *pdXD, double *pdYD, double *pdZD, double *pdRD);		
LabView:	Not supported		
Parameter:	XD, YD, ZD, RD	velocity level for standstill signal	
Example:	LSX.GetHaltSignalVel (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?haltsignalvel	-

LSX_SetHaltSignalVel			
Description:	Command for setting the velocity level for the standstill signal. The standstill signal is switched off by exceeding one of the velocity levels and by a movement in one of the manual modes or started by a command. (For signal connector assignment see command "digoutlinktosignal", for control cabinet variant only: see Section 5 Connector assignment "Additional digital outputs")		
Delphi:	function LSX_SetHaltSignalVel (LSID: Integer; X, Y, Z, R: Integer): Integer;		
C++:	int SetHaltSignalVel (double dXD, double dYD, double dZD, double dRD);		
LabView:	Not supported		
Parameter:	XD, YD, ZD, RD	velocity level for standstill signal	
Example:	LSX.SetHaltSignalVel (2, 4, 3, 1);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!haltsignalvel	LSX_ValidPar

LSX_GetThresholdSignalVel			
Description:	Command for reading the value for the velocity threshold signal. The signal is only switched off by exceeding one of the velocity level values.		
Delphi:	function LSX_GetThresholdSignalVel (LSID: Integer; var XD, YD, ZD, RD: Double): Integer;		
C++:	int GetThresholdSignalVel (double *pdXD, double *pdYD, double *pdZD, double *pdRD);		
LabView:	Not supported		
Parameter:	XD, YD, ZD, RD	velocity threshold signal	
Example:	LSX.GetThresholdSignalVel (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?thresholdsignalvel	-

LSX_SetThresholdSignalVel			
Description:	Command for setting the value for the velocity threshold signal. The signal is only switched off by exceeding one of the velocity level values.		
Delphi:	function LSX_SetThresholdSignalVel (LSID: Integer; X, Y, Z, R: Integer): Integer;		
C++:	int SetThresholdSignalVel (double dXD, double dYD, double dZD, double dRD);		
LabView:	Not supported		
Parameter:	XD, YD, ZD, RD	velocity threshold signal	
Example:	LSX.SetThresholdSignalVel (2, 4, 3, 1);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!thresholdsignalvel	LSX_ValidPar

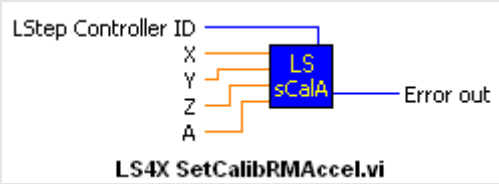
LSX_GetCSOffset			
Description:	Command for reading the coordinate system offset (position value after calibrating)		
Delphi:	function LSX_GetCSOffset (LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetCSOffset (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Length or angle of coordinate system offset	
Example:	LSX.GetCSOffset (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?csoffset	-

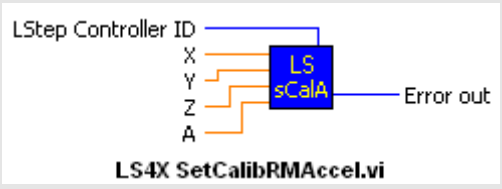
LSX_SetCSOffset			
Description:	Command for setting the coordinate system offset (position value after calibrating)		
Delphi:	function LSX_SetCSOffset (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetCSOffset (double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Length or angle of coordinate system offset	
Example:	LSX.SetCSOffset (5, 10, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!csoffset	After calibrating

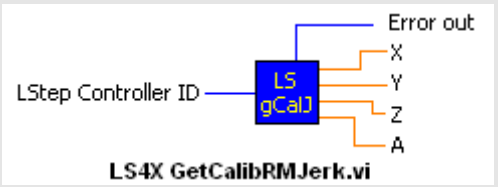
LSX_GetCalibRMOffsetSWAct			
Description:	Command for reading the status of the limit switch evaluation in offset movement during calibration and table stroke measuring. An offset is possible beyond the limit switches in the offset movement when the limit switch evaluation is deactivated.		
Delphi:	function LSX_GetCalibRMOffsetSWAct (LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetCalibRMOffsetSWAct (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Setting: On/Off	
Example:	LSX.GetCalibRMOffsetSWAct (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?calibrmoffsetswact	-

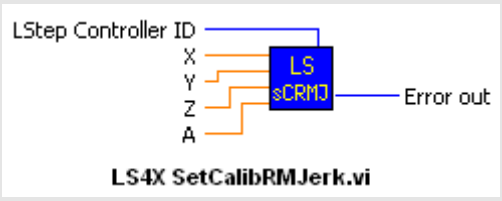
LSX_SetCalibRMOffsetSWAct			
Description:	Command for activating and deactivating the limit switch evaluation in offset movement during calibration and table stroke measuring. An offset is possible beyond the limit switches in the offset movement when the limit switch evaluation is deactivated.		
Delphi:	function LSX_SetCalibRMOffsetSWAct (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetCalibRMOffsetSWAct (double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Setting: On/Off	
Example:	LSX.SetCalibRMOffsetSWAct (False, False, True, True);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!calibrmoffsetswact	Immediately

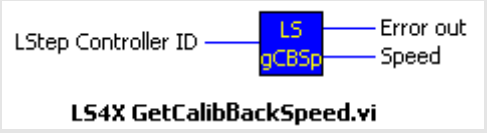
4.2.8 Reference travel

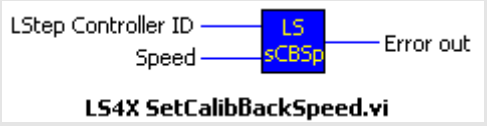
LSX_GetCalibRMAccel			
Description:	Inquires the accelera tion to be used for calibration. (only for LSTEP express)		
Delphi:	function LSX_GetCalibRMAccel(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetCalibRMAccel(double *pdXD, double *pdYD, double *pdZD, double *pdAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Acceleration values in the set dimension/s²	
Example:	LSX.GetCalibRMAccel(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?calibrmaccel	-

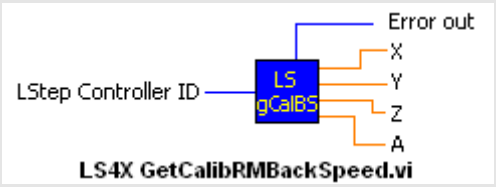
LSX_SetCalibRMAccel			
Description:	Sets acceleration for calibration process. (only for LSTEP express)		
Delphi:	function LSX_SetCalibRMAccel(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetCalibRMAccel(double dXD, double dYD, double dZD, double dAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Acceleration values in the set dimension/s²	
Example:	LSX.SetCalibRMAccel (1.0, 1.5, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!calibrmaccel	-

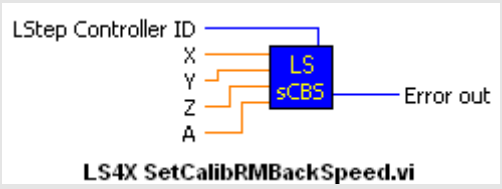
LSX_GetCalibRMJerk			
Description:	Inquiry of the jerk to be used during the calibration process. (only for LSTEP express)		
Delphi:	function LSX_GetCalibRMJerk(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;		
C++:	int GetCalibRMJerk(double *pdXD, double *pdYD, double *pdZD, double *pdAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Jerk values in the set dimension/s²	
Example:	LSX.GetCalibRMJerk(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?calibrmjerk	-

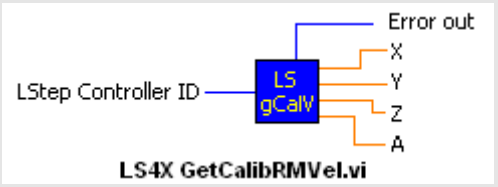
LSX_SetCalibRMJerk			
Description:	Setting of the jerk to be used during the calibration process. (only for LSTEP express)		
Delphi:	function LSX_SetCalibRMJerk(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetCalibRMJerk(double dXD, double dYD, double dZD, double dAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Jerk values in the set dimension/s²	
Example:	LSX.SetCalibRMJerk(1.0, 1.5, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!calibrmjerk	-

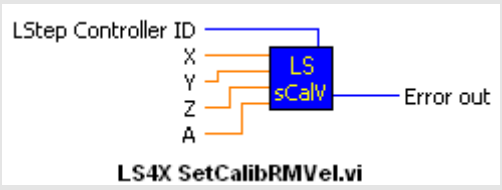
LSX_GetCalibBackSpeed			
Description:	Reads positioning speeds for leaving the limit switch during the calibration process. (only for LSTEP 2000 series. As regards LSTEP express, LS_GetCalibRMBackSpeed is to be used)		
Delphi:	function LSX_GetCalibBackSpeed(LSID: Integer; var Speed: Integer): Integer;		
C++:	Int GetCalibBackSpeed(int *plSpeed)		
LabView:			
Parameter:	Speed	Speed, equivalent to the reading value * 0.01 rps.	
Example:	LSX.GetCalibBackSpeed (&Speed);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?calbspeed	-

LSX_SetCalibBackSpeed			
Description:	Sets the positioning speeds for leaving the limit switches during the calibration process. (only for LSTEP 2000 series. As regards LSTEP express, LS_SetCalibRMBackSpeed is to be used)		
Delphi:	function LSX_SetCalibBackSpeed(LSID: Integer; Speed: Integer): Integer;		
C++:	Int SetCalibBackSpeed(int lSpeed)		
LabView:			
Parameter:	Speed	Speed, equivalent to the reading value * 0.01 rps.	
Example:	LSX.SetCalibBackSpeed (10);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!calbspeed	-

LSX_GetCalibRMBackSpeed			
Description:	Inquiry of positioning speed for leaving the limit switches to be used during the calibration process or the table stroke measuring. (only for LSTEP express)		
Delphi:	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 in the set dimension/s	
Example:	LSX.GetCalibRMBackSpeed(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?calibrmbspeed	-

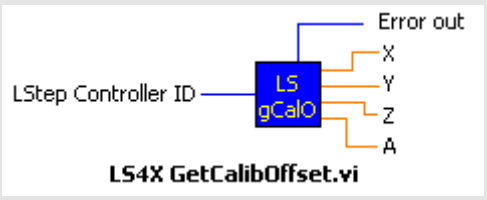
LSX_SetCalibRMBackSpeed			
Description:	Setting of positioning speeds for leaving the limit switches to be used during the calibration process or the table stroke measuring. (only for LSTEP express)		
Delphi:	function LSX_SetCalibRMBackSpeed(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetCalibRMBackSpeed(double dXD, double dYD, double dZD, double dAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Speed values in the set dimension/s	
Example:	LSX.SeCalibRMBackSpeed(1.0, 15.0, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?calibrmbspeed	-

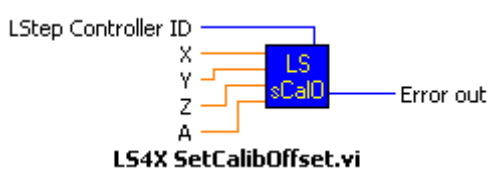
LSX_GetCalibRMVel			
Description:	Inquiry of the positioning speed to be used during the calibration process. (only for LSTEP express)		
Delphi:	function LSX_GetCalibRMVel(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;		
C++:	int GetCalibRMVel(double *pdXD, double *pdYD, double *pdZD, double *pdAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Speed values in the set dimension/s	
Example:	LSX.GetCalibRMVel(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?calibrmvel	-

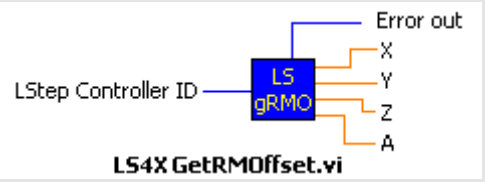
LSX_SetCalibRMVel			
Description:	Setting of the positioning speeds to be used during the calibration process. (only for LSTEP express)		
Delphi:	function LSX_SetCalibRMVel (LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetCalibRMVel(double dXD, double dYD, double dZD, double dAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Speed values in the set dimension/s	
Example:	LSX.SeCalibRMVel(1.0, 15.0, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!calibrmvel	-

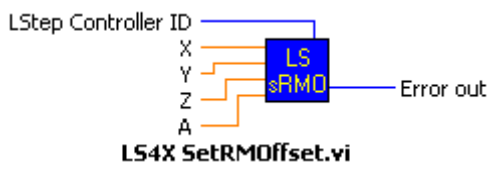
LSX_GetRefSpeed			
Description:	Reads the rotation speed at which the axes move while searching the reference mark. The speed is equivalent to the output value * 0.01 rps. (not for LSTEP express)		
Delphi:	function LSX_GetRefSpeed(LSID: Integer; var lSpeed: Integer): Integer;		
C++:	int GetRefSpeed(int *plSpeed);		
LabView:			
Parameter:	lSpeed	Speed value	
Example:	LSX.GetRefSpeed(&lSpeed);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?calrefspeed	-

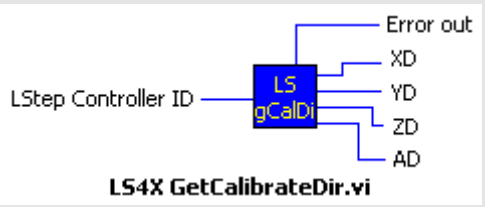
LSX_SetRefSpeed			
Description:	Sets the rotation speed at which the axes move while searching the reference mark. The speed is equivalent to the output value * 0.01 rps. (not for LSTEP express)		
Delphi:	function LSX_SetRefSpeed(LSID: Integer; lSpeed: Integer): Integer;		
C++:	int SetRefSpeed(int lSpeed);		
LabView:			
Parameter:	lSpeed	Speed	
Example:	LSX.SetRefSpeed(10); //The speed for searching the reference mark is 0.1 rps.		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!calrefspeed	-

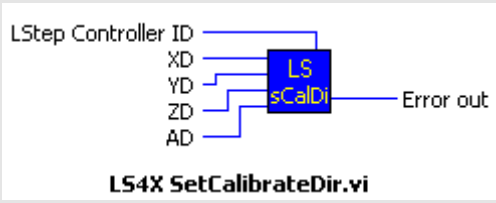
LSX_GetCalibOffset			
Description:	Function for inquiring a calibration offset controlled during calibration after moving away from the limit switch.		
Delphi:	function LSX_GetCalibOffset(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetCalibOffset (double *pdX, double *pdY, double *pdZ, double *pdR);		
LabView:			
Parameter:	X, Y, Z, A	Calibration offset in the set axis dimension	
Example:	LSX.GetCalibOffset(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?caliboffset	-

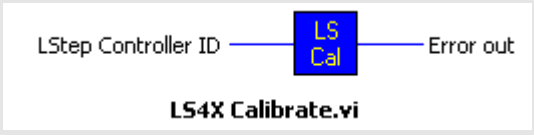
LSX_SetCalibOffset			
Description:	Function for setting a calibration offset controlled during calibration after moving away from the limit switch.		
Delphi:	function LSX_SetCalibOffset(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetCalibOffset (double dX,double dY,double dZ,double dA);		
LabView:			
Parameter:	X, Y, Z, A	Calibration offset in the set axis dimension.	
Example:	LSX.SetCalibOffset(1, 1, 1, 1);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!caliboffset	-

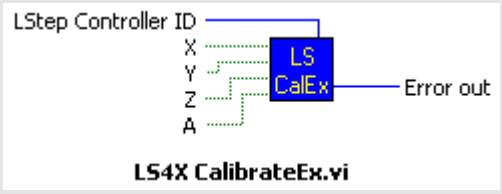
LSX_GetRMOffset			
Description:	Inquiry of the set offset for the next table stroke measurement.		
Delphi:	function LSX_GetRMOffset(LSID: Integer; varX, Y, Z, A: Double): Integer;		
C++:	int GetRMOffset(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	X, Y, Z, A	Offest for table stroke measuring in the set axis dimension	
Example:	LSX.GetRMOffset(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?rmoffset	-

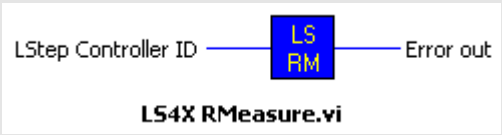
LSX_SetRMOffset			
Description:	Setting of the offset for the next table stroke measurement.		
Delphi:	function LSX_SetRMOffset(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetRMOffset (double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	X, Y, Z, A	Offest for table stroke measuring in the set axis dimension	
Example:	LSX.SetRMOffset(1.0, 1.0, 1.0, 1.0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!rmoffset	-

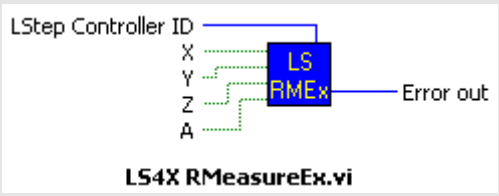
LSX_GetCalibrateDir		
Description:	Inquiry whether the sign reversal is active during calibration.	
Delphi:	function LSX_GetCalibrateDir(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;	
C++:	int GetCalibrateDir (int *plXD, int *plYD, int *plZD, int *plAD);	
LabView:		
Parameter:	XD, YD, ZD, AD	32-bit integer with indication of sign reversal. 0 = No sign reversal 1 = Sign reversal
Example:	LSX.GetCalibrateDir(&XD, &YD, &ZD, &AD);	
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?caldir	-

LSX_SetCalibrateDir			
Description:	Activation of sign reversal during calibration.		
Delphi:	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	32-bit integer with indication of sign reversal. 0 = No sign reversal 1 = Sign reversal	
Example:	LSX.SetCalibrateDir(1, 1, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!caldir	-

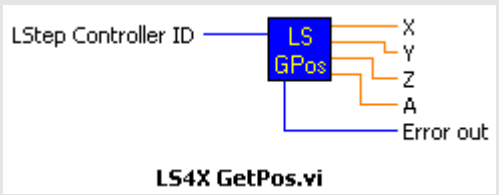
LSX_Calibrate			
Description:	This function starts the calibration routine. All enabled axes are moved towards lower position values. The movements are interrupted as soon as the limit switches are reached. The position value is set to 0.		
Delphi:	function LSX_Calibrate(LSID: Integer): Integer;		
C++:	int Calibrate();		
LabView:			
Parameter:	-		
Example:	LSX.Calibrate();		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!cal	-

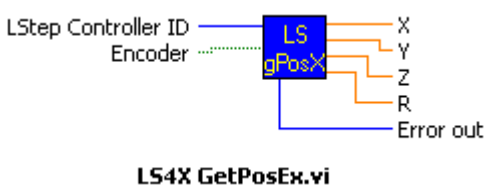
LSX_CalibrateEx			
Description:	Function for starting the calibration routine of a single axis.		
Delphi:	function LSX_CalibrateEx(LSID: Integer; Flags: Integer): Integer;		
C++:	int CalibrateEx(int lFlags);		
LabView:			
Parameter:	Flags	Bit mask Bit 0 = X-axis Bit 1 = Y-axis … Value 0 = Do not calibrate axis Value 1 = Calibrate axis	
Example:	LSX.CalibrateEx(6); // Only calibrate Y and Z-axis		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!cal	-

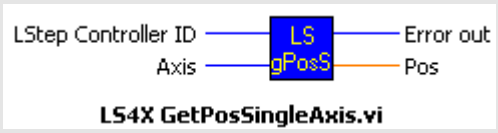
LSX_RMeasure			
Description:	Starts the table stroke measuring process. All enabled axes are moved towards higher position value. The movements are interrupted as soon as the limit switches are reached.		
Delphi:	function LSX_RMeasure(LSID: Integer): Integer;		
C++:	int RMeasure();		
LabView:			
Parameter:	-		
Example:	LSX.RMeasure();		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!rm	-

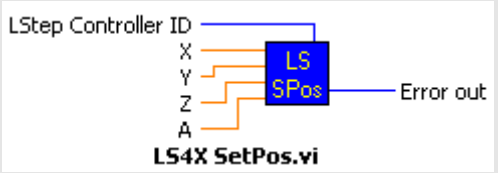
LSX_RMeasureEx			
Description:	Starts the table stroke measuring process. Only the specified axes are moved towards higher position value. The movements are interrupted as soon as the limit switches are reached.		
Delphi:	function LSX_RMeasureEx(LSID: Integer; Flags: Integer): Integer;		
C++:	int RMeasureEx (int lFlags);		
LabView:			
Parameter:	Flags	Bit mask Bit 0 = X-axis Bit 1 = Y-axis … Value 0 = Do not calibrate axis Value 1 = Calibrate axis	
Example:	LSX.RMeasureEx(2); // Measure table stroke (Y-axis only)		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!rm	-

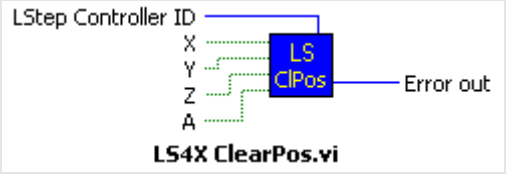
4.2.9 Travel commands and position administration

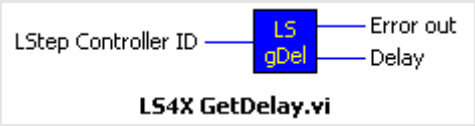
LSX_GetPos			
Description:	Inquiry of current encoder or position values of all axes. For non-existing axes, a value of 0.0 is returned.		
Delphi:	function LSX_GetPos(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetPos(double *pdX,double *pdY,double *pdZ,double *pdA);		
LabView:			
Parameter:	X, Y, Z, A	Position values	
Example:	LSX.GetPos(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?pos	-

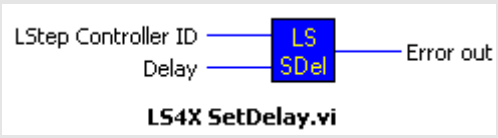
LSX_GetPosEx			
Description:	Function for inquiring the current encoder or position values of all axes. For non-existing axes, a value of 0.0 is returned. The position data source may be modified by using the encoder parameter.		
Delphi:	function LSX_GetPosEx(LSID: Integer; var X, Y, Z, R: Double; Encoder: LongBool): Integer;		
C++:	int GetPosEx(double *pdX,double *pdY,double *pdZ,double *pdA,BOOL Encoder);		
LabView:			
Parameter:	X, Y, Z, A	Position values in the set unit	
	Encoder	Read encoder value True = Show encoder values if encoder is connected False = Show position values	
Example:	LSX.GetPosEx(&X, &Y, &Z, &A, true);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!encpos ?pos	-

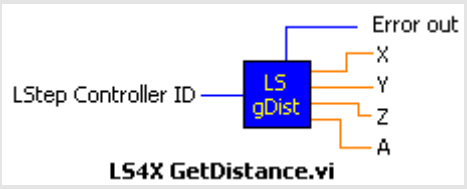
LSX_GetPosSingleAxis			
Description:	Inquiry of current position of a single axis. For non-existing axes, a value of 0.0 is returned		
Delphi:	function LSX_GetPosSingleAxis(LSID: Integer; Axis: Integer; var Pos: Double): Integer;		
C++:	int GetPosSingleAxis(int lAxis,double *pdPos);		
LabView:			
Parameter:	Axis	Axis whose position value is to be inquired X = 1 Y = 2 …	
	Pos	Position value	
Example:	LSX.GetPosSingleAxis(2, &YPos); // Read Y-axis position		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?pos	-

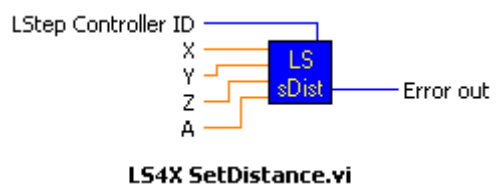
LSX_SetPos			
Description:	Function for setting a new position value. The current position is set to the transmitted one. The system zero point is shifted appropriately.		
Delphi:	function LSX_SetPos(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetPos(double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	X, Y, Z, A	Position values in the set unit of the axis	
Example:	LSX.SetPos(10, 10, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!pos	-

LSX_ClearPos			
Description:	Sets the position to Zero, including the internal counter. This function is needed for endless axes, because the controller can only process a range of +1000 motor revolutions. The function for an identified encoder is notcarried out for the relevant axis. (not for LSTEP express, see Modulo operation)		
Delphi:	function LSX_ClearPos(LSID: Integer; lFlags: Integer): Integer;		
C++:	int ClearPos (int lFlags);		
LabView:			
Parameter:	lFlags	Bit mask Bit 0 = X-axis Bit 1 = Y-axis … Value 0 = Position is not reset to zero Value 1 = Position is reset to zero	
Example:	LSX.ClearPos(5); //Positions of x and z-axes are reset to zero.		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!clearpos	-

LSX_GetDelay			
Description:	Reads the delay of the vector start. (not for LSTEP express)		
Delphi:	function LSX_GetDelay(LSID: Integer; var Delay: Integer): Integer;		
C++:	int GetDelay (int *plDelay);		
LabView:			
Parameter:	Delay	Delay in ms	
Example:	LSX.GetDelay(&Delay);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?delay	-

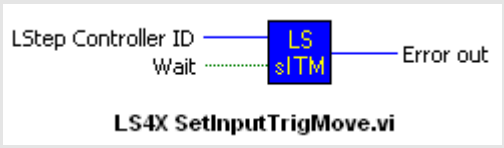
LSX_SetDelay			
Description:	The delay command is used to generate e a vector start delay. (not for LSTEP express)		
Delphi:	function LSX_SetDelay(LSID: Integer; Delay: Integer): Integer;		
C++:	int SetDelay(int lDelay);		
LabView:			
Parameter:	Delay	Delay in ms	
Example:	LSX.SetDelay(1000); // 1s delay		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!delay	-

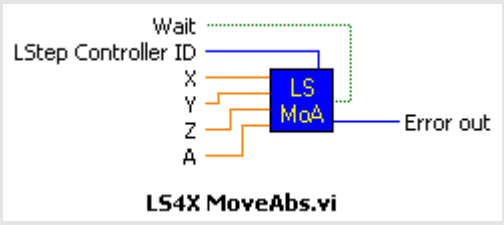
LSX_GetDistance			
Description:	Delivers the distance for LSX_MoveRelShort		
Delphi:	function LSX_GetDistance(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetDistance(double *pdX, double *pdY, double *pdZ, double *pdR);		
LabView:			
Parameter:	X, Y, Z, A	Current distance of all axes dependent on the set dimensions.	
Example:	LSX.GetDistance(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?distance	-

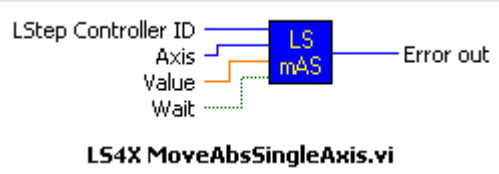
LSX_SetDistance			
Description:	Set distance for LSXMoveRelShort		
Delphi:	function LSX_SetDistance(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetDistance(double dX,double dY,double dZ,double dA);		
LabView:			
Parameter:	X, Y, Z, A	Distance to be travelled, dependent on the set axis dimension	
Example:	LSX.SetDistance(1, 2, 0, 0); /* Distances are set for the X and Y-axis, Z and A are not moved when the function LS_MoveRelShort is activated. */		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!distance	-

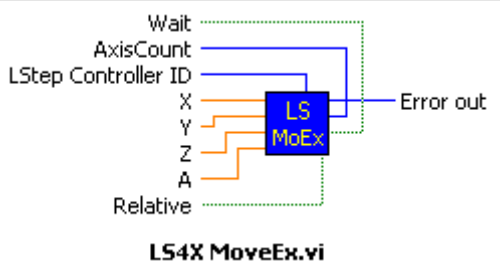
LSX_GetInputTrigMove		
Description:	Shows the configuration of Pin1 on the MFP. (not for LSTEP express)	
Delphi:	function LSX_GetInputTrigMove (LSID: Integer; var Mode: Integer): Integer;	
C++:	int GetInputTrigMove (int *plMode);	
LabView:		
Parameter:	lMode	Preset mode 0 = Function not active 1 = Absolute positioning at positive edge 2 = Absolute positioning at negative edge 3 = Relative positioning at positive edge 4 = Relative positioning at negative edge
Example:	LSX.GetInputTrigMove(&lMode);	

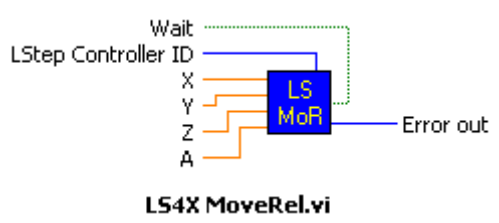
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?itm	-

<table>
<tr><th colspan="4">LSX_SetInputTrigMove</th></tr>
<tr><td>Description:</td><td colspan="3">Configures the Pin 1 on the MFP so that a move may be started with an external signal. The movement is made in accordance with the value set by function LSX_SetDistance.
(not for LSTEP express)</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_SetInputTrigMove(LSID: Integer; Mode: Integer; Wait: LongBool): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int SetInputTrigMove(int lMode, BOOL bWait);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>lMode</td><td colspan="2">Mode to be set
0 = Function not active
1 = A bsolute positioning at positive edge
2 = A bsolute positioning at negative edge
3 = Re lative positioning at positive edge
4 = R elative positioning at negative edge</td></tr>
<tr><td>Parameter:</td><td>bWait</td><td colspan="2">Wait for a move
1 = Waiting until a move is made after receiving an external signal; subsequently, the mode is set to 0.
0 = No move is awaited.
bWait is not evaluated if lMode = 0.</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.SetInputTrigMove(3, False);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>1</td><td>!itm</td><td>-</td></tr>
</table>

<table>
<tr><th colspan="5">LSX_MoveAbs</th></tr>
<tr><td>Description:</td><td colspan="4">Approaching an absolute position from the current position. The move is interpolated linearly.</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_MoveAbs(LSID: Integer; X, Y, Z, A: Double; Wait: LongBool): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int MoveAbs (double dX, double dY, double dZ, double dA, BOOL Wait);</td></tr>
<tr><td>LabView:</td><td colspan="4"> 
</td></tr>
<tr><td rowspan="2">Parameter:</td><td>X, Y, Z, A</td><td colspan="3">Absolute position in the set axis dimension.</td></tr>
<tr><td>Wait</td><td colspan="3">Wait for the end of the movement
True = Wait
False = Do not wait</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.MoveAbs(10.0, 10.0, 10.0, 10.0, true);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>!moa</td><td colspan="2">-</td></tr>
</table>

<table>
<tr><th colspan="4">LSX_MoveAbsSingleAxis</th></tr>
<tr><td>Description:</td><td colspan="3">Absolute positioning of a single axis from the current position</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_MoveAbsSingleAxis(LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int MoveAbsSingleAxis (int lAxis,double dValue,BOOL Wait);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="3">Parameter:</td><td>Axis</td><td colspan="2">Axis to be moved
X = 1
Y = 2
…</td></tr>
<tr><td>Value</td><td colspan="2">Absolute position in the set axis dimension.</td></tr>
<tr><td>Wait</td><td colspan="2">Wait for the end of the movement
True = Wait
False = Do not wait</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.MoveAbsSingleAxis(2, 10.0); // Move Y-axis to 10mm absolute position</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>!moa</td><td>-</td></tr>
</table>

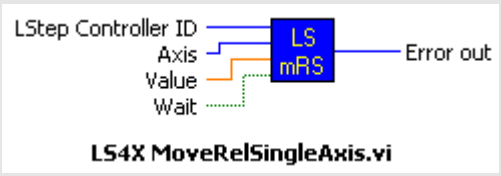
LSX_MoveEx			
Description:	The function LSX_MoveEx is an extended move command. It may carry out relative and absolute move commands, synchronously and asynchronously. The number of axes to be moved can be determined by the AxisCount parameter, please also refer to the description of the AxisCount parameter.		
Delphi:	function LSX_MoveEx(LSID: Integer; X, Y, Z, A: Double; Relative, Wait: LongBool; AxisCount: Integer): Integer;		
C++:	int MoveEx(double dX, double dY, double dZ, double dA, BOOL bRelative, BOOL bWait, int lAxisCount);		
LabView:			
Parameter:	X, Y, Z, A	Position vector in the set unit of the axis	
	Relative	Indicates whether the position vector is to be moved relatively True = Vector is moved relatively to the current position False = Vector is moved absolutely to the current position	
	Wait	Wait for end of movement True = The function only returns after reaching the target position. False = The function returns immediately after sending the command.	
	AxisCount	Number of axes to be moved 1 = X-axis on 2 = X and Y-axis 3 = X, Y and Z-axis …	
Example:	LS_MoveEx(2.0, 3.0, 0, 0, true, true, 2) ; // X and Y are moved relative by 2 or 3		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!moa / !mor	-

LSX_MoveRel			
Description:	Function for moving a relative vector from the current position.		
Delphi:	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, A	Relative position indication in the set axis dimension	
	Wait	Wait for the end of the movement True = Wait False = Do not wait	
Example:	LSX.MoveRel(10.0, 10.0, 10.0, 10.0, true);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!mor	-

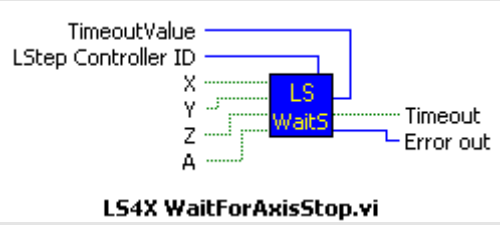
LSX_MoveRelShort			
Description:	This command should be used, so that a series of consecutive relative travel commands (of the same distance) are controlled more quickly. The distance must previously have been set with LSX_SetDistance once.		
Delphi:	function LSX_MoveRelShort(LSID: Integer): Integer;		
C++:	int MoveRelShort();		
LabView:			
Parameter:	-		

Example:	LSX.SetDistance(1.0, 1.0, 0, 0);
	for (i = 0; i < 10; i++) LSX.MoveRelShort();
	// Relative positioning of X and Y axis by 1 mm 10 times		

Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!m	-

LSX_MoveRelSingleAxis			
Description:	Relative positioning of a single axis from the current position.		
Delphi:	function LSX_MoveRelSingleAxis(LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;		
C++:	int MoveRelSingleAxis(int lAxis,double dValue,BOOL Wait);		
LabView:			
Parameter:	Axis	Axis to be moved X = 1 Y = 2 …	
	Value	Relative position in the set axis dimension.	
	Wait	Wait for the end of the movement True = Wait False = Do not wait	
Example:	LSX.MoveRelSingleAxis(3, 5.0); // Move Z-axis by 5mm in positive direction		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!mor	-

LSX_StopAxes			
Description:	Function for stopping all movements.		
Delphi:	function LSX_StopAxes(LSID: Integer): Integer;		
C++:	int StopAxes ();		
LabView:			
Parameter:	-		
Example:	LSX.StopAxes();		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!a	-

LSX_WaitForAxisStop			
Description:	The function returns as soon as the axes selected in the bit mask AFlags have reached their target position. LSX_WaitForAxisStop uses ‚?statusaxis' to poll the status of the axes.		
Delphi:	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 … Value 0 = Do not calibrate axis Value 1 = Calibrate axis	
	AtimeoutValue	Timeout in milliseconds. The value 0 deactivates the timeout function.	
	ATimeout	The ATimeout flag indicates whether a timeout has occurred. This is the case if the movement of the indicated axes is still active after expiry of the time in ATimeOutValue. True = Timeout has occurred, movement still active False = No timeout has occurred, movement completed	

Example:	LSX.WaitForAxisStop(3, 0, flag);
	// Wait until X and Y-axis have stopped, no timeout
	LSX.WaitForAxisStop(7, 10000, flag);
	// Wait until X and Y-axis have stopped, 10 seconds Timeout		

Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?statusaxis	-

LSX_GetPosWindowRange			
Description:	Shows the preset taget windows. The 'Target reached' message gets active, if the axis is within the target window for the target window time.		
Delphi:	function LSX_GetPosWindowRange(LSID: Integer,var X,Y,Z,A: Double) : Integer;		
C++:	int GetPosWindowRange (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Preset target window	
Example:	LSX.GetPosWindowRange(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?poswindowrange	-

LSX_SetPosWindowRange			
Description:	Sets the target window. The 'Target reached' message gets active, if the axis is within the target window for the target window time.		
Delphi:	function LSX_SetPosWindowRange(LSID: Integer; X,Y,Z,A : double): Integer;		
C++:	int SetPosWindowRange (double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Preset target window	
Example:	LSX.SetPosWindowRange(0.01,0.02,0.03,0.04);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!poswindowrange	-

LSX_GetPosWindowTime			
Description:	Shows the target window time. The 'Target reached' message gets active, if the axis is within the target window for the target window time.		
Delphi:	function LSX_GetPosWindowTime(LSID: Integer;var X,Y,Z,A: Integer): Integer;		
C++:	int GetPosWindowTime (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Target window time	
Example:	LSX.GetPosWindowTime(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?poswindowtime	-

LSX_SetPosWindowTime			
Description:	Sets the target window time. The 'Target reached' message gets active, if the axis is within the target window for the target window time.		
Delphi:	function LSX_SetPosWindowTime(LSID: Integer; X,Y,Z,A : integer): Integer;		
C++:	int SetPosWindowTime(int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Target window time	
Example:	LSX.SetPosWindowTime(100,100,100,100);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!poswindowtime	-

LSX_GetPosWindowTimeout			
Description:	Shows the timeout of all axes. The timeout starts after the adjustment (after completion of the travel movment) in the target window was aborted by an error message. The timeout has to be selected longer or equal to the target window time.		
Delphi:	function LSX_GetPosWindowTimeout(LSID: Integer;var X,Y,Z,A: Integer): Integer;		
C++:	int GetPosWindowTimeout (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Preset timeout	
Example:	LSX.GetPosWindowTimeout(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?poswindowtimeout	-

LSX_SetPosWindowTimeout			
Description:	Sets the timeout of all axes. The timeout starts after the adjustment (after completion of the travel movment) in the target window was aborted by an error message. The timeout has to be selected longer or equal to the target window time.		
Delphi:	function LSX_SetPosWindowTimeout(LSID: Integer; X,Y,Z,A : Integer): Integer;		
C++:	int SetPosWindowTimeout(int lX, int lY, int lZ, int lA) ;		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Preset Timeout	
Example:	LSX.SetPosWindowRange(800,800,800,800);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!poswindowtimeout	-

LSX_GetPosWindowCheck			
Description:	Shows if the target window check ist active		
Delphi:	function LSX_GetPosWindowCheck (LSID: Integer;var X,Y,Z,A: LongBool):Integer;		
C++:	int GetPosWindowCheck(BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Nicht unterstützt		
Parameter:	X,Y,Z,A	State oft he target window check	
Example:	LSX.GetPosWindowCheck(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?poswindowcheck	-

LSX_SetPosWindowCheck			
Description:	Activates/Deactivates the target window check		
Delphi:	function LSX_SetPosWindowCheck (LSID: Integer;X,Y,Z,A: LongBool):Integer;		
C++:	int SetPosWindowCheck (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	State oft he target window check	
Example:	LSX.SetPosWindowCheck(True,True,True,True);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!poswindowcheck	-

LSX_SetHalt			
Description:	Stops travel movments. The movment delays are applied for stopping the movment		
Delphi:	function LSX_SetHalt (LSID: Integer;X,Y,Z,R: LongBool):Integer;		
C++:	int SetHalt (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Axis mask (see command „?statusaxis") with an '@' for axes stopped by a stop. There is no feedback if no movment is active.	
Example:	LSX.SetHalt(True,True,True,True);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!halt	-

LSX_GetPosMode			
Description:	Command for reading mode of position reference value while switching position controller and powerstages on or off.		
Delphi:	function LSX_GetPosMode (LSID: Integer; var X, Y, Z, A: LongBool): Integer;		
C++:	int GetPosMode (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Mode: On / Off	
Example:	LSX.GetPosMode (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?posmode	-

LSX_SetPosMode			
Description:	Command for setting mode of position reference value while switching position controller and powerstages on or off.		
Delphi:	function LSX_SetPosMode (LSID: Integer; X, Y, Z, A: LongBool): Integer;		
C++:	int SetPosMode (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Mode: On / Off	
Example:	LSX.SetPosMode (False, True, True, False);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!posmode	Sofort

LSX_MoveAbsV, LSX_MoveAbsVW			
Description:	Approaching an absolute position. A linear interpolation is performed in consideration of the axis-specific limit values for velocity, acceleration and jerk. Input values depend on the dimension.		
Delphi:	function LSX_MoveAbsV(LSID: Integer; X: double; Y: double; Z: double; A: double; var Feedback: PAnsiChar; MaxLen: Integer): Integer; function LSX_MoveAbsVW(LSID: Integer; X: double; Y: double; Z: double; A: double; var Feedback: PWideChar; MaxLen: Integer): Integer;		
C++:	int MoveAbsV (double dX, double dY, double dZ, double dA, char *pcFeedback, int lMaxLen); int MoveAbsVW (double dX, double dY, double dZ, double dA, TCHAR *pcFeedback, int lMaxLen);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Travel range of ervery axis	
Feedback:	Feedback	For every axis after approaching: @	
Example:	LSX.MoveAbsV (2.5, 4, 8.3, 10, pcFeedback, 256);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!moav	Immediately

LSX_MoveRelV, LSX_MoveRelVW			
Description:	Positioning relative to starting position. A linear interpolation is performed in consideration of the axis-specific limit values for velocity, acceleration and jerk. Input values depend on the dimension.		
Delphi:	function LSX_MoveRelV(LSID: Integer; X: double; Y: double; Z: double; A: double; var Feedback: PAnsiChar; MaxLen: Integer): Integer; function LSX_MoveRelVW(LSID: Integer; X: double; Y: double; Z: double; A: double; var Feedback: PWideChar; MaxLen: Integer): Integer;		
C++:	int MoveRelV (double dX, double dY, double dZ, double dA, char *pcFeedback, int lMaxLen); int MoveRelVW (double dX, double dY, double dZ, double dA, TCHAR *pcFeedback, int lMaxLen);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Travel range of ervery axis	
Feedback:	Feedback	For every axis after approaching: @	
Example:	LSX.MoveRelV (2.5, 4, 8.3, 10, pcFeedback,256);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!morv	Immediately

LSX_GetAutoKomm			
Description:	Command for reading the status of the manually auto commutation for servo axes with incremental measuring system. By auto commutation, the orientation of the motor windings to the measuring system is identified by means of travel movements in order to allow for servo operation. In addition, the calculated and measured measuring system orientation are compared. For the purpose of successful auto commutation, the deviation between the calculated and the measured measuring system orientation must not exceed a maximum of ± 30% (see command "GetAutoKommResult").		
Delphi:	function LSX_GetAutoKomm (LSID: Integer; var X, Y, Z, A: LongBool): Integer;		
C++:	int GetAutoKomm (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Status: 0: no auto commutation 1: auto commutation completed	
Example:	LSX.GetAutoKomm (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?autokomm	-

LSX_SetAutoKomm, LSX_SetAutokommW			
Description:	Command for manually initiating the auto commutation for servo axes with incremental measuring system. By auto commutation, the orientation of the motor windings to the measuring system is identified by means of travel movements in order to allow for servo operation. In addition, the calculated and measured measuring system orientation are compared. For the purpose of successful auto commutation, the deviation between the calculated and the measured measuring system orientation must not exceed a maximum of ± 30% (see command "GetAutoKommResult").		
Delphi:	function LSX_SetAutoKomm(LSID: Integer; var Feedback: PAnsiChar; MaxLen: Integer): Integer; function LSX_SetAutoKommW(LSID: Integer; var Feedback: PWideChar; MaxLen: Integer): Integer;		
C++:	int SetAutoKomm (char *pcFeedback, int lMaxLen); int SetAutoKommW (TCHAR *pcFeedback, int lMaxLen);		
LabView:	Not supported		
Parameter:	Feedback	Feedback for each axis upon successful auto commutation: @	
Example:	LSX.SetAutoKomm(pcFeedback,256);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!autokomm	Immediately

LSX_GetAutoKommResult			
Description:	Command for reading the Relationship between the measured measuring system orientation and the calculated orientation from the auto commutation in [%], value range is ± 999.99 %. The optimum results corresponds to 100%. For activating servo operation, the value must be between 70% and 130%.		
Delphi:	function LSX_GetAutoKommResult (LSID: Integer; var X1, X2, Y1, Y2, Z1, Z2, A1, A2: Double): Integer;		
C++:	int GetAutoKommResult (double *pdX1, double *pdX2, double *pdY2, double *pdY2, double *pdZ1, double *pdZ2, double *pdA1, double *pdA2);		
LabView:	Not supported		
Parameter:	X1, X2, Y1, Y2, Z1, Z2, A1, A2	Relationship between the measured measuring system orientation and the calculated orientation (2 values per axis)	
Example:	LSX. GetAutoKommResult (&X1, &X2, &Y1, &Y2, &Z1, &Z2, &A1, &A2);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!autokommresult	-

LSX_GetMixedMoveAxisMode			
Description:	Reads the axis mode for a combined travel movement. The mode relationship, the associated units and the underlying functions may be looked up in the table "Combined travel movement - mode relationship".		
Delphi:	function LSX_GetMixedMoveAxisMode (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetMixedMoveAxisMode (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Axis mode: 0: linear movement 1: sine movement 2: cosine movement	
Example:	LSX.GetMixedMoveAxisMode (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?mmaxism	-

LSX_SetMixedMoveAxisMode			
Description:	Sets the axis mode for a combined travel movement. The mode relationship, the associated units and the underlying functions may be looked up in the table "Combined travel movement - mode relationship".		
Delphi:	function LSX_SetMixedMoveAxisMode (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetMixedMoveAxisMode (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Axis mode: 0: linear movement 1: sine movement 2: cosine movement	
Example:	LSX.SetMixedMoveAxisMode (2, 1, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!mmaxism	MixedMoveAbs / MixedMoveRel

LSX_MixedMoveAbs, LSX_MixedMoveAbsW			
Description:	Approaching an absolute position. A movement is performed in consideration of the axis-specific limit values for velocity, acceleration and jerk. The input values depend on the preset mode of the combined travel movement (see table "Combined travel movement - mode relationship").		
	function LSX_MixedMoveAbs(LSID: Integer; X: Double; Y: Double; Z: Double; A: Double; var Feedback: PAnsiChar; MaxLen: Integer): Integer; function LSX_MixedMoveAbsW(LSID: Integer; X: Double; Y: Double; Z: Double; A: Double; var Feedback: PWideChar; MaxLen: Integer): Integer;		
C++:	int MixedMoveAbs (double dX, double dY, double dZ, double dA, char *pcFeedback, int lMaxLen); int MixedMoveAbsW (double dX, double dY, double dZ, double dA, TCHAR *pcFeedback, int lMaxLen);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Travel range of ervery axis	
Feedback:	Feedback	For each axis after approaching: @	
Example:	LSX.MixedMoveAbs (2.5, 4, 8.3, 10, pcFeedback,256);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!mmoa	Immediately

LSX_MixedMoveRel, LSX_MixedMoveRelW			
Description:	Positioning relative to starting position. A movement is performed in consideration of the axis-specific limit values for velocity, acceleration and jerk. The input values depend on the preset mode of the combined travel movement (see table "Combined travel movement - mode relationship").		
Delphi:	function LSX_MixedMoveRel(LSID: Integer; X: Double; Y: Double; Z: Double; A: Double; var Feedback: PAnsiChar; MaxLen: Integer): Integer; LSX_MixedMoveRelW(LSID: Integer; X: Double; Y: Double; Z: Double; A: Double; var Feedback: PWideChar; MaxLen: Integer): Integer;		
C++:	int MixedMoveRel (double dX, double dY, double dZ, double dA, char *pcFeedback, int lMaxLen); int MixedMoveRelW (double dX, double dY, double dZ, double dA, TCHAR *pcFeedback, int lMaxLen);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Travel range of ervery axis	
Feedback:	Feedback	For each axis after approaching: @	
Example:	LSX.MixedMoveRel (2.5, 4, 8.3, 10, pcFeedback,256);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!mmor	Immediately

LSX_GetMixedMovePos			
Description:	Command for reading position values of combined travel movement. If mode 0 is adjusted, the position is not set or 0 is read.		
Delphi:	function LSX_GetMixedMovePos (LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetMixedMovePos (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Position values	
Example:	LSX.GetMixedMovePos (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?mmpos	-

LSX_SetMixedMovePos			
Description:	Command for setting position values of combined travel movement. If mode 0 is adjusted, the position is not set or 0 is read.		
Delphi:	function LSX_SetMixedMovePos (LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetMixedMovePos (double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Position values	
Example:	LSX.SetMixedMovePos (360, 200, 300, 200);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!mmpos	Immediately

LSX_GetMixedMoveAmpl			
Description:	Command for reading the amplitude or the scaling factor for combined travel movements. The effect of the amplitude may be looked up in table "Combined travel movement - mode relationship".		
Delphi:	function LSX_GetMixedMoveAmpl (LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetMixedMoveAmpl (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Amplitude / scaling factor	
Example:	LSX.GetMixedMoveAmpl (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?mmampl	-

LSX_SetMixedMoveAmpl			
Description:	Command for setting the amplitude or the scaling factor for combined travel movements. The effect of the amplitude may be looked up in table "Combined travel movement - mode relationship".		
Delphi:	function LSX_SetMixedMoveAmpl (LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetMixedMoveAmpl (double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Amplitude / scaling factor	
Example:	LSX.SetMixedMoveAmpl (10, 300, -9, 100);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!mmampl	MixedMoveAbs / MixedMoveRel

LSX_GetIndexTableDivider			
Description:	Command for reading the factor of a rotary axis for partial head operation. This command can only be used with the dimensions 'microsteps', 'degrees' and 'revolutions'.		
Delphi:	function LSX_GetIndexTableDivider (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetIndexTableDivider (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Preset factor of a rotary axis	
Example:	LSX.GetIndexTableDivider (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?itd	-

LSX_SetIndexTableDivider			
Description:	Command for determining the factor of a rotary axis for partial head operation. This command can only be used with the dimensions 'microsteps', 'degrees' and 'revolutions'.		
Delphi:	function LSX_SetIndexTableDivider (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetIndexTableDivider (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Preset factor of a rotary axis	
Example:	LSX.SetIndexTableDivider (50, 10, 5, 20);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!itd	Immediately

LSX_MoveIndexTable, LSX_MoveTableIndexW			
Description:	Command for approaching a position in partial head operation. The transferred value represents the multiplier for calculating the target position. This command can only be used with the dimensions 'microsteps', 'degrees' and 'revolutions'.		
Delphi:	function LSX_MoveIndexTable(LSID: Integer; X: Integer; Y: Integer; Z: Integer; A: Integer; var Feedback: PAnsiChar; MaxLen: Integer): Integer; function LSX_MoveIndexTableW(LSID: Integer; X: Integer; Y: Integer; Z: Integer; A: Integer; var Feedback: PWideChar; MaxLen: Integer): Integer;		
C++:	int MoveIndexTable (int lX, int lY, int lZ, int lA, char *pcFeedback, int lMaxLen); int MoveIndexTableW (int lX, int lY, int lZ, int lA, TCHAR *pcFeedback, int lMaxLen);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Travel range of ervery axis	
Feedback:	Feedback	For each axis after approaching: @	
Example:	LSX.MoveIndexTable (5, 4, 2, 8, pcFeedback,256);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!mita	Immediately

LSX_GetMoveSeqStatusPos			
Description:	Command for reading the status of a selected configurable movement sequence (at present, only sequence no. 1 is integrated in the controller, other sequences upon request). The selected sequence may be started by '!mss 1'.		
Delphi:	function LSX_GetMoveSeqStatusPos (LSID: Integer; var status: LongBool): Integer;		
C++:	int GetMoveSeqStatusPos (BOOL *pbstatus);		
LabView:	Not supported		
Parameter:	status	Status of the movement sequence: 0: No movement sequence selected 1: Sequence 1 (n-fold relative movement to table position) is selected	
Example:	LSX.GetMoveSeqStatusPos (&status);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?mssp	-

LSX_SetMoveSeqStatusPos			
Description:	Command for selecting a configurable movement sequence (at present, only sequence no. 1 is integrated in the controller, other sequences upon request). The selected sequence may be started by '!mss 1'.		
Delphi:	function LSX_SetMoveSeqStatusPos (LSID: Integer; status: LongBool): Integer;		
C++:	int SLSX_GetIndexTableDivideretMoveSeqStatusPos (BOOL bstatus);		
LabView:	Not supported		
Parameter:	status	Status of the movement sequence: 0: No movement sequence selected 1: Sequence 1 (n-fold relative movement to table position) is selected	
Example:	LSX.SetMoveSeqStatusPos (1);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!mssp	Immediately

Currently implemented movement sequence 1:

```
vardef
  int a = 0
  int b = 0
  int c = 0
  int d = 0
  int e = 0
endvardef
prog
  autostatus 0
  loop a
    set b = c
    for b to d step e
      mtpr b
      waitforaxisstop
      next
  endloop
  autostatus 1
  sendstatusaxis
end prog
```

LSX_GetMoveSeqStatus			
Description:	Command for reading the status of the selected movement sequence (see command 'MoveSeqStatusPos)'. The sequence can be started, stopped or processed in individual steps.		
Delphi:	function LSX_GetMoveSeqStatus (LSID: Integer; var status, line: Integer): Integer;		
C++:	int GetMoveSeqStatus (int *plstatus, int *plline);		
LabView:	Not supported		
Parameter:	Status, Line	Sequence status and next following program line: 0: Sequence inactive 1: Sequence active Next following sequence line	
Example:	LSX.GetMoveSeqStatus (&status);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?mss	-

LSX_SetMoveSeqStatus, LSX_SetMoveSeqStatusW			
Description:	Command for controlling the selected movement sequence (see command 'MoveSeqStatusPos)'. The sequence can be started, stopped or processed in individual steps.		
Delphi:	function LSX_SetMoveSeqStatus(LSID: Integer; status: Integer; var Feedback: PAnsiChar; MaxLen: Integer): Integer; function LSX_SetMoveSeqStatusW(LSID: Integer; status: Integer; var Feedback: PWideChar; MaxLen: Integer): Integer;		
C++:	int SetMoveSeqStatus (int lstatus, char *pcFeedback, int lMaxLen); int SetMoveSeqStatusW (int lstatus, TCHAR *pcFeedback, int lMaxLen);		
LabView:	Not supported		
Parameter:	Status	Status of the movement sequence: 0: Stops sequence 1: Starts sequence 2: Performs individual step	
Example:	LSX.SetMoveSeqStatus (1, pcFeedback,256);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!mss	Immediately

LSX_GetMoveSeqVar			
Description:	Command for reading variable values in order to be able to influence movement sequences in their properties. If no value is set before a program is started, default values are used. These will not initiate any movement in standard programs.		
Delphi:	function LSX_GetMoveSeqVar (LSID: Integer; var a, var b, var c, var d, var e: Double): Integer;		
C++:	int GetMoveSeqVar (double *pda, double *pdb, double *pdc, double *pdd, double *pde);		
LabView:	Not supported		
Parameter:	a, b, c, d, e	Values of all variables (a through e)	
Example:	LSX.GetMoveSeqVar (&a, &b, &c, &d, &e);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?msv	-

LSX_SetMoveSeqVar			
Description:	Command for setting variable values in order to be able to influence movement sequences in their properties. If no value is set before a program is started, default values are used. These will not initiate any movement in standard programs.		
Delphi:	function LSX_SetMoveSeqVar (LSID: Integer; a, b, c, d, e: Double): Integer;		
C++:	int SetMoveSeqVar (double da, double db, double dc, double dd, double de);		
LabView:	Not supported		
Parameter:	a, b, c, d, e	Values of all variables (a through e)	
Example:	LSX.SetMoveSeqVar (5, 0, 2, 4, 1);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!msv	MoveSeqStatus

4.2.10 Table commands

LSX_GetTablePos			
Description:	Command for reading the table position. Up to 1000 values may be saved for each axis. The table positions are approached by using the commands "MoveTable-PosAbs' and 'MoveTablePosRel'.		
Delphi:	function LSX_GetTablePos (LSID: Integer; var position: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetTablePos (int lposition; double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	Position, X, Y, Z, A	Table position and associated position values	
Example:	LSX.GetTablePos (5, &X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?tpos	-

LSX_SetTablePos			
Description:	Command for setting the table position. Up to 1000 values may be saved for each axis. The table positions are approached by using the commands "MoveTablePosAbs' and 'MoveTablePosRel'.		
Delphi:	function LSX_SetTablePos (LSID: Integer; var position: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetTablePos (int lposition, double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	Position, X, Y, Z, A	Table position and associated position values	
Example:	LSX.SetTablePos (5, 10, 7.5, 210, 25.7);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!tpos	Immediately

LSX_MoveTablePosAbs, LSX_MoveTablePosAbsW			
Description:	Absolutely approaches the position values of the individual axes as stored in the transferred table position.		
Delphi:	function LSX_MoveTablePosAbs(LSID: Integer; X: Integer; Y: Integer; Z: Integer; A: Integer; var Feedback: PAnsiChar; MaxLen: Integer): Integer; function LSX_MoveTablePosAbsW(LSID: Integer; X: Integer; Y: Integer; Z: Integer; A: Integer; var Feedback: PWideChar; MaxLen: Integer): Integer;		
C++:	int MoveTablePosAbs (int lX, int lY, int lZ, int lA, char *pcFeedback, int lMaxLen); int MoveTablePosAbsW (int lX, int lY, int lZ, int lA, TCHAR *pcFeedback, int lMaxLen);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Table position for each axis	
Feedback:	Feedback	For each axis after reaching the target position: @	
Example:	LSX.MoveTablePosAbs (5, 2, 7, 81, pcFeedback,256);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!mtpa	Immediately

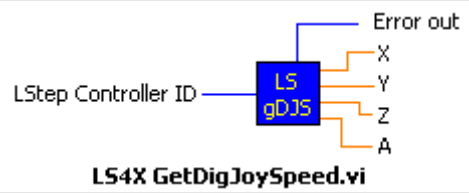
LSX_ MoveTablePosRel. LSX_MoveTablePosRelW			
Description:	Moves the axis/axes by the values stored in the transferred table position relative to the starting position.		
Delphi:	function LSX_MoveTablePosRel(LSID: Integer; X: Integer; Y: Integer; Z: Integer; A: Integer; var Feedback: PAnsiChar; MaxLen: Integer): Integer; function LSX_MoveTablePosRelW(LSID: Integer; X: Integer; Y: Integer; Z: Integer; A: Integer; var Feedback: PWideChar; MaxLen: Integer): Integer;		
C++:	int MoveTablePosRel (int lX, int lY, int lZ, int lA, char *pcFeedback, int lMaxLen); int MoveTablePosRelW (int lX, int lY, int lZ, int lA, TCHAR *pcFeedback, int lMaxLen);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Table position for each axis	
Feedback:	Feedback	For each axis after reaching the target position: @	
Example:	LSX.MoveTablePosRel (5, 2, 7, 81, pcFeedback,256);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!mtpr	Immediately

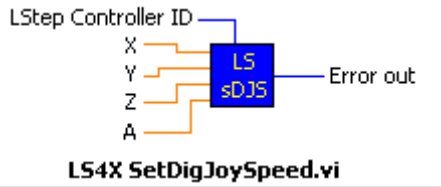
4.2.11 Joystick and handwheel

LSX_GetHandWheel			
Description:	Function for reading the handwheel status. (not for LSTEP express)		
Delphi:	function LSX_GetHandWheel(LSID: Integer; var PositionCount, Encoder: Long-Bool): Integer;		
C++:	int GetHandWheel(BOOL *pbHandWheelOn, BOOL *pbPositionCount, BOOL *pbEncoder);		
LabView:			
Parameter:	HWOn	Handwheel active True = Handwheel is activated False = Handwheel is deactivated	
	PosCount	Position counter is active True = Position counter is activated False = Position counter is deactivated	
	Encoder	Encoder values are used for position counting if available. True = Encoder values are used False = Encoder values are not used	
Example:	LSX.GetHandWheel(&HWOn, &PosCount, &Encoder);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?hw	-

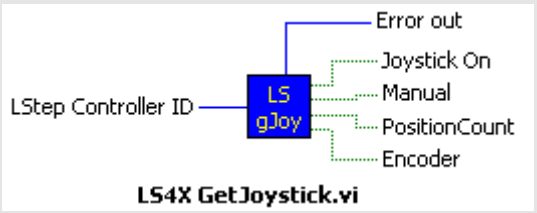
LSX_SetHandWheelOff			
Description:	Function deactivates the handwheel. (not for LSTEP express)		
Delphi:	function LSX_SetHandWheelOff(LSID: Integer): Integer;		
C++:	int SetHandWheelOff();		
LabView:			
Parameter:	-		
Example:	LSX.SetHandWheelOff();		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!hw	-

LSX_SetHandWheelOn		
Description:	Function activates the handwheel. (not for LSTEP express)	
Delphi:	function LSX_SetHandWheelOn(LSID: Integer; PositionCount, Encoder: Long-Bool): Integer;	
C++:	int SetHandWheelOn(BOOL fPositionCount,BOOL fEncoder);	
LabView:		
Parameter:	PositionCount	Activates or deactivates position counting True = On False = Off
	Encoder	Encoder values are used for position counting if available. True = Use encoder values False = Do not use encoder values
Example:	LSX.SetHandWheelOn(true, true); // Handwheel On with position counting (encoder values)	
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!hw	-

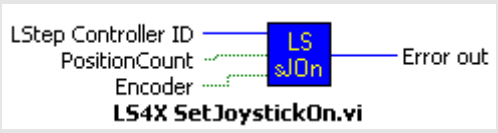
LSX_GetDigJoySpeed			
Description:	Reading of set speed for moving at a constant speed (not for LSTEP express)		
Delphi:	function LSX_GetDigJoySpeed(LSID: Integer; var dX, dY, dZ, dA: Double): Integer;		
C++:	int GetDigJoySpeed(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	dX, dY, dZ, dA	Speed values in rps	
Example:	LSX.GetDigJoySpeed(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?speed	-

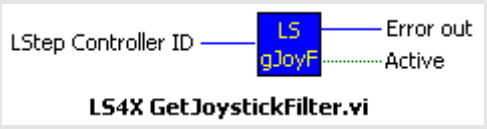
LSX_SetDigJoySpeed			
Description:	This command is used to move single axes at constant speed. If absolute or relative positioning is requested after carrying out the function, the digital joystick may be deactivate by using the function LSX_SetDigJoyOff. (not for LSTEP express)		
Delphi:	function LSX_SetDigJoySpeed(LSID: Integer; dX, dY, dZ, dA: Double): Integer;		
C++:	int SetDigJoySpeed (double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	dX, dY, dZ, dA	Speed in rps	
Example:	LSX.SetDigJoySpeed(0, 10.0, 25.0, 0); //axes X and A – speed 0 and joystick operation "OFF", axis Y – speed 10.0 rps and joystick operation "ON", axis Z – speed 25.0 rps and joystick operation "ON".		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?speed	-

LSX_SetDigJoyOff			
Description:	Deactivates the digital joystick. (not for LSTEP express)		
Delphi:	function LSX_SetDigJoyOff(LSID: Integer): Integer;		
C++:	int SetDigJoyOff() ;		
LabView:			
Parameter:	-		
Example:	LSX.SetDigJoyOff() ;		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!speed	-

<table>
<tr><th colspan="4">LSX_GetJoystick</th></tr>
<tr><td>Description:</td><td colspan="3">Inquiry of the current condition of the analogue joystick.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_GetJoystick(LSID: Integer; var JoystickOn, Manual, PositionCount, Encoder: LongBool): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int GetJoystick(BOOL *pbJoystickOn, BOOL *pbManual, BOOL *pbPositionCount, BOOL *pbEncoder);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="4">Parameter:</td><td>JoyOn</td><td colspan="2">Joystick is active
True = Joystick is activated
False = Joystick is deactivated</td></tr>
<tr><td>Manual</td><td colspan="2">Activation type of manual mode (not for LSTEP express)
True = Joystick has been activated manually by switch
False = Joystick is set to Automatic</td></tr>
<tr><td>PosCount</td><td colspan="2">Position counter activated (not for LSTEP express)
True = Position counter is activated
False = Position counter is deactivated</td></tr>
<tr><td>Enc</td><td colspan="2">Encoder values are used for position counting if available
(not for LSTEP express).
True = Encoder values are used
False = Encoder values are not used</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.GetJoystick(&JoyOn, &Manual, &PosCount, &Enc);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>?joy</td><td>-</td></tr>
</table>

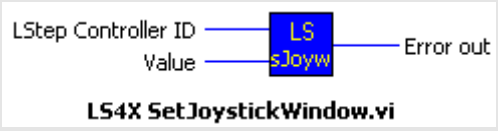
LSX_SetJoystickOff			
Description:	Function deactivates the analogue joystick.		
Delphi:	function LSX_SetJoystickOff(LSID: Integer): Integer;		
C++:	int SetJoystickOff();		
LabView:			
Parameter:	-		
Example:	LSX.SetJoystickOff();		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!joy	-

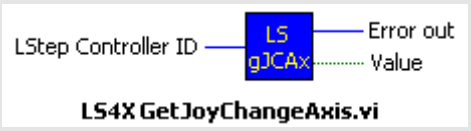
LSX_SetJoystickOn		
Description:	Function activates the analogue joystick.	
Delphi:	function LSX_SetJoystickOn(LSID: Integer; PositionCount, Encoder: LongBool): Integer;	
C++:	int SetJoystickOn(BOOL PositionCount,BOOL Encoder);	
LabView:		
Parameter:	PositionCount	Activates or deactivates position counting. True = On False = Off
	Encoder	Encoder values are used for position counting if available. (not for LSTEP express) True = Use encoder values False = Do not use encoder values
Example:	LSX.SetJoystickOn(true, true); // Joystick On with position counting (encoder values)	
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!joy	-

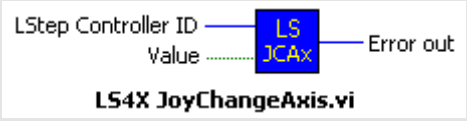
LSX_GetJoystickFilter			
Description:	Indicates whether filtering and hysteresis are activated in joystick operation. (not for LSTEP express)		
Delphi:	function LSX_GetJoystickFilter(LSID: Integer; var bActive: LongBool): Integer;		
C++:	int GetJoystickFilter(BOOL *pbActive);		
LabView:			
Parameter:	bActive	Currently set value True = Filtering is activated False = Filtering is deactivated	
Example:	LSX.GetJoystickFilter(&Active);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?joyfilter	-

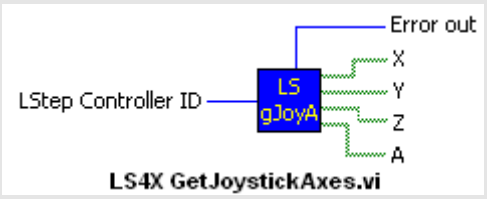
LSX_SetJoystickFilter			
Description:	Activation/Deactivation of filtering and hysteresis in joystick operation. (not for LSTEP express)		
Delphi:	function LSX_SetJoystickFilter(LSID: Integer; bActive: LongBool): Integer;		
C++:	int SetJoystickFilter(BOOL bActive);		
LabView:			
Parameter:	bActive	Filter activation True = Activate filter False = Deactivate filter	
Example:	LSX.SetJoystickFilter(True);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!joyfilter	-

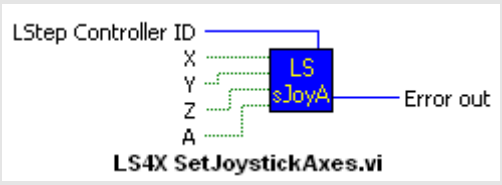
LSX_GetJoystickWindow			
Description	Function for reading the joystick window. The joystick window defines and analogous range in which the axes do not move.		
Delphi	function LSX_GetJoystickWindow(LSID: Integer; var AValue: Integer): Integer;		
C++	int GetJoystickWindow(int *plAValue);		
LabView:			
Parameter	AValue	Analogous range in which the axes do not move.	
Example	LSX.GetJoystickWindow(&AValue) ;		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?joywindow	-

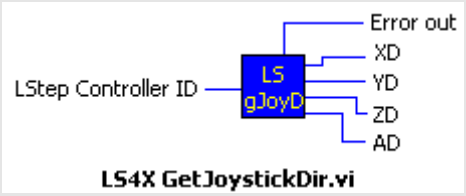
LSX_SetJoystickWindow			
Description	Function for setting the joystick window. The joystick window defines and analogous range in which the axes do not move.		
Delphi	function LSX_SetJoystickWindow(LSID: Integer; AValue: Integer): Integer;		
C++	int SetJoystickWindow(int lAValue);		
LabView:			
Parameter	AValue	Analogous range in which the axes do not move.	
Example	LSX.SetJoystickWindow(20) ;		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!joywindow	SetJoystickOn

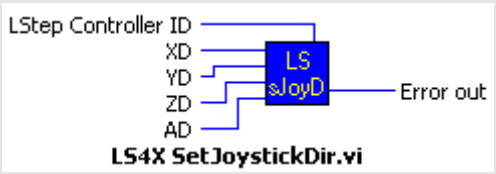
LSX_GetJoyChangeAxis			
Description:	Reads joystick axis allocation (not for LSTEP express)		
Delphi:	LSX_GetJoyChangeAxis(LSID: Integer; var Value: LongBool): Integer;		
C++:	int GetJoyChangeAxis(BOOL *pbValue);		
LabView:			
Parameter:	Value	Axis allocation True = Conventional joystick evaluation False = Allocation of X and Y axes is exchanged	
Example:	LSX.GetJoyChangeAxis(&Value);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?joychangeaxis	-

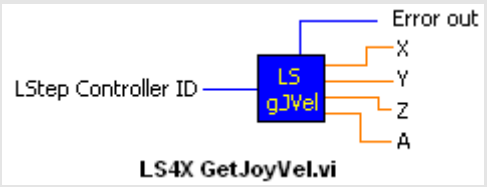
LSX_JoyChangeAxis			
Description:	Sets joystick axis allocation (not for LSTEP express)		
Delphi:	LSX_JoyChangeAxis(LSID: Integer; Value: LongBool): Integer;		
C++:	int JoyChangeAxis(BOOL bValue);		
LabView:			
Parameter:	Value	Axis allocation True = Conventional joystick evaluation False = Allocation of X and Y axes is exchanged	
Example:	LSX.JoyChangeAxis(true);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!joychangeaxis	-

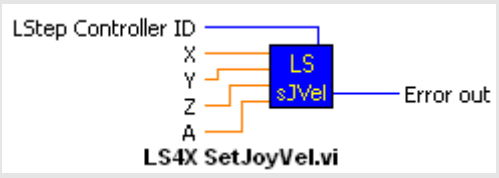
<table>
<tr><th colspan="5">LSX_GetJoystickAxes</th></tr>
<tr><td>Description:</td><td colspan="4">Shows the axis for which the joystick is active if joystick operation is activated. (only for LSTEP express)</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_GetJoystickAxes(LSID: Integer; var Flags: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int GetJoystickAxes(int *plFlags);</td></tr>
<tr><td>LabView:</td><td colspan="4"> 
</td></tr>
<tr><td>Parameter:</td><td>Flags</td><td colspan="3">Integer containing the bit mask in bits 0-4 after calling the function.
Bit 0 = X-axis
Bit 1 = Y-axis
…
Value 0 = Joystick remains deactivated
Value 1 = Joystick is activated</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.GetJoystickAxes(&Flags);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>2</td><td>?joyenable</td><td colspan="2">-</td></tr>
</table>

<table>
<tr><th colspan="4">LSX_SetJoystickAxes</th></tr>
<tr><td>Description:</td><td colspan="3">Allows joystick operation for the indicated axes.
(only for LSTEP express)</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_SetJoystickAxes(LSID: Integer; Flags: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int SetJoystickAxes(int Flags);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>Flags</td><td colspan="2">Bit mask with joystick axes to be activated when the joystick is enabled.
Bit 0 = X-axis
Bit 1 = Y-axis
…
Value 0 = Joystick remains deactivated
Value 1 = Joystick is activated</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.SetJoystickAxes(3);
/* X and Y-axis – joystick enabled (bits 0 and 1 set), Z and A-axis – joystick “OFF” (Bit 2 = 0) */</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>2</td><td>SetJoystickOn / SetJoy-stickOff</td><td>SetJoystickOn</td></tr>
</table>

<table>
<tr><th colspan="4">LSX_GetJoystickDir</th></tr>
<tr><td>Description:</td><td colspan="3">Reads out the direction of rotation of the motor for joystick.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_GetJoystickDir(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int GetJoystickDir(int *plXD, int *plYD, int *plZD, int *plRD);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>X, Y, Z, A</td><td colspan="2">Rotation direction of the motor
LSTEP 2000 series:
0 = Axis disabled
1 = Positive direction of rotation
-1 = Negative direction of rotation
2 = Positive direction of rotation with current reduction
-2 = Negative direction of rotation with current reduction
LSTEP express series:
0 = Normal direction of rotation
1 = Reverse direction of rotation</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.GetJoystickDir(&X, &Y, &Z, &A);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>?joydir</td><td>-</td></tr>
</table>

<table>
<tr><th colspan="5">LSX_SetJoystickDir</th></tr>
<tr><td>Description:</td><td colspan="4">Set joystick direction of rotation.</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_SetJoystickDir(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int SetJoystickDir(int lXD,int lYD,int lZD,int lAD);</td></tr>
<tr><td>LabView:</td><td colspan="4"> 
</td></tr>
<tr><td>Parameter:</td><td>X, Y, Z, A</td><td colspan="3">Rotation direction of the motor
LSTEP 2000 series:
0 = Axis disabled
1 = Positive direction of rotation
-1 = N egative direction of rotation
2 = Positive direction of rotation with current reduction
-2 = Negative direction of rotation with current reduction
LSTEP express series:
0 = Normal direction of rotation
1 = Reverse direction of rotation</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.SetJoystickDir(1, 1, -1, 0);
/* X and Y-axis have positive direction of rotation; Z-axis has negative direction of rotation; A-axis is disabled */</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>!joydir</td><td colspan="2">SetJoystickOn</td></tr>
</table>

LSX_GetJoyVel			
Description:	Inquiry of the maximum positioning speeds in joystick operation. (only for LSTEP express)		
Delphi:	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 in the set dimension/s	
Example:	LSX.GetJoyVel(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?joyvel	-

LSX_SetJoyVel			
Description:	Setting of the maximum positioning speeds in joystick operation. (only for LSTEP express)		
Delphi:	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, AD	Speed values in the set dimension/s	
Example:	LSX.SetJoyVel(1.0, 15.0, 0, 0);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!joyvel	SetJoystickOn

LSX_GetJoyRedcur			
Description:	Shows if the current reduction in joystick operation is active. When the current reduction is active, the motor current is reduced to the value set with "reduction" when the joystick is in its off-position.		
Delphi:	function LSX_GetJoyRedCur (LSID: Integer;var X,Y,Z,A: LongBool):Integer;		
C++:	int GetJoyRedCur (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	x,y,z,a	State oft he current reduction	
Example:	LSX.GetJoyRedCur(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?joyredcur	-

LSX_SetJoyRedcur			
Description:	Activates/Deactivates the current reduction in joystick operation. When the current reduction is active, the motor current is reduced to the value set with "reduction" when the joystick is in its off-position.		
Delphi:	function LSX_SetJoyRedCur (LSID: Integer;X,Y,Z,R: LongBool):Integer; function LSX_SetJoyRedCur (LSID: Integer; Axis:Integer; Reduce:Boolean): Integer;		
C++:	int SetJoyRedCur (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not supported		
Parameter:	x,y,z,a	State of the current reduction	
Example:	LSX.SetJoyRedCur(True,True,True,True);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!joyredcur	SetJoystickOn

LSX_GetJoytoAxis			
Description:	Shows the assigned joystick inputs 1 trough 4 to the mechanical axes X,Y,Z,A.		
Delphi:	function LSX_GetJoytoAxis(LSID: Integer; var X,Y,Z,A: Integer): Integer;		
C++:	int GetJoytoAxis (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	x,y,z,a	Assigned Joystick input	
Example:	LSX.GetJoytoAxis(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?joytoaxis	-

LSX_SetJoytoAxis			
Description:	Assigns the joystick inputs 1 trough 4 to the mechanical axes X,Y,Z,A.		
Delphi:	function LSX_SetJoytoAxis(LSID: Integer; X,Y,Z,A: Integer): Integer;		
C++:	int SetJoytoAxis (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	x,y,z,a	Assigned Joystick input	
Example:	LSX.SetJoytoAxis (1,2,3,4);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!joytoaxis	SetJoystickOn

LSX_GetManModePreselection			
Description:	Shows the requested manual mode upon activation by means of a digital input.		
Delphi:	function LSX_GetManModePreselection(LSID: Integer;var MMode: Integer): Integer;		
C++:	int GetManModePreselection (int *plMMode);		
LabView:	Not supported		
Parameter:	0 → No manual mode		
	1 → Joystick mode		
	2 → Inching operation		
	3 → Trackball		
Example:	LSX.GetManmodePreselection(&MMode) ;		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?manmodepreselection	-

LSX_SetManModePreselection			
Description:	Selects the requested manual mode upon activation by menas of digital input.		
Delphi:	function LSX_SetManModePreSelection(LSID:Integer;MMode: Integer): Integer;		
C++:	int SetManModePreselection (int lMMode);		
LabView:	Not supported		
Parameter:	0 → No manual mode		
	1 → Joystick mode		
	2 → Inching operation		
	3 → Trackball		
Example:	LSX.SetManModePreselection(0) ;		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	! manmodepreselection	-

LSX_GetManModeLinktoAxis			
Description:	Shows the link for the manual operation of one axis to another axis. The linked axis is moved in the manual mode according to the parameters (for manual mode) of the axis to which it was linked. Multiple linking is not permissible and is automatically cancelled upon re- allocation. Both axes are stopped if one of the axes involved reaches its travel range limit. Linking servo axes to stepping motor axes and vice versa has not been implemented.		

Delphi:	function LSX_GetManModeLinktoAxis (LSID: Integer;Axis: Integer;var Link2: AnsiChar): Integer;
	function LSX_GetManModeLinktoAxisX(LSID: Integer;var Link2: AnsiChar): Integer;
	function LSX_GetManModeLinktoAxisY(LSID: Integer;var Link2: AnsiChar): Integer;
	function LSX_GetManModeLinktoAxisZ(LSID: Integer;var Link2: AnsiChar): Integer;
	function LSX_GetManModeLinktoAxisA(LSID: Integer;var Link2: AnsiChar): Integer;		

C++:	int GetManModeLinktoAxis (int lAxis, char *pcLink2); int GetManModeLinktoAxisX (char *pcLink2); int GetManModeLinktoAxisY (char *pcLink2); int GetManModeLinktoAxisZ (char *pcLink2); int GetManModeLinktoAxisA (char *pcLink2);		
LabView:	Not supported		
Parameter:	Axis to be linked:		Axis to which the link is to be made:
	1=X, 2=Y, 3=Z, 4=A		X Y Z A
Example:	LSX.GetManmodeLinktoAxis(&Link2) ;		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?manmodelinktoaxis	-

LSX_SetManModeLinktoAxis			
Description:	Command for linking the manual operation of one axis to another axis. The linked axis is moved in the manual mode according to the parameters (for manual mode) of the axis to which it was linked. Multiple linking is not permissible and is automatically cancelled upon re- allocation. Both axes are stopped if one of the axes involved reaches its travel range limit. Linking servo axes to stepping motor axes and vice versa has not been implemented.		
Delphi:	function LSX_SetManModeLinktoAxis(LSID: Integer; Link1,Link2: Char): Integer;		
C++:	int SetManModeLinktoAxis (char cLink1, char cLink2);		
LabView:	Not supported		
Parameter:	Axis to be linked:	Axis to which the link is to be made:	
	x,y,z,a	x,y,z,a	
Example:	LSX.SetManModeLinktoAxis (x,y) ;		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	! manmodelinktoaxis	-

LSX_GetTipp			
Description:	Shows if inching operation is active		
Delphi:	function LSX_GetTipp (LSID: Integer;var Tipp: LongBool): Integer;		
C++:	int GetTipp (BOOL *pbTipp);		
LabView:	Not supported		
Parameter:	State oft he inching operation		
Example:	LSX.GetTipp(&Tipp);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?tipp	-

LSX_SetTipp			
Description:	Activate/Deactivate inching operation.		
Delphi:	function LSX_SetTipp (LSID: Integer;Tipp: LongBool): Integer;		
C++:	int SetTipp (BOOL bTipp);		
LabView:	Not supported		
Parameter:	State oft he inching operation		
Example:	LSX.SetTipp(True);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!tipp	-

LSX_GetTippEnable			
Description:	Shows if individual axes are enabled for inching operation.		
Delphi:	function LSX_GetTippEnable (LSID: Integer;var X,Y, Z,A: LongBool): Integer;		
C++:	int GetTippEnable (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Current settings	
Example:	LSX.GetTippEnable(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?tippenable	-

LSX_SetTippEnable			
Description:	Activates/Deactivates individual axes for inching operation.		
Delphi:	function LSX_SetTippEnable (LSID: Integer;X,Y,Z,A: LongBool): Integer;		
C++:	int SetTipp (BOOL bTipp);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Current settings	
Example:	LSX.SetTippEnable(True,True,True,True);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!tippenable	SetTipp

LSX_GetTippRedCur			
Description:	Shows if the current reduction in inching operation is active. When the current reduction is active and all axes are at standstill, the motor current is reduced to the set value with "reduction".		
Delphi:	function LSX_GetTippRedCur (LSID: Integer; var X,Y, Z,A: LongBool): Integer;		
C++:	int GetTippRedCur (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Activating/Deactivating Current Reduction	
Example:	LSX.GetTippRedCur(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?tippredcur	-

LSX_SetTippRedCur			
Description:	Activates/Deactivates the current reduction in inching operation. When the current reduction is active and all axes are at standstill, the motor current is reduced to the set value with "reduction".		
Delphi:	function LSX_SetTippRedCur (LSID: Integer;X,Y,Z,A: LongBool): Integer;		
C++:	int SetTippRedCur (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Activating/Deactivating Current Reduction	
Example:	LSX.SetTippRedCur(True,True,True,True);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!tippredcur	SetTipp

LSX_GetTippVel			
Description:	Shows the travel velocities in inching operation		
Delphi:	function LSX_GetTippVel(LSID: Integer;var X,Y, Z,A: Double): Integer;		
C++:	int GetTippVel (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Velocity	
Example:	LSX.GetTippVel(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?tippvel	-

LSX_SetTippVel			
Description:	Sets the travel velocities in inching operation		
Delphi:	function LSX_SetTippVel(LSID: Integer;X,Y,Z,A: Double): Integer;		
C++:	int SetTippVel (double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Velocity	
Example:	LSX.SetTippVel(10,10 ,5.5 ,10);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!tippvel	SetTipp

LSX_GetTippDir			
Description:	Shows the assigned the travel direction to the inching inputs		
Delphi:	function LSX_GetTippDir (LSID: Integer;var X,Y,Z,A: LongBool): Integer;		
C++:	int GetTippDir (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Direction	
Example:	LSX.GetTippDir(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?tippdir	-

LSX_SetTippDir			
Description:	Assigns the travel direction to the inching inputs		
Delphi:	function LSX_SetTippDir (LSID: Integer;X,Y,Z,A: LongBool): Integer;		
C++:	int SetTippDir (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	Direction	
Example:	LSX.SetTippDir(True,True,True ,False);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!tippdir	SetTipp

LSX_GetTippOutPass			
Description:	Command for reading the filter time constant in inching operation. The filter prevents sudden changes in the velocity (ramp function) when the inching inputs are activated.		
Delphi:	function LSX_GetTippOutPass (LSID: Integer;var X,Y,Z,A: Integer): Integer;		
C++:	int GetTippOutPass (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X,Y, Z,A	filter time constant	
Example:	LSX.GetTippOutPass (&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?tippoutpass	-

LSX_SetTippOutPass			
Description:	Command for setting the filter time constant in inching operation. The filter prevents sudden changes in the velocity (ramp function) when the inching inputs are activated.		
Delphi:	function LSX_SetTippOutPass (LSID: Integer;X,Y,Z,A: Integer): Integer;		
C++:	int SetTippOutPass (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X,Y, Z,A	filter time constant	
Example:	LSX.SetTippOutPass (20,20,10 ,10);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!tippoutpass	SetTipp

LSX_GetTrackBall			
Description:	Command for reading the status of the trackball operation		
Delphi:	function LSX_GetTrackBall (LSID: Integer; var status: LongBool): Integer;		
C++:	int GetTrackBall (BOOL *pbstatus);		
LabView:	Not supported		
Parameter:	status	Setting: On/Off	
Example:	LSX.GetTrackBall (&status);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?tb	-

LSX_SetTrackBall			
Description:	Command for activating and deactivating trackball operation		
Delphi:	function LSX_SetTrackBall (LSID: Integer; status: LongBool): Integer;		
C++:	int SetTrackBall (BOOL bstatus);		
LabView:	Not supported		
Parameter:	status	Setting: On/Off	
Example:	LSX.SetTrackBall (True);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!tb	Immediately

LSX_GetTrackBallEnable			
Description:	Command for reading the status of individual axes for trackball operation.		
Delphi:	function LSX_GetTrackBallEnable (LSID: Integer; var X, Y, Z, A: LongBool): Integer;		
C++:	int GetTrackBallEnable (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Setting per axis: On/Off	
Example:	LSX.GetTrackBallEnable (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?tbenable	-

LSX_SetTrackBallEnable			
Description:	Command for enabling individual axes for trackball operation.		
Delphi:	function LSX_SetTrackBallEnable (LSID: Integer; X, Y, Z, A: LongBool): Integer;		
C++:	int SetTrackBallEnable (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Setting per axis: On/Off	
Example:	LSX.SetTrackBallEnable (True, True, True, True);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!tbenable	TrackBall

LSX_GetTrackBallRedCur			
Description:	Command for reading the status of the current reduction in trackball operation. When the current reduction is active and all axes are at standstill, the motor current is reduced to the set value with "reduction".		
Delphi:	function LSX_GetTrackBallRedCur (LSID: Integer; var X, Y, Z, A: LongBool): Integer;		
C++:	int GetTrackBallRedCur (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Setting per axis: On/Off	
Example:	LSX.GetTrackBallRedCur (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?tbredcur	-

LSX_SetTrackBallRedCur			
Description:	Command for activating and deactivating the current reduction in trackball operation. When the current reduction is active and all axes are at standstill, the motor current is reduced to the set value with "reduction".		
Delphi:	function LSX_SetTrackBallRedCur (LSID: Integer; X, Y, Z, A: LongBool): Integer;		
C++:	int SetTrackBallRedCur (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Setting per axis: On/Off	
Example:	LSX.SetTrackBallRedCur (True, True, True, True);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!tbredcur	TrackBall

LSX_GetTrackBallVel			
Description:	Command for reading the maximum travel velocities in trackball operation.		
Delphi:	function LSX_GetTrackBallVel (LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetTrackBallVel (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Velocity per axis	
Example:	LSX.GetTrackBallVel (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?tbvel	-

LSX_SetTrackBallVel			
Description:	Command for setting the maximum travel velocities in trackball operation.		
Delphi:	function LSX_SetTrackBallVel (LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetTrackBallVel (double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Velocity per axis	
Example:	LSX.SetTrackBallVel (10, 10, 10, 10);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!tbvel	TrackBall

LSX_GetTrackBallOutPass			
Description:	Command for reading the filter time constant in trackball operation. The filter prevents sudden changes in the velocity (ramp function) when the inching inputs are activated.		
Delphi:	function LSX_GetTrackBallOutPass (LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetTrackBallOutPass (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	filter time constant per axis, range: 0 to 500 000 µs	
Example:	LSX.GetTrackBallOutPass (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?tboutpass	-

LSX_SetTrackBallOutPass			
Description:	Command for setting the filter time constant in trackball operation. The filter prevents sudden changes in the velocity (ramp function) when the inching inputs are activated.		
Delphi:	function LSX_SetTrackBallOutPass (LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetTrackBallOutPass (double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	filter time constant per axis, range: 0 to 500 000 µs	
Example:	LSX.SetTrackBallOutPass (20, 20, 20, 20);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!tboutpass	TrackBall

LSX_GetTrackBallDir			
Description:	Command for reading the assignment of the travel direction to the trackball direction of rotation.		
Delphi:	function LSX_GetTrackBallDir (LSID: Integer; var X, Y, Z, A: LongBool): Integer;		
C++:	int GetTrackBallDir (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Travel direction per axis: On/Off	
Example:	LSX.GetTrackBallDir (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?tbdir	-

LSX_SetTrackBallDir			
Description:	Command for assigning the travel direction to the trackball direction of rotation.		
Delphi:	function LSX_SetTrackBallDir (LSID: Integer; X, Y, Z, A: LongBool): Integer;		
C++:	int SetTrackBallDir (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Travel direction per axis: On/Off	
Example:	LSX.SetTrackBallDir (True, True, True, True);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!tbdir	TrackBall

LSX_GetTrackBallToAxis			
Description:	Command for reading the assignment of the trackball axes (horizontal and vertical) to the mechanical axes X, Y, Z and A.		
Delphi:	function LSX_GetTrackBallToAxis (LSID: Integer;var X,Y,Z,A: Integer): Integer;		
C++:	int GetTrackBallToAxis (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X,Y, Z,A	Assignment per axis: 0: No axis assignment 1: Horizontal trackball axis 2. Vertical trackball axis	
Example:	LSX.GetTrackBallToAxis (&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?tbtoaxis	-

LSX_SetTrackBallToAxis			
Description:	Command for assigning the trackball axes (horizontal and vertical) to the mechanical axes X, Y, Z and A.		
Delphi:	function LSX_SetTrackBallToAxis (LSID: Integer;X,Y,Z,A: Integer): Integer;		
C++:	int SetTrackBallToAxis (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X,Y, Z,A	Assignment per axis: 0: No axis assignment 1: Horizontal trackball axis 2. Vertical trackball axis	
Example:	LSX. SetTrackBallToAxis (2,1,0 ,0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!tbtoaxis	Trackball

4.2.12 Control panel with trackball and joystick keys

LSX_GetBPZ			
Description:	Reads the status of the additional control panel with track ball. (not for LSTEP express)		
Delphi:	function LSX_GetBPZ(LSID: Integer; var AValue: Integer): Integer;		
C++:	int GetBPZ(int *plAValue);		
LabView:	-		
Parameter:	AValue	Control panel activity 0 = Control panel is "OFF". 1 = Control panel is active and the track ball runs with a step resolution of 0.1µ. 2 = Control panel active and the track ball runs with factor.	
Example:	LSX.GetBPZ(&AValue);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?bpz	-

LSX_SetBPZ			
Description	Sets the status of the additional control panel with track ball. (not for LSTEP express)		
Delphi	function LSX_SetBPZ(LSID: Integer; AValue: Integer): Integer;		
C++	int SetBPZ(int lAValue);		
LabView:	-		
Parameter	AValue	Control panel activity 0 = Control panel is "OFF". 1 = Control panel is active and the track ball runs with a step resolution of 0.1µ. 2 = Control panel active and the track ball runs with factor.	
Example	LSX.SetBPZ(1);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?bpz	-

LSX_GetBPZJoyspeed			
Description:	Inquiry of control panel Joystick speed. (not for LSTEP express)		
Delphi:	function LSX_GetBPZJoyspeed(LSID: Integer; APar: Integer; var AValue: Double): Integer;		
C++:	int GetBPZJoyspeed(int lAPar, double *pdAValue);		
LabView:	-		
Parameter:	APar	Parameter 1 = X-axis 2 = Y-axis 3 = Z-axis	
	AValue	Maximum speed in rps	
Example:	GetBPZJoyspeed(1, &AValue); // Reading of set speed of parameter 1.		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?joyspeed	-

LSX_SetBPZJoyspeed			
Description	Set operating panel speed of joystick (not for LSTEP express)		
Delphi	function LSX_SetBPZJoyspeed(LSID: Integer; APar: Integer; AValue: Double): Integer;		
C++	int SetBPZJoyspeed(int lAPar, double dAValue);		
LabView:	-		
Parameter:	APar	Parameter 1 = X-axis 2 = Y-axis 3 = Z-axis	
	AValue	Maximum speed in rps	
Example	SetBPZJoyspeed(1, 25) // Write parameter 1 to speed 25		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!joyspeed	-

LSX_GetBPZTrackballBacklash			
Description	Function for reading the set trackball backlash on the control panel. (not for LSTEP express)		
Delphi	function LSX_GetBPZTrackballBackLash(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++	int GetBPZTrackballBackLash(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	-		
Parameter	X, Y, Z, A	Backlash in mm.	
Example	LSX.GetBPZTrackballBackLash(&X, &Y, &Z, &R);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?bpzbl	-

LSX_SetBPZTrackballBacklash			
Description	Function for setting the trackball backlash on the control panel. (not for LSTEP express)		
Delphi	function LSX_SetBPZTrackballBackLash(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++	int SetBPZTrackballBackLash (double dX, double dY, double dZ, double dA);		
LabView:	-		
Parameter	X, Y, Z, A	Backlash to be set	
Example	LSX.SetBPZTrackballBackLash(0.01, 0.01, 0.01, 0.01);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!bpzbl	-

LSX_GetBPZTrackballFactor			
Description	Reading the factor for the control panel trackball. (not for LSTEP express)		
Delphi	function LSX_GetBPZTrackballFactor(LSID: Integer; AValue: Double): Integer;		
C++	int GetBPZTrackballFactor(double *pdAValue);		
LabView:	-		
Parameter	AValue	Trackball factor in motor increments/trackball impulse	
Example	LSX.GetBPZTrackballFactor(&AValue) ;		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?bpztf	-

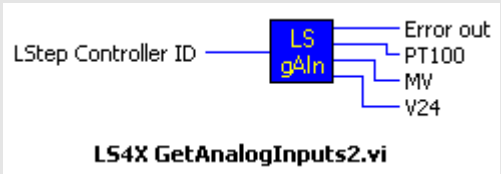
LSX_SetBPZTrackballFactor			
Description	Function for setting the trackball factor on the control panel. (not for LSTEP express)		
Delphi	function LSX_SetBPZTrackballFactor(LSID: Integer; AValue: Double): Integer;		
C++	int SetBPZTrackballFactor(double dAValue);		
LabView	-		
Parameter	AValue	Trackball factor in motor increments/trackball impulse e.g. factor = 1, i.e. a trackball impulse accounts for one motor increment.	
Example	LSX.SetBPZTrackballFactor(1.0) ;		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!bpztf	-

LSX_GetJoyOutPass			
Description	Reads the filter time constant oft he joystick function. The filter prevents jerky changes in the velocity (ramp function).		
Delphi	function LSX_GetJoyOutPass (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++	int GetJoyOutPass (int *plX, int *plY, int *plZ, int *plA);		
LabView	Not supported		
Parameter	X, Y, Z, A	Filter time constant, area: 0 – 500 000 µs	
Example	LSX.GetJoyOutPass (&X, &Y, &Z, &A);		
Other	Compatability	"SendString" command	Activation (LSTEP express Serie)
	2	? joyoutpass	-

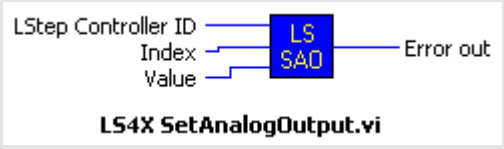
LSX_SetJoyOutPass			
Description	Sets the filter time constant oft he joystick function. The filter prevents jerky changes in the velocity (ramp function).		
Delphi	function LSX_SetJoyOutPass (LSID: Integer; X, Y, Z, A): Integer;		
C++	int SetJoyOutPass (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter	X, Y, Z, A	Filter time constant, area: 0 – 500 000 µs	
Example	LSX.SetJoyOutPass (20, 20, 20, 10);		
Other:	Compatability	"SendString" command	Activation (LSTEP express Serie)
	2	!joyoutpass	-

4.2.13 Digital and analogue inputs and outputs

LSX_GetAnalogInput			
Description:	Function for reading the current value of the analogue channel.		
Delphi:	function LSX_GetAnalogInput(LSID: Integer; Index: Integer; var Value: Integer): Integer;		
C++:	int GetAnalogInput(int lIndex,int *plValue);		
LabView:			
Parameter:	Index	Analogue channel to be read	
	Value	Pointer to integer value to where the status of the analogue channel is copied.	
Example:	LSX.GetAnalogInput(0, &Input0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?anain	-

LSX_GetAnalogInputs2			
Description:	Reading of the status of the analogue channels (channels 6, 7, 8). (only with the LSTEP-PCI, LSTEP-PC)		
Delphi:	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 where GetAnalogInputs2 is supposed to write the status of the analogue channel.	
Example:	LSX.GetAnalogInputs2(&PT100, &MV, &V24);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	-	-

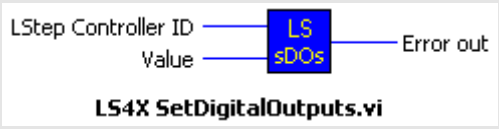
LSX_GetPT100Resistance			
Description:	Outputs the resistance value of a PT100 connected to the analog input (pins 9 and 10). Sensor resistance range: 100–144 Ω ≙ 0–115 °C If the resistance is outside the permitted range, the function returns error code 5 (value outside the valid range).		
Delphi:	function LSX_GetPT100Resistance(LSID: Integer; var Resistance: Double): Integer;		
C++:	int GetPT100Resistance (double *pdResistance);		
LabView:	-		
Parameter:	Resistance	Pointer to float value where PT100Resistance is supposed to write the current resistance.	
Example:	LSX.GetPT100Resistance(&PT100);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?pt100res	-

LSX_SetAnalogOutput			
Description:	Function for setting an analogue output.		
Delphi:	function LSX_SetAnalogOutput(LSID: Integer; Index: Integer; Value: Integer): Integer;		
C++:	int SetAnalogOutput(int lIndex,int lValue);		
LabView:			
Parameter:	Index	Number of analogue channel	
	Value	Control of analogue output in %	
Example:	LSX.SetAnalogOutput(0, 100); // Set output 0 to maximum		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!anaout	-

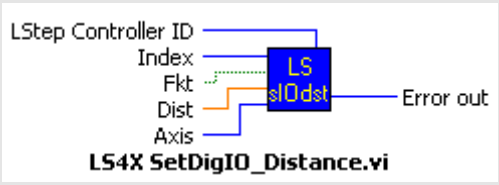
LSX_GetDigitalInputs			
Description:	Function for reading the digital inputs 0 through 15.		
Delphi:	function LSX_GetDigitalInputs(LSID: Integer; var Value: Integer): Integer;		
C++:	int GetDigitalInputs(int *plValue);		
LabView:			
Parameter:	Value	Pointer to integer value containing the status of the digital inputs as a bit mask. Bit 0 = Input 0 Bit 1 = Input 1 … Value 0 = A logical 0 is given on the input Value 1 = A logical 1 is given on the input	
Example:	int Eingange; LSX.GetDigitalInputs(&Inputs); if (inputs & 16) ... // if Input pin 4 is set		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?digin	-

LSX_GetDigitalInputsE			
Description:	Reading of additional digital inputs 16 through 31.		
Delphi:	function LSX_GetDigitalInputsE(LSID: Integer; var Value: Integer): Integer;		
C++:	int GetDigitalInputsE(int *plValue);		
LabView:	Not supported		
Parameter:	Value	Pointer to integer value containing the status of the digital inputs as a bit mask. Bit 0 = Input 16 Bit 1 = Input 17 … Value 0 = A logical 0 is given on the input Value 1 = A logical 1 is given on the input	
Example:	LSX.GetDigitalInputsE(&i);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?diginext (LStepex-press)/ ?edigin	-

LSX_SetDigitalOutput			
Description:	Function for setting a digital output.		
Delphi:	function LSX_SetDigitalOutput(LSID: Integer; Index: Integer; Value: LongBool): Integer;		
C++:	int SetDigitalOutput(int lIndex,BOOL Value);		
LabView:			
Parameter:	Index	Number of the digital output 0 = Output 0 1 = Output 1 …	
	Value	Set status to “0” or “1” True = Set output to 1 False = Set output to 0	
Example:	LSX.SetDigitalOutput(0, true); // Set output pin 0 to “1"		
Other:	Compatibility	“SendString” command	Activation (LSTEP express series)
	3	!digout	-

LSX_SetDigitalOutputs			
Description:	Function for simultaneously setting the digital outputs 0 through 15.		
Delphi:	function LSX_SetDigitalOutputs(LSID: Integer; Value: Integer): Integer;		
C++:	int SetDigitalOutputs(int lValue);		
LabView:			
Parameter:	Value	Bit mask for setting the outputs Bit 0 = Output 0 Bit 1 = Output 1 … Value 0 = Set output to 0 Value 1 = Set output to 1	
Example:	LSX.SetDigitalOutputs($03); // Set outputs 0 and 1 to 1, the remaining ones to 0		
Other:	Compatibility	“SendString” command	Activation (LSTEP express series)
	3	!digout	-

<table>
<tr><th colspan="5">LSX_SetDigitalOutputsE</th></tr>
<tr><td>Description:</td><td colspan="4">Function for simultaneously setting the digital outputs 16 through 31.</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_SetDigitalOutputsE(LSID: Integer; Value: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int SetDigitalOutputsE(int lValue);</td></tr>
<tr><td>LabView:</td><td colspan="4">Not supported</td></tr>
<tr><td>Parameter:</td><td>Value</td><td colspan="3">Bit mask for setting the outputs
Bit 0 = Output 16
Bit 1 = Output 17
…
Value 0 = Set output to 0
Value 1 = Set output to 1</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.SetDigitalOutputsE($03); // Set outputs 16 and 17 to 1, the remaining ones to 0</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>!digoutext (LStepex-press) / !edigout</td><td colspan="2">-</td></tr>
</table>

LSX_SetDigIO_Distance			
Description:	Function for activating an output dependent on the set distance before/behind of the target position. (not for LSTEP express)		
Delphi:	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	Number of the digital output	
	Fkt	Activation type False = Activation of an output dependent on the set distance before the target position. True = Activation of an output dependent on the set distance behind the start position.	
	Dist	Distance in the set dimension	
	Axis	Axis to which the function is to be allocated. 1 = X-axis 2 = Y-axis …	
Example:	LSX.SetDigIO_Distance(7, false, 78.9, 3); /* Output 7 is activated 78.9mm before the target position (Z-axis) is reached. */		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!digfkt/ !edigfkt	-

LSX_SetDigIO_EmergencyStop			
Description:	Function for allocating a digital input as emergency-stop pin (not for LSTEP express)		
Delphi:	function LSX_SetDigIO_EmergencyStop(LSID: Integer; Index: Integer): Integer;		
C++:	int SetDigIO_EmergencyStop(int lIndex);		
LabView:			
Parameter:	Index	Number of digital input to which the function is to be allocated 0 = Input 0 1 = Input 1 …	
Example:	LSX.SetDigIOEmergencyStop(15); // Not-Stopp-Pin 15		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!digfkt / !edigfkt	-

LSX_SetDigIO_Off			
Description:	Deactivation of the function of the digital inputs/outputs. (No impact on inputs/outputs). (not for LSTEP express)		
Delphi:	function LSX_SetDigIO_Off(LSID: Integer; Index: Integer): Integer;		
C++:	int SetDigIO_Off(int lIndex);		
LabView:			
Parameter:	Index	Number of digital input whose function allocation is to be deactivated. 0 = Input 0 1 = Input 1 …	
Example:	LSX.SetDigIO_Off(0); // dig. Fkt. Input/Output pin 0 OFF		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!digfkt / !edigfkt	-

LSX_SetDigIO_Polarity			
Description:	Setting of the polarity for the different functions of digital inputs/outputs (not for LSTEP express)		
Delphi:	function LSX_SetDigIO_Polarity(LSID: Integer; Index: Integer; High: LongBool): Integer;		
C++:	int SetDigIO_Polarity(int lIndex,BOOL High);		
LabView:			
Parameter:	Index	Number of digital input whose polarity is to be changed. 0 = Input 0 1 = Input 1 …	
	High	Polarity setting True = High-active False = Low-active	
Example:	LSX.SetDigIO_Polarity(3, True); // Input-/Outputpin 3 high-active		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!digfkt / !edigfkt	-

<table>
<tr><th colspan="4">LSX_GetDigOutLinktoSignal</th></tr>
<tr><td>Description:</td><td colspan="3">Shows if a signal is assigned to one of the 16 respectively 32 digital outputs (24V). If a signal is assigned to an output, this can no longer be described by "!digout'.The signals 1 through 6 are inquired and put out at a sampling rate of about 1 ms and may only be assigned to one output. The signals 10 through 13, 100 through 115 and 200 through 205 are inquired and put out with at a rate of 8 ms and may be assigned to several outputs.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_GetDigOutLinktoSignal (LSID: Integer; Output: Integer;
var toSignal: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int GetDigOutLinktoSignal (int lOutput, int *pltoSignal);</td></tr>
<tr><td>LabView:</td><td colspan="3">Not supported</td></tr>
<tr><td rowspan="15">Parameter:</td><td>Output</td><td colspan="2">Signal</td></tr>
<tr><td rowspan="14">0 to 15 respectively 31</td><td colspan="2">0 ➔ Cancel signal assignment</td></tr>
<tr><td colspan="2">1 ➔ Standstill message</td></tr>
<tr><td colspan="2">2 ➔ Stop active message</td></tr>
<tr><td colspan="2">3 ➔ Target window message</td></tr>
<tr><td colspan="2">4 ➔ At least one power amplifiers activated message</td></tr>
<tr><td colspan="2">5 ➔ Manual operation active message</td></tr>
<tr><td colspan="2">6 ➔ Velocity below threshold</td></tr>
<tr><td colspan="2">10 ➔ Manual operation in X-axis active message</td></tr>
<tr><td colspan="2">11 ➔ Manual operation in Y-axis active message</td></tr>
<tr><td colspan="2">12 ➔ Manual operation in Z-axis active message</td></tr>
<tr><td colspan="2">13 ➔ Manual operation in A-axis active message</td></tr>
<tr><td colspan="2">100 through 131 ➔ Digital input 0 ... 31 (24V)</td></tr>
<tr><td colspan="2">200 through 205 ➔ Digital input 0 ... 5 (TTL level)</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.GetDigitalOutLinktoSignal(Output,&toSignal);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” Command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>2</td><td>?digoutlinktosignal</td><td>-</td></tr>
</table>

LSX_SetDigitalOutLinktoSignal			
Description:	Assigns a signal to one of the 16 respectively 32 digital outputs (24V). If a signal is assigned to an output, this can no longer be described by "!digout'. The signals 1 through 6 are inquired and put out at a sampling rate of about 1 ms and may only be assigned to one output. The signals 10 through 13, 100 through 115 and 200 through 205 are inquired and put out with at a rate of 8 ms and may be assigned to several outputs.		
Delphi:	function LSX_SetDigOutLinktoSignal (LSID : Integer;Output: Integer; toSignal: Integer): Integer;		
C++:	int SetDigOutLinktoSignal (int lOutput, int ltoSignal);		
LabView:	Not supported		
Parameter:	Output	Signal	
	0 to 15 respectively 31	0 = Cancel signal assignment	
		1 = Standstill message	
		2 = Stop active message	
		3 = Target window message	
		4 = At least one power amplifiers activated message	
		5 = Manual operation active message	
		6 = Velocity below threshold	
		10 = Manual operation in X-axis active message	
		11 = Manual operation in Y-axis active message	
		12 = Manual operation in Z-axis active message	
		13 = Manual operation in A-axis active message	
		100 through 131 = Digital input 0 ... 31 (24V)	
		200 through 205 = Digital input 0 ... 5 (TTL level)	
Beispiel:	LSX.SetDigitalOutLinktoSignal(5,1);		
Sonstiges:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!digoutlinktosignal	-

LSX_GetInvertDigOutSignal			
Description:	Shows if a signal is inverted, which can be assigned to a digital output by using command „digoutlinktosignal“.		
Delphi:	function LSX_GetInvertDigOutSignal (LSID: Integer;Output: Integer; var invert: LongBool): Integer;		
C++:	int GetInvertDigOutSignal (int lOutput, BOOL *pbinvert);		
LabView:	Not supported		
Parameter:	Signal		
	False = No inverting True = Inverting		
Example:	LSX.GetInvertDigOutSignal(Output,&invert);		
Other:	Compatibility	“SendString” Command	Activation (LSTEP express series)
	2	?invertdigoutsingal	-

LSX_SetInvertDigOutSignal			
Description:	Inverts a signal, which can be assigned to a digital output by using command „digoutlinktosignal“.		
Delphi:	function LSX_SetDigOutLinktoSignal (LSID : Integer;Output: Integer; toSignal: Integer): Integer;		
C++:	int SetDigOutLinktoSignal (int lOutput, int ltoSignal);		
LabView:	Not supported		
Parameter:	Signal		
	False = No inverting True = Inverting		
Example:	LSX.SetInvertDigOutSignal(5,True);		
Other:	Compatibility	“SendString” Command	Activation (LSTEP express series)
	2	!invertdigoutsignal	-

<table>
<tr><th colspan="5">LSX_GetDigInStatus</th></tr>
<tr><td>Description:</td><td colspan="4">Digital inputs 0 to 15
Commands for reading the status of the 16 digital inputs with 24 V level or 6 inputs with TTL level if the same are changed. The inputs are checked for changes at cyclical intervals of approx. 8 ms. Upon activation of automatic transmission, the input status is issued once directly.</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_GetDigInStatus (LSID: Integer; status: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int GetDigInStatus (int *plstatus);</td></tr>
<tr><td>LabView:</td><td colspan="4">Not Supported</td></tr>
<tr><td>Parameter:</td><td>status</td><td colspan="3">0: Inactive
1: Active, bit-wise transmission (16 bit)
2: Active, transmission in hexadecimal format with 4 places, low byte and high byte are reversed
3: Active, bit-wise transmission (16 bit), extended record channel 4
4: Active, transmission in hexadecimal format with 4 places, low byte and high byte are reversed, extended record channel 4</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.GetDigInStatus (&status);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” Command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>2</td><td>?diginstatus</td><td colspan="2">-</td></tr>
</table>

LSX_SetDigInStatus, LSX_SetDigInStatus			
Description:	Digital inputs 0 to 15 Commands for activating the automatic transmission of the 16 digital inputs with 24 V level or 6 inputs with TTL level if the same are changed. The inputs are checked for changes at cyclical intervals of approx. 8 ms. Upon activation of automatic transmission, the input status is issued once directly.		
Delphi:	function LSX_SetDigInStatus(LSID: Integer; status: Integer; var Feedback: PAnsiChar; MaxLen: Integer): Integer; function LSX_SetDigInStatusW(LSID: Integer; status: Integer; var Feedback: PWideChar; MaxLen: Integer): Integer;		
C++:	int SetDigInStatus (int lstatus, char *pcFeedback, int lMaxLen); int SetDigInStatusW (int lstatus, TCHAR *pcFeedback, int lMaxLen);		
LabView:	Not Supported		
Parameter:	status	0: Inactive 1: Active, bit-wise transmission (16 bit) 2: Active, transmission in hexadecimal format with 4 places, low byte and high byte are reversed 3: Active, bit-wise transmission (16 bit), extended record channel 4 4: Active, transmission in hexadecimal format with 4 places, low byte and high byte are reversed, extended record channel 4	
Feedback	Feedback	Bit-wise transmission: !0 xxxxxxxxxxxxxxxx (diginstatus) !1 xxxxxxxxxxxxxxxx (diginextstatus) !2 xxxxxxxxxxxxxxxx (ttldiginstatus) Hexadecimal format: !0 xxxx (diginstatus / low byte and high byte reversed) !1 xxxx (diginextstatus / low byte and high byte reversed) !2 xxxx (ttldiginstatus / low byte and high byte reversed)	
Example:	LSX.SetDigInStatus (status, pcFeedback,256);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!diginstatus	Immediately

LSX_GetDigInExtStatus			
Description:	Digital inputs 16 to 31 Commands for reading the status of the 16 digital inputs with 24 V level or 6 inputs with TTL level if the same are changed. The inputs are checked for changes at cyclical intervals of approx. 8 ms. Upon activation of automatic transmission, the input status is issued once directly.		
Delphi:	function LSX_GetDigInExtStatus (LSID: Integer; status: Integer): Integer;		
C++:	int GetDigInExtStatus (int *plstatus);		
LabView:	Not Supported		
Parameter:	status	0: Inactive 1: Active, bit-wise transmission (16 bit) 2: Active, transmission in hexadecimal format with 4 places, low byte and high byte are reversed 3: Active, bit-wise transmission (16 bit), extended record channel 4 4: Active, transmission in hexadecimal format with 4 places, low byte and high byte are reversed, extended record channel 4	
Example:	LSX.GetDigInExtStatus (&status);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?diginextstatus	-

LSX_SetDigInExtStatus, LSX_SetDigInExtStatusW			
Description:	Digital inputs 16 to 31 Commands for activating the automatic transmission of the 16 digital inputs with 24 V level or 6 inputs with TTL level if the same are changed. The inputs are checked for changes at cyclical intervals of approx. 8 ms. Upon activation of automatic transmission, the input status is issued once directly.		
Delphi:	function LSX_SetDigInExtStatus(LSID: Integer; status: Integer;var Feedback: PAnsiChar; MaxLen: Integer): Integer; function LSX_SetDigInExtStatusW(LSID: Integer; status: Integer; var Feedback: PWideChar; MaxLen: Integer): Integer;		
C++:	int SetDigInExtStatus (int lstatus, char *pcFeedback, int lMaxLen); int SetDigInExtStatusW (int lstatus, TCHAR *pcFeedback, int lMaxLen);		
LabView:	Not Supported		
Parameter:	status	0: Inactive 1: Active, bit-wise transmission (16 bit) 2: Active, transmission in hexadecimal format with 4 places, low byte and high byte are reversed 3: Active, bit-wise transmission (16 bit), extended record channel 4 4: Active, transmission in hexadecimal format with 4 places, low byte and high byte are reversed, extended record channel 4	
Feedback	Feedback	Bit-wise transmission: !0 xxxxxxxxxxxxxxx (diginstatus) !1 xxxxxxxxxxxxxxx (diginextstatus) !2 xxxxxxxxxxxxxxx (ttldiginstatus) Hexadecimal format: !0 xxxx (diginstatus / low byte and high byte reversed) !1 xxxx (diginextstatus / low byte and high byte reversed) !2 xxxx (ttldiginstatus / low byte and high byte reversed)	
Example:	LSX.SetDigInExtStatus (status, pcFeedback,256);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!diginextstatus	Immediately

LSX_GetTTLDigIn			
Description:	Command for reading the 6 digital inputs with TTL level		
Delphi:	function LSX_GetTTLDigIn (LSID: Integer; status: Integer): Integer;		
C++:	int GetTTLDigIn (int *plstatus);		
LabView:	Not Supported		
Parameter:	status	Status of digital inputs: On/Off	
Example:	LSX.GetTTLDigIn (&status);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?ttldigin	-

LSX_GetTTLDigOut			
Description:	Command for reading the 4 digital TTL outputs. The function must be enabled by configuring pins 16 through 19 of the 50-pole multifunction port (MFP) as TTL outputs (see command "ttloutconfig").		
Delphi:	function LSX_GetTTLDigOut (LSID: Integer; status: Integer): Integer;		
C++:	int GetTTLDigOut (int *plstatus);		
LabView:	Not Supported		
Parameter:	status	Output status	
Example:	LSX.GetTTLDigOut (&status);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?ttldigout	-

LSX_SetTTLDigOut			
Description:	Command for setting and deleting the 4 digital TTL outputs. The function must be enabled by configuring pins 16 through 19 of the 50-pole multifunction port (MFP) as TTL outputs (see command "ttloutconfig").		
Delphi:	function LSX_SetTTLDigOut (LSID: Integer; status: Integer): Integer;		
C++:	int SetTTLDigOut (int *plstatus);		
LabView:	Not Supported		
Parameter:	status	Output status	
Example:	LSX.SetTTLDigOut (1110);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!ttldigout	Immediately

LSX_GetMotorBrakeDigOut			
Description:	Command for reading the motor brakes as digital outputs. The function must be enabled by configuring the motor brakes as digital outputs (see command "motorbrake").		
Delphi:	function LSX_GetMotorBrakeDigOut (LSID: Integer; var X, Y, Z, A: LongBool): Integer;		
C++:	int GetMotorBrakeDigOut (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not Supported		
Parameter:	X, Y, Z, A	Setting per axis: HIGH/LOW	
Example:	LSX.GetMotorBrakeDigOut (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?motorbrakedigout	-

LSX_SetMotorBrakeDigOut			
Description:	Command for setting the motor brakes as digital outputs. The function must be enabled by configuring the motor brakes as digital outputs (see command "motorbrake").		
Delphi:	function LSX_SetMotorBrakeDigOut (LSID: Integer; X, Y, Z, A: LongBool): Integer;		
C++:	int SetMotorBrakeDigOut (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not Supported		
Parameter:	X, Y, Z, A	Setting per axis: HIGH/LOW	
Example:	LSX.SetMotorBrakeDigOut (1, 1, 1, 0);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!motorbrakedigout	Immediately

LSX_GetMotorBrakeOut			
Description:	Command for reading the motor brake outputs		
Delphi:	function LSX_GetMotorBrakeOut (LSID: Integer; var X, Y, Z, A: LongBool): Integer;		
C++:	int GetMotorBrakeOut (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not Supported		
Parameter:	X, Y, Z, A	Setting per axis: HIGH/LOW	
Example:	LSX.GetMotorBrakeOut (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?motorbrakeout	-

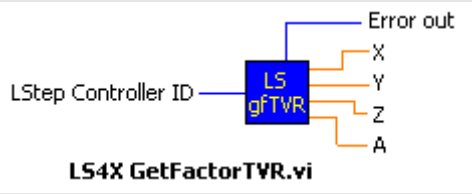
LSX_GetTVR			
Description:	Command for reading the status of TVR-IN.		
Delphi:	function LSX_GetTVR (LSID: Integer; var X, Y, Z, A: LongBool): Integer;		
C++:	int GetTVR (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not Supported		
Parameter:	X, Y, Z, A	Settin per axis: On/Off	
Example:	LSX.GetTVR (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?tvr	-

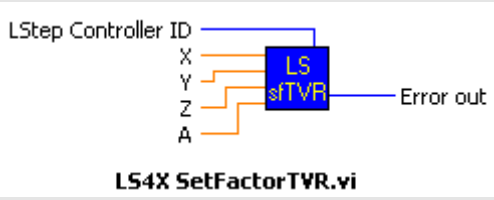
LSX_SetTVR			
Description:	Command for activating and deactivating TVR-IN.		
Delphi:	function LSX_SetTVR (LSID: Integer; X, Y, Z, A: LongBool): Integer;		
C++:	int SetTVR (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not Supported		
Parameter:	X, Y, Z, A	Settin per axis: On/Off	
Example:	LSX.SetTVR (1, 1, 1, 0);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!tvr	Immediately

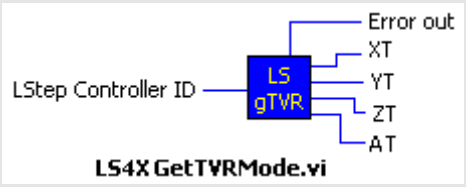
LSX_GetTVRToAxis			
Description:	Command for reading th assignment of the two QEP inputs to the TVR-IN axes.		
Delphi:	function LSX_GetTVRToAxis (LSID: Integer;var X,Y,Z,A: Integer): Integer;		
C++:	int GetTVRToAxis (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not Supported		
Parameter:	X,Y,Z,A	Assignment: 0: No axis assignment 1: QEP 1 2. QEP 2	
Example:	LSX.GetTVRToAxis (&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?tvrtoaxis	-

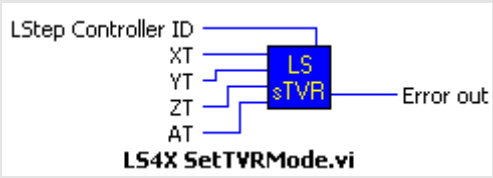
LSX_SetTVRToAxis			
Description:	Command for assigning the two QEP inputs to the TVR-IN axes.		
Delphi:	function LSX_SetTVRToAxis (LSID: Integer;X,Y,Z,A: Integer): Integer;		
C++:	int SetTVRToAxis (int lX, int lY, int lZ, int lA);		
LabView:	Not Supported		
Parameter:	X,Y,Z,A	Assignment: 0: No axis assignment 1: QEP 1 2. QEP 2	
Example:	LSX.SetTVRToAxis (2,1,0 ,0);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!tvrtoaxis	TVR

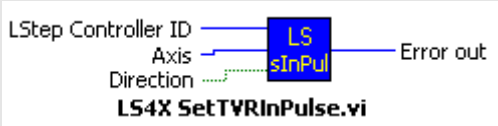
4.2.14 Cycle Forward / Back In

LSX_GetFactorTVR			
Description:	Function for reading the cycle forward/back factor.		
Delphi:	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, A	Set cycle forward/back factor in motor increments/cycle	
Example:	LSX.GetFactorTVR(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?tvrf	-

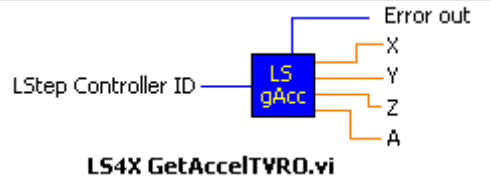
LSX_SetFactorTVR			
Description:	Function for setting the cycle forward/back factor.		
Delphi:	function LSX_SetFactorTVR(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetFactorTVR(double dX,double dY,double dZ,double dA);		
LabView:			
Parameter:	X, Y, Z, A	Cycle forward/back factor in motor increments/cycle to be set	
Example:	LSX.SetFactorTVR(2.0, 2.0, 0, 0); /* Cycle forward/back is to operate with factor 2 for the X and Y-axis */		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!tvrf	tvr

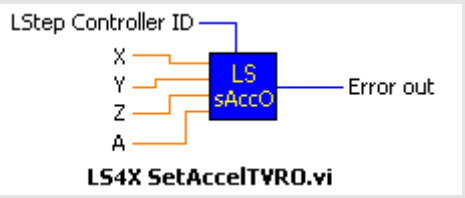
<table>
<tr><th colspan="4">LSX_GetTVRMode</th></tr>
<tr><td>Description:</td><td colspan="3">Reads the set cycle forward/back mode.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_GetTVRMode(LSID: Integer; var XT, YT, ZT, AT: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int GetTVRMode(int *plXT, int *plYT, int *plZT, int *plAT);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>XT, YT, ZT, AT</td><td colspan="2">Set cycle forward/back mode
0 = Pulse Forward/Back is “OFF”
1 = Normal cycle Forward/Back processing
2 = Cycle Forward/Back operates with a factor
3 = Cycle Forward/Back processing requires external enabling by the triggerout pin (MFP).
4 = Combination of 2 & 3.</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.GetTVRMode(&XT, &YT, &ZT, &AT);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>1</td><td>?tvr</td><td>-</td></tr>
</table>

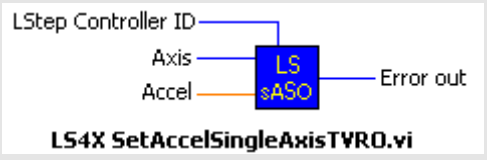
<table>
<tr><th colspan="5">LSX_SetTVRMode</th></tr>
<tr><td>Description:</td><td colspan="4">Function for setting the cycle forward/back mode.</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_SetTVRMode(LSID: Integer; XT, YT, ZT, AT: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int SetTVRMode(int lXT, int lYT,int lZT,int lAT);</td></tr>
<tr><td>LabView:</td><td colspan="4"> 
</td></tr>
<tr><td>Parameter:</td><td>XT, YT, ZT, AT</td><td colspan="3">Cycle forward/back mode to be set
0 = Pulse Forward/Back is “OFF”
1 = Normal cycle Forward/Back processing
2 = Cycle Forward/Back operates with a factor
3 = Cycle Forward/Back processing requires external enabling by the triggerout pin (MFP).
4 = Combination of 2 & 3.</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.SetTVRMode(1, 1, 0, 0); // TVR X- and Y-axis On</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>1</td><td>!tvr</td><td colspan="2">-</td></tr>
</table>

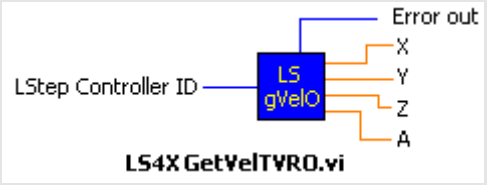
LSX_SetTVRInPulse			
Description:	This function can be used to control the cycle forward/back function by software. (not for LSTEP express)		
Delphi:	function LSX_SetTVRInPulse(LSID: Integer; Axis: Integer; Direction: Boolean): Integer;		
C++:	int SetTVRInPulse(int Axis, BOOL Direction);		
LabView:			
Parameter:	Axis	Axis to which the software pulse is to be sent 1 = X-axis 2 = Y-axis …	
	Direction	Direction signal True = Forward False = Back	
Example:	LSX.SetTVRInPulse (2, true); // 1 cycle forward on Y-axis.		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!px / !nx !py / !ny …	-

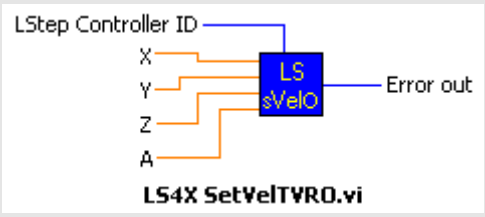
4.2.15 Cycle Forward/Back outputs for additional axes

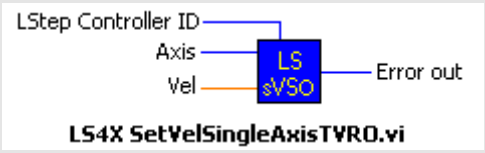
LSX_GetAccelTVRO			
Description:	Reads the set accelerations for additional cycle forward/back output axes (not for LSTEP express)		
Delphi:	function LSX_GetAccelTVRO(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetAccelTVRO(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	X, Y, Z, A	Acceleration values in rps²	
Example:	LSX.GetAccelTVRO(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?tvroa	-

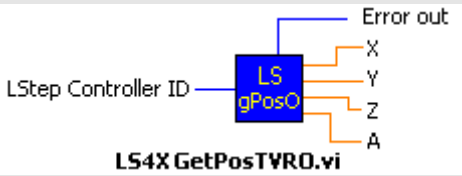
LSX_SetAccelTVRO			
Description:	Setting of acceleration for additional cylce forward/back output axes (not for LSTEP express)		
Delphi:	function LSX_SetAccelTVRO(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetAccelTVRO(double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	X, Y, Z, A	Acceleration values in rps²	
Example:	LSX.SetAccelTVRO(1.0, 1.5, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!tvroa	-

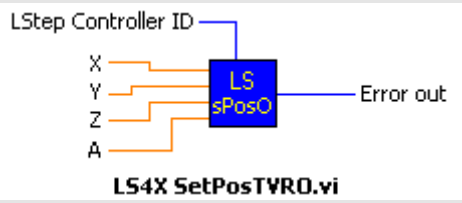
LSX_SetAccelSingleAxisTVRO			
Description:	Function for setting the acceleration for a single TVRO axis (not for LSTEP express)		
Delphi:	function LSX_SetAccelSingleAxisTVRO(LSID: Integer; Axis: Integer; Accel: Double): Integer;		
C++:	int SetAccelSingleAxisTVRO(int lAxis, double dAccel);		
LabView:			
Parameter:	Axis	Axis whose acceleration is to be set TVRO X = 1 TVRO Y = 2 …	
	Accel	Acceleration to be set in rps²	
Example:	LSX. SetAccelSingleAxis(2, 50.0); // The Z-axis is accelerated with 50 rps²		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!tvroa	-

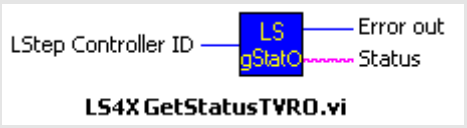
LSX_GetVelTVRO			
Description:	Reads the set velocities for the TVRO axes from the controller (not for LSTEP express)		
Delphi:	function LSX_GetVelTVRO(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetVelTVRO(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	X, Y, Z, A	Speed values in rps	
Example:	LSX.GetVelTVRO(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?tvrov	-

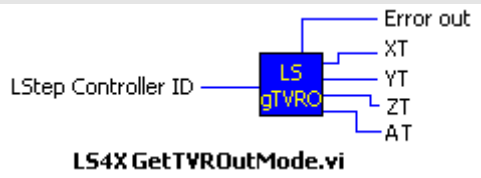
LSX_SetVelTVRO			
Description:	Function for setting the velocity for the TVRO axes (not for LSTEP express)		
Delphi:	function LSX_SetVelTVRO(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetVelTVRO(double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	X, Y, Z, A	Velocities in rps	
Example:	LSX.SetVelTVRO(1.0, 1.5, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!tvrov	-

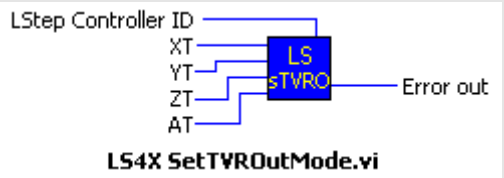
LSX_SetVelSingleAxisTVRO			
Description:	Function for setting the velocity for a single TVRO axis. (not for LSTEP express)		
Delphi:	function LSX_SetVelSingleAxisTVRO(LSID: Integer; Axis: Integer; Vel: Double): Integer;		
C++:	int SetVelSingleAxisTVRO(int lAxis, double dVel);		
LabView:			
Parameter:	Axis	Axis whose acceleration is to be set TVRO X = 1 TVRO Y = 2 …	
	Vel	Velocity value to be set in rps	
Example:	LSX.SetVelSingleAxis(1, 10.0); // The X-axis should run with a max. velocity 10 rp/s		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!tvrov	-

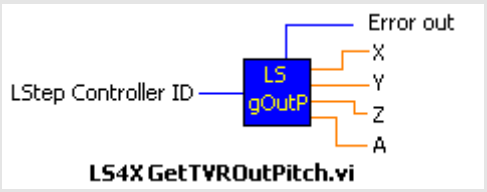
LSX_GetPosTVRO			
Description:	Function for reading the position of the additional TVRO axes (not for LSTEP express)		
Delphi:	function LSX_GetPosTVRO(LSID: Integer; var dX, dY, dZ, dA: Double): Integer;		
C++:	int GetPosTVRO (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	dX, dY, dZ, dA	Position value dependent on the set dimension	
Example:	LSX.GetPosTVRO(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?tvropos	-

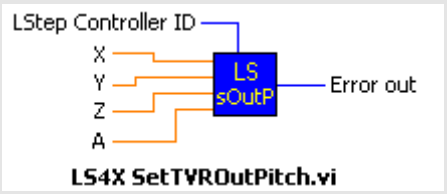
LSX_SetPosTVRO			
Description:	Sett ing of position of the additional TVRO axes (not for LSTEP express)		
Delphi:	function LSX_SetPosTVRO(LSID: Integer; dX, dY, dZ, dA: Double): Integer;		
C++:	int SetPosTVRO(double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	dX, dY, dZ, dA	Position value dependent on the set dimension	
Example:	LSX.SetPosTVRO(10.0, 5.0, 0.0, 0.0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!tvropos	-

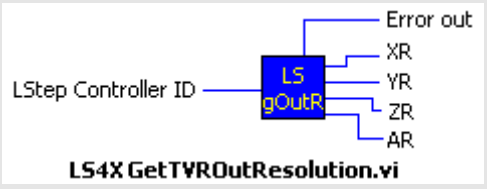
<table>
<tr><th colspan="4">LSX_GetStatusTVRO, LSX_GetStatusTVROW</th></tr>
<tr><td>Description:</td><td colspan="3">Delivers the current status of the additional TVRO axes
(not for LSTEP express)</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_GetStatusTVRO(LSID: Integer; pcStat: PAnsiChar; MaxLen: Integer): Integer;
function LSX_GetStatusTVRO(LSID: Integer; pcStat: PWideChar; MaxLen: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int GetStatusTVRO(char *pcStat, int lMaxLen);
int GetStatusTVROW(TCHAR *pcStat, int lMaxLen);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="2">Parameter:</td><td>pcStat</td><td colspan="2">Pointer to a buffer to which the status string is returned. The axis mask contains one of the following characters:
@ = Axis is at standstill
M = Axis is moving (Motion)
- = Axis is not enabled</td></tr>
<tr><td>MaxLen</td><td colspan="2">Maximum number of characters, which may be copied into the buffer.</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.GetStatusTVRO(pcStat, 256);
// Move additional Z-axis by 5mm in positive direction</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>1</td><td>?tvrostatus</td><td>-</td></tr>
</table>

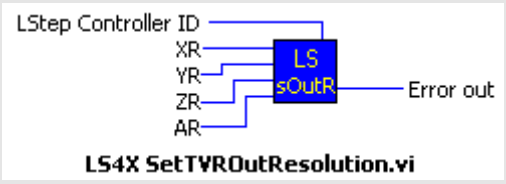
LSX_GetTVROutMode			
Description:	Function for reading the set mode from cycle forward/back output (not for LSTEP express)		
Delphi:	function LSX_GetTVROutMode(LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetTVROutMode(int *plXT, int *plYT, int *plZT, int *plAT);		
LabView:			
Parameter:	X, Y, Z, A	Set TVRO mode 0 = Cycle Forward/Backward is deactivated 1 = Cycle Forward/Backward is activated	
Example:	LSX.GetTVROutMode(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?tvrout	-

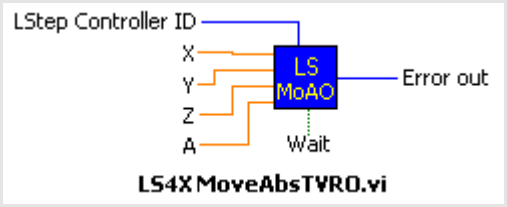
LSX_SetTVROutMode			
Description:	Function for setting the set mode of the cycle forward/back output (not for LSTEP express)		
Delphi:	function LSX_SetTVROutMode(LSID: Integer; Integer): Integer; Integer;		
C++:	int SetTVROutMode(int lXT, int lYT, int lZT, int lAT);		
LabView:			
Parameter:	X, Y, Z, A	TVRO mode to be set 0 = Cycle Forward/Backward is deactivated 1 = Cycle Forward/Backward is activated	
Example:	LSX.SetTVROutMode(1, 0, 1, 0); //Cycle Forw/Back is to be activated for axes x and z, and deactivated for y and a.		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!tvrout	-

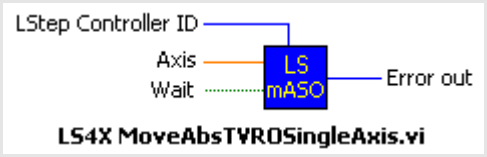
LSX_GetTVROutPitch			
Description:	Reads the spindle pitch of the additional TVRO axes (not for LSTEP express)		
Delphi:	function LSX_GetTVROutPitch (LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetTVROutPitch (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	X, Y, Z A	Spindle pitches in mm/revolution	
Example:	LSX.GetTVROutPitch(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?tvropitch	-

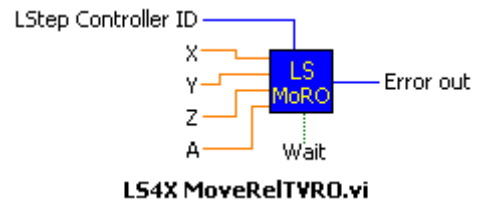
LSX_SetTVROutPitch			
Description:	Sets the spindle pitches for the additional axes (not for LSTEP express)		
Delphi:	function LSX_SetTVROutPitch(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetTVROutPitch(double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	X, Y, Z, A	Spindle pitches in mm/revolution	
Example:	LSX.SetTVROutPitch(1.0, 4.0, 1.0, 1.0); /* Spindle pitch for Y-axis is 4 mm. For X, Z and A axes, spindles with a pitch of 1 mm are used*/		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!tvropitch	-

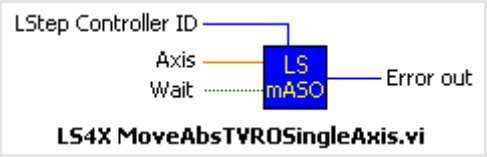
LSX_GetTVROutResolution			
Description:	Reads the resolution set in the LSTEP for the power amplifier to be controlled (not for LSTEP express)		
Delphi:	function LSX_GetTVROutResolution(LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetTVROutResolution(int *plX, int *plY, int *plZ, int *plA);		
LabView:			
Parameter:	X, Y, Z, A	Set resolution in impulses/revolution	
Example:	LSX.GetTVROutResolution(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?tvrores	-

LSX_SetTVROutResolution			
Description:	Writes the resolution for controlling the external power amplifier into the LSTEP controller (not for LSTEPexpress). (not for LSTEP express)		
Delphi:	Integer; X, Y, Z, A: Integer): Integer; Integer;		
C++:	int SetTVROutResolution(int lX, int lY, int lZ, int lA);		
Parameter:			
X, Y, Z, A	Resolution to be set in impulses/revolution	Example:	
Example:	LSX.SetTVROutResolution(1000, 1000, 0, 0); Other:		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	LS_MoveAbsTVRO	-

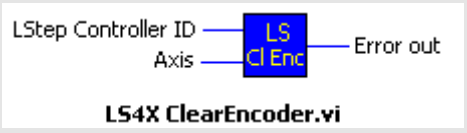
<table>
<tr><th colspan="4">LSX_MoveAbsTVRO</th></tr>
<tr><td>Description:</td><td colspan="3">Approach absolute position with the TVRO axes from the current position.
(not for LSTEP express)</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_MoveAbsTVRO(LSID: Integer; X, Y, Z, A: Double; Wait: Long-Bool): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int MoveAbsTVRO(double dX, double dY, double dZ, double dA, BOOL bWait);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="2">Parameter:</td><td>X, Y, Z, A</td><td colspan="2">Absolute position indication in the set axis dimension</td></tr>
<tr><td>Wait</td><td colspan="2">Specifies whether the function should return directly or only after reaching the position.
True = Wait until position is reached
False = Do not wait until position is reached</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.MoveAbsTVRO(10.0, 10.0, 10.0, 10.0, true);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>1</td><td>!tvromoa</td><td>-</td></tr>
</table>

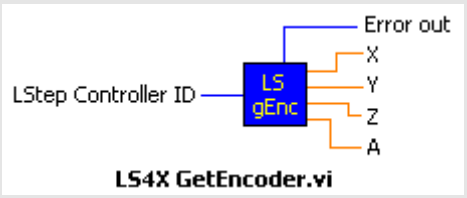
<table>
<tr><th colspan="5">LSX_MoveAbsTVROSingleAxis</th></tr>
<tr><td>Description:</td><td colspan="4">Absolute positioning of single TVRO axis
(not for LSTEP express)</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_MoveAbsTVROSingleAxis(LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int MoveAbsTVROSingleAxis(int lAxis, double dValue, BOOL bWait);</td></tr>
<tr><td>LabView:</td><td colspan="4"> 
</td></tr>
<tr><td rowspan="2">Parameter:</td><td>Axis</td><td colspan="3">Axis to be moved
1 = X-axis
2 = Y-axis
…</td></tr>
<tr><td>Value</td><td colspan="3">Position (input depends on the set dimension)</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.MoveAbsTVROSingleAxis(2, 10.0);
//Position additional Y-axis to 10mm absolute</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>1</td><td>!tvromoa</td><td colspan="2">-</td></tr>
</table>

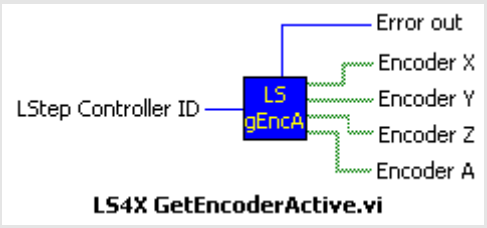
LSX_MoveRelTVRO			
Description:	Function for moving a relative vector from the current position (not for LSTEP express)		
Delphi:	function LSX_MoveRelTVRO(LSID: Integer; X, Y, Z, A: Double; Wait: LongBool): Integer;		
C++:	int MoveRelTVRO(double dX, double dY, double dZ, double dR, BOOL bWait);		
LabView:			
Parameter:	X, Y, Z, A	Relative position indication in the set axis dimension	
	Wait	Wait for the end of the movement True = Wait False = Do not wait	
Example:	LSX.MoveRelTVRO(10.0, 10.0, 10.0, 10.0, true);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!tvromor	-

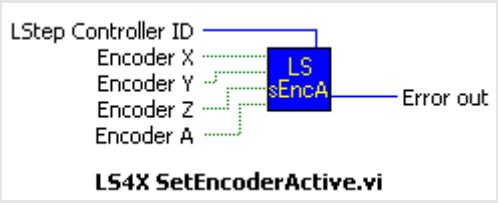
LSX_MoveRelTVROSingleAxis			
Description:	Function for relative positioning of a single TVRO axis (not for LSTEP express)		
Delphi:	function LSX_MoveRelTVROSingleAxis(LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;		
C++:	int MoveRelTVROSingleAxis(int lAxis,double dValue,BOOL Wait);		
LabView:			
Parameter:	Axis	Axis to be moved TVRO X = 1 TVRO Y = 2 …	
	Value	Relative position in the set axis dimension.	
	Wait	Wait for the end of the movement True = Wait False = Do not wait	
Example:	LSX.MoveRelTVROSingleAxis(3, 5.0); // Move additional Z-axis by 5mm in positive direction		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!tvromor	-

4.2.16 Encoder settings

LSX_ClearEncoder			
Description:	Function for setting the encoder counter to zero (not for LSTEP express)		
Delphi:	function LSX_ClearEncoder(LSID: Integer; lAxis: Integer): Integer;		
C++:	int ClearEncoder(int lAxis);		
LabView:			
Parameter:	lAxis	Axis whose encoder values are to be set to zero. X = 1 Y = 2 …	
Example:	LSX.ClearEncoder (2); //Set encoder counter of Y-axis to zero		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!clearhwcount	-

LSX_GetEncoder			
Description:	This function reads all encoder positions from the controller (not for LSTEP express)		
Delphi:	function LSX_GetEncoder(LSID: Integer; XP, YP, ZP, AP: Double): Integer;		
C++:	int GetEncoder(double *pdXP, double *pdYP, double *pdZP, double *pdAP);		
LabView:			
Parameter:	XP, YP, ZP, AP	Number of encoder increments, with quadruple interpolation	
Example:	LSX.GetEncoder(&XP, &YP, &ZP, &AP);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?hwcount	-

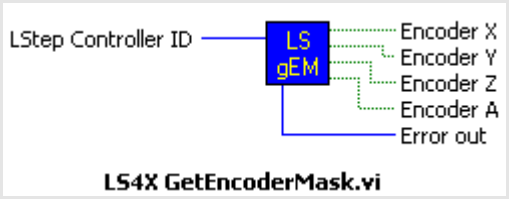
<table>
<tr><th colspan="4">LSX_GetEncoderActive</th></tr>
<tr><td>Description:</td><td colspan="3">Reads which encoders are activated after calibration
(not for LSTEP express)</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_GetEncoderActive(LSID: Integer; var Flags: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int GetEncoderActive(int *plFlags);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>Flags</td><td colspan="2">32-bit integer, with one bit mask in the bits 0-4
Bit 0 = X-axis
Bit 1 = Y-axis
…
Value 0 = Encoder is to be activated
Value 1 = Encoder is not to be activated</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.GetEncoderActive(&Flags);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>1</td><td>?encmask</td><td>-</td></tr>
</table>

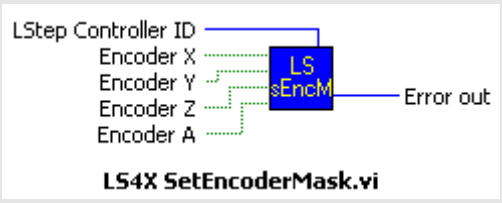
LSX_SetEncoderActive	
Description:	This function may be used to select the encoders to be activated after calibration.
Delphi:	function LSX_SetEncoderActive(LSID: Integer; Flags: Integer): Integer;
C++:	int SetEncoderActive(int lFlags);
LabView:	

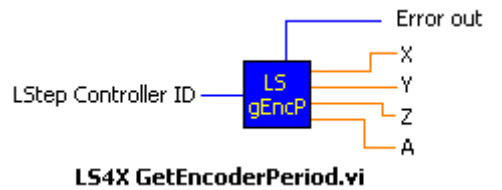
Parameter:	Flags	32-bit integer, with one bit mask in the bits 0-4 Bit 0 = X-axis Bit 1 = Y-axis … Value 0 = Encoder is not to be activated after the calibration Value 1 = Encoder is to be activated after the calibration	

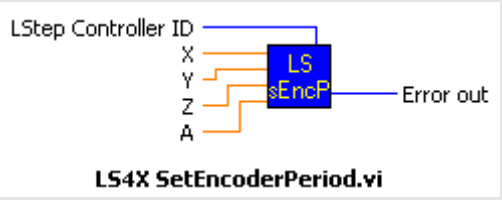
Example:	LSX.SetEncoderActive(0);
	// Deactivate all encoders
	LSX.SetEncoderMask(2);
	// Activate Y-axis encoder		

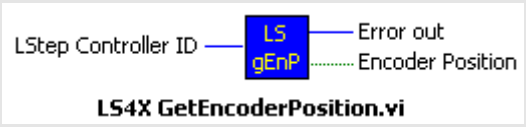
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?encmask	-

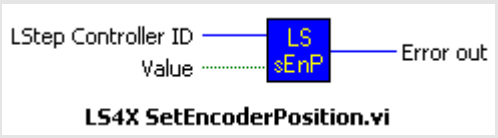
<table>
<tr><th colspan="4">LSX_GetEncoderMask</th></tr>
<tr><td>Description:</td><td colspan="3">Function for reading the encoder activation.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_GetEncoderMask(LSID: Integer; var Flags: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int GetEncoderMask(int *plFlags);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>Flags</td><td colspan="2">32-bit integer, with one bit mask in the bits 0-4
Bit 0 = X-axis
Bit 1 = Y-axis
…
Value 0 = Encoder was not identified or is not active
Value 1 = Encoder was identified or is active</td></tr>
<tr><td>Example:</td><td colspan="3">int EncMask;
LSX.GetEncoderMask(&EncMask);
if (EncMask & 2) ...
// If Y-axis encoder is connected + nactive</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>?enc</td><td>-</td></tr>
</table>

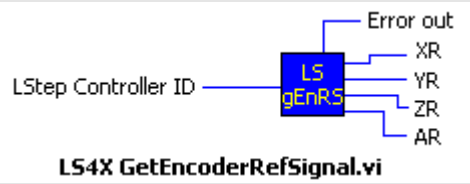
<table>
<tr><th colspan="4">LSX_SetEncoderMask</th></tr>
<tr><td>Description:</td><td colspan="3">Function for activating/deactivating the encoder
(not for LSTEP express)</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_SetEncoderMask(LSID: Integer; Value: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int SetEncoderMask(int lValue);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>Value</td><td colspan="2">32-bit integer, with one bit mask in the bits 0-4
Bit 0 = X-axis
Bit 1 = Y-axis
…
Value 0 = Deactivate encoder
Value 1 = Activate encoder</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.SetEncoderMask(0);
// Deactivate all encoders
LSX.SetEncoderMask(2);
// Activate Y-axis encoder</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>1</td><td>!enc</td><td>-</td></tr>
</table>

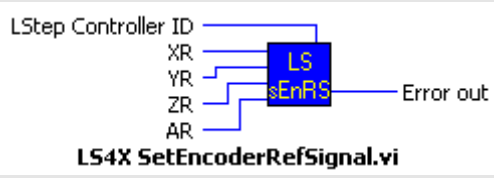
LSX_GetEncoderPeriod			
Description:	Function for reading the encoder period lengths.		
Delphi:	function LSX_GetEncoderPeriod(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetEncoderPeriod(double *pdX, double *pdY, double *pdZ, double *pdR);		
LabView:			
Parameter:	X, Y, Z, A	Set period lengths LSTEP 2000 series = m/s LSTP express series = Set dimension/s	
Example:	LSX.GetEncoderPeriod(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?encperiod	-

LSX_SetEncoderPeriod			
Description:	Function for setting the encoder period lengths.		
Delphi:	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, A	Period lengths to be set LSTEP 2000 series = m/s LSTEP express series = Set dimension/s	
Example:	LSX.SetEncoderPeriod(0.1, 0.1, 0.1, 0.1); // Encoder period length of all axes is 0.1mm		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!encperiod	-

LSX_GetEncoderPosition			
Description:	Reading of source data settings of position indication.		
Delphi:	function LSX_GetEncoderPosition(LSID: Integer; Value: LongBool): Integer;		
C++:	int GetEncoderPosition(BOOL *pbValue);		
LabView:			
Parameter:	Value	Source of position indication True = Use encoder values for position inquiry False = Do not use encoder values for position inquiry	
Example:	LSX.GetEncoderPosition(&Value);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?encpos	-

LSX_SetEncoderPosition			
Description:	Setting of source data of position indication.		
Delphi:	function LSX_SetEncoderPosition(LSID: Integer; Value: LongBool): Integer;		
C++:	int SetEncoderPosition(BOOL fValue);		
LabView:			
Parameter:	Value	Source of position indication True = Use encoder values for position inquiry False = Do not use encoder values for position inquiry	
Example:	LSX.SetEncoderPosition(true);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!encpos	-

LSX_GetEncoderRefSignal			
Description:	Reads whether the reference signal of the encoder is evaluated during calibration.		
Delphi:	function LSX_GetEncoderRefSignal(LSID: Integer; var XR, YR, ZR, AR: Integer): Integer;		
C++:	int GetEncoderRefSignal(int *plXR, int *plYR, int *plZR, int *plAR);		
LabView:			
Parameter:	X, Y, Z, A	Set value 1 = The reference signal is evaluated during calibration 0 = The reference signal is not evaluated	
Example:	LSX.GetEncoderRefSignal(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?encref	-

LSX_SetEncoderRefSignal			
Description:	Function for activating the evaluation of the reference signal of an encoder.		
Delphi:	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, A	Set value 1 = The reference signal is evaluated during calibration 0 = The reference signal is not evaluated	
Example:	LSX.SetEncoderRefSignal(1, 1, 0, 0); /* The reference signal of the encoders x and y is evaluated during calibration. */		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!encref	-

LSX_GetEncDir			
Description:	Function for inquiring the counting directions of the position encoder		
Delphi:	function LSX_GetEncDir(LSID: Integer; var X,Y,Z,A LongBool): Integer;		
C++:	int GetEncDir (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	False = No change of the rotating direction True = Change oft he rotating direction	
Example:	LSX.GetEncDir(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Com-mandl	Activation (LSTEP express series)
	2	?encdir	-

LSX_SetEncDir			
Description:	Adjusts the counting direction oft he position encoder		
Delphi:	function LSX_SetEncDir(LSX: Integer; X,Y,Z,A: LongBool): Integer;		
C++:	int SetEncDir (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	False = No change of the rotating direction True = Change of the rotating direction	
Example:	LSX.SetEncDir(True,True,False,False); /* Change in the rotating direction oft the first and the second axis */		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!encdir	-

LSX_GetEncPolePairs			
Description:	Function for inquiring the amount of position encoder pole pairs per rotation.		
Delphi:	function LSX_GetEncPolePairs(LSID:Integer; var X,Y,Z,A: Integer): Integer;		
C++:	int GetEncPolePairs (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Number of position encoder pole pairs per rotation	
Example:	LSX.GetEncPolePairs(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?encpolepairs	-

LSX_SetEncPolePairs			
Description:	Sets the amount of position encoder pole pairs per rotation.		
Delphi:	function LSX_SetEncPolePairs(LSID:Integer;X,Y,Z,A: Integer): Integer;		
C++:	int SetEncPolePairs (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Number of position encoder pole pairs per rotation	
Example:	LSX.SetEncPolePairs(50,50,50,50);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!encpolepairs	-

LSX_GetEnctoAxis		
Description:	Function for inquiring the encoder inputs assigned to the position encoder interpretation	
Delphi:	function LSX_GetEnctoAxis(LSID:Integer; var X,Y,Z,A: Integer): Integer;	
C++:	int GetEnctoAxis (int *plX, int *plY, int *plZ, int *plA);	
LabView:	Not supported	
Parameter:	X, Y, Z, A	0 = No encoder input 1 = QEP1 (TTL level, MFP) 2 = QEP2 (TTL level, MFP) 3 = QEP3 (TTL or RS422 level, optional) 4 = QEP4 (TTL or RS422 level, optional) 5 = QEP5 (TTL or RS422 level, optional) 6 = QEP6 (TTL or RS422 level, optional 7 = ENC1 (sine / cosine signals, optional) 8 = ENC2 (sine / cosine signals, optional) 9 = ENC3 (sine / cosine signals, optional) 10 = ENC4 (sine / cosine signals, optional) 11 = ENC5 (sine / cosine signals, optional) 12 = ENC6 (sine / cosine signals, optional) 13 = ENC7 (sine / cosine signals, optional) 14 = ENC8 (sine / cosine signals, optional)
Example:	LSX.GetEnctoAxis(&X,&Y,&Z,&A);	
Other:	Compatibility	"SendString"Command	Activation(LSTEP express series)
	2	?enctoaxis	-

<table>
<tr><th colspan="5">LSX_SetEnctoAxis</th></tr>
<tr><td>Description:</td><td colspan="4">Assigns encoder inputs to the position encoder interpretations of all axes (position encoder 1).</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_SetEnctoAxis(LSID:Integer;X,Y,Z,A: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int SetEnctoAxis (int lX, int lY, int lZ, int lA);</td></tr>
<tr><td>LabView:</td><td colspan="4">Not supported</td></tr>
<tr><td>Parameter:</td><td>X, Y, Z, A</td><td colspan="3">0 = No encoder input
1 = QEP1 (TTL level, MFP)
2 = QEP2 (TTL level, MFP)
3 = QEP3 (TTL or RS422 level, optional)
4 = QEP4 (TTL or RS422 level, optional)
5 = QEP5 (TTL or RS422 level, optional)
6 = QEP6 (TTL or RS422 level, optional
7 = ENC1 (sine / cosine signals, optional)
8 = ENC2 (sine / cosine signals, optional)
9 = ENC3 (sine / cosine signals, optional)
10 = ENC4 (sine / cosine signals, optional)
11 = ENC5 (sine / cosine signals, optional)
12 = ENC6 (sine / cosine signals, optional)
13 = ENC7 (sine / cosine signals, optional)
14 = ENC8 (sine / cosine signals, optional)</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.SetEnctoAxis(0,0,0,0);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td colspan="2">“SendString” Command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>2</td><td colspan="2">!enctoaxis</td><td>validconfig</td></tr>
</table>

<table>
<tr><th colspan="5">LSX_GetEncType</th></tr>
<tr><td>Description:</td><td colspan="4">Function for inquiring the psoition encoder type.</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_GetEncType(LSID:Integer; var X,Y,Z,A: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int GetEncType (int *plX, int *plY, int *plZ, int *plA);</td></tr>
<tr><td>LabView:</td><td colspan="4">Not supported</td></tr>
<tr><td>Parameter:</td><td>X, Y, Z, A</td><td colspan="3">0 = No position encoder
1 = Motor side rotative 1Vss encoder system
2 = Motor side rotative 11µA encoder system
3 = Motor side rotative MR encoder system
4 = Motor side rotative TTL encoder system
5 = Linear 1Vss encoder system
6 = Linear 11µA encoder system
7 = Linear MR encoder system
8 = Linear TTL encoder system
9 = Drive side rotative 1Vss encoder system
10 = Drive side rotative 11µA encoder system
11 = Drive side rotative MR encoder system
12 = Drive side rotative TTL encoder system</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.GetKommEncType(&X,&Y,&Z,&A);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” Command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>2</td><td>?enctype</td><td colspan="2">-</td></tr>
</table>

LSX_SetEncType			
Description:	Sets the position encoder type.		
Delphi:	function LSX_SetEncType(LSID:Integer;X,Y,Z,A: Integer): Integer;		
C++:	int SetEncType (int lX, int lY, int lZ, int lA) ;		
LabView:	Not supported		
Parameter:	X, Y, Z, A	0 = No position encoder 1 = Motor side rotative 1Vss encoder system 2 = Motor side rotative 11µA encoder system 3 = Motor side rotative MR encoder system 4 = Motor side rotative TTL encoder system 5 = Linear 1Vss encoder system 6 = Linear 11µA encoder system 7 = Linear MR encoder system 8 = Linear TTL encoder system 9 = Drive side rotative 1Vss encoder system 10 = Drive side rotative 11µA encoder system 11 = Drive side rotative MR encoder system 12 = Drive side rotative TTL encoder system	
Example:	LSX.SetEnctype(0,1,2,3);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!enctype	ValidConfig

LSX_GetKommMode			
Description:	Shows the commutation mode in servo operation		
Delphi:	function LSX_GetKommMode(LSID:Integer; var X,Y,Z,A:LongBool): Integer;		
C++:	int GetEncType (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Assigned Value: False = Use seperate encoder input for commutation True = Use position encoder values for commutation	
Example:	LSX.GetKommMode (&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?kommmode	-

LSX_SetKommMode			
Description:	Sets the commutation mode in servo operation		
Delphi:	function LSX_SetKommMode (LSID:Integer;X,Y,Z,A:LongBool): Integer;		
C++:	int SetKommMode (int lX, int lY, int lZ, int lA) ;		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Assigned Value: False = Use seperate encoder input for commutation True = Use position encoder values for commutation	
Example:	LSX.SetKommMode (False,False,False,False);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!kommmode	-

LSX_GetKommEncDir			
Description:	Shows the commutation encoder counting direction		
Delphi:	function LSX_GetKommEncDir (LSID:Integer; var X,Y,Z,A:LongBool): Integer;		
C++:	int GetKommEncDir (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Assigned Value: False = No change of the rotating direction True = Change of the rotating direction	
Example:	LSX.GeKommtEncDir(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?kommencdir	-

LSX_SetKommEncDir			
Description:	Sets the commutation encoder counting direction.		
Delphi:	function LSX_SetKommEncDir (LSID:Integer;X,Y,Z,A:LongBool): Integer;		
C++:	int SetKommEncDir (BOOL bX, BOOL bY,BOOL bZ,BOOL bA) ;		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Assigned Value: False = No change of the rotating direction True = Change of the rotating direction	
Example:	LSX.SetEncDir(True,True,True,True);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!kommencdir	-

LSX_GetKommEncPolePairs			
Description:	Shows the commutation encoder signal period.		
Delphi:	function LSX_GetKommEncPolePairs (LSID:Integer; var X,Y,Z,A: Integer): Integer;		
C++:	int GetKommEncPolePairs (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Commutation encoder pole pairs per rotation	
Example:	LSX.GetKommEncPolePairs (&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?kommencpolepairs	-

LSX_SetKommEncPolePairs			
Description:	Sets the commutation encoder signal period.		
Delphi:	function LSX_SetKommEncPolePairs (LSID:Integer;X,Y,Z,A: Integer): Integer;		
C++:	int SetKommEncPolePairs (int lX, int lY, int lZ, int lA) ;		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Assigned Value: Commutation encoder signal period	
Example:	LSX.SetKommEncPolePairs (50,10,2500,4000);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!kommencpolepairs	-

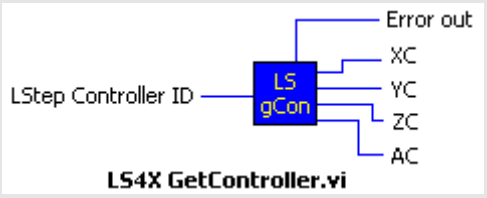
LSX_GetKommEnctoAxis			
Description:	Shows the assigned encoder inputs to the commutation encoder interpretations of the axes.		
Delphi:	function LSX_GetKommEnctoAxis (LSID:Integer; var X,Y,Z,A: Integer): Integer;		
C++:	int GetKommEnctoAxis (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Assigned Value: 0 = No encoder input 1 = QEP1 (TTL level, MFP) 2 = QEP2 (TTL level, MFP) 3 = QEP3 (TTL or RS422 level, optional) 4 = QEP4 (TTL or RS422 level, optional) 5 = QEP5 (TTL or RS422 level, optional) 6 = QEP6 (TTL or RS422 level, optional) 7 = ENC1 (sine / cosine signals, optional) 8 = ENC2 (sine / cosine signals, optional) 9 = ENC3 (sine / cosine signals, optional) 10 = ENC4 (sine / cosine signals, optional) 11 = ENC5 (sine / cosine signals, optional) 12 = ENC6 (sine / cosine signals, optional) 13 = ENC7 (sine / cosine signals, optional) 14 = ENC8 (sine / cosine signals, optional)	
Example:	LSX.GetKommEnctoAxis (&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?kommenctoaxis	-

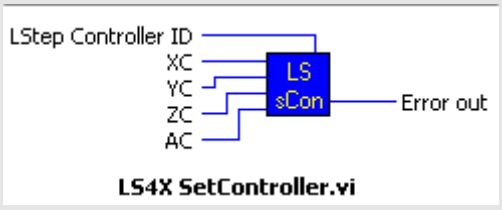
LSX_SetKommEnctoAxis			
Beschreibung:	Assigns encoder inputs to the commutation encoder interpretations of the axes.		
Delphi:	function LSX_SetKommEnctoAxis (LSID:Integer;X,Y,Z,A: Integer): Integer;		
C++:	int SetKommEnctoAxis (int lX, int lY, int lZ, int lA) ;		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Assigned Value: 0 = No encoder input 1 = QEP1 (TTL level, MFP) 2 = QEP2 (TTL level, MFP) 3 = QEP3 (TTL or RS422 level, optional) 4 = QEP4 (TTL or RS422 level, optional) 5 = QEP5 (TTL or RS422 level, optional) 6 = QEP6 (TTL or RS422 level, optional) 7 = ENC1 (sine / cosine signals, optional) 8 = ENC2 (sine / cosine signals, optional) 9 = ENC3 (sine / cosine signals, optional) 10 = ENC4 (sine / cosine signals, optional) 11 = ENC5 (sine / cosine signals, optional) 12 = ENC6 (sine / cosine signals, optional) 13 = ENC7 (sine / cosine signals, optional) 14 = ENC8 (sine / cosine signals, optional)	
Example:	LSX.SetKommEnctoAxis (0,1,2,3);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!kommenctoaxis	-

LSX_GetKommEncType			
Description:	Shows the commutation encoder type		
Delphi:	function LSX_GetKommEncType(LSID:Integer; var X,Y,Z,A: Integer): Integer;		
C++:	int GetKommEncType (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Nicht unterstützt		
Parameter:	X, Y, Z, A	Assigned Value: 0 = No position encoder 1 = Motor side rotative 1Vss encoder system 2 = Motor side rotative 11µA encoder system 3 = Motor side rotative TTL encoder system 5 = Linear 1Vss encoder system 6 = Linear 11µA encoder system 7 = Linear MR encoder system 8 = Linear TTL encoder system 9 = Drive side rotative 1Vss encoder system 10 = Drive side rotative 11µA encoder system 11 = Drive side rotative rotative MR encoder system 12 = Drive side rotative TTL encoder system	
Example:	LSX.GetKommEncType(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?kommenctype	-

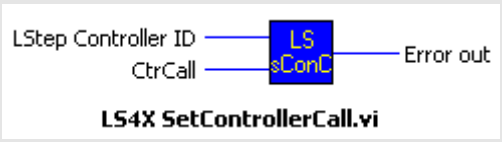
LSX_SetKommEncType			
Description:	Sets the commutation encoder type		
Delphi:	function LSX_SetKommEncType(LSID:Integer;X,Y,Z,A: Integer): Integer;		
C++:	int SetKommEncType (int lX, int lY, int lZ, int lA) ;		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Assigned Value: 0 = No position encoder 1 = Motor side rotative 1Vss encoder system 2 = Motor side rotative 11µA encoder system 3 = Motor side rotative TTL encoder system 5 = Linear 1Vss encoder system 6 = Linear 11µA encoder system 7 = Linear MR encoder system 8 = Linear TTL encoder system 9 = Drive side rotative 1Vss encoder system 10 = Drive side rotative 11µA encoder system 11 = Drive side rotative rotative MR encoder system 12 = Drive side rotative TTL encoder system	
Example:	LSX.SetKommEnctype(0,1,2,3);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!kommenctype	-

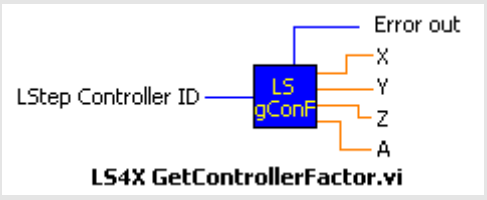
4.2.17 Controller settings

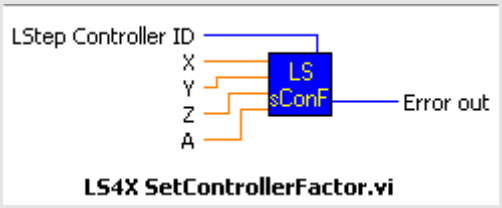
LSX_GetController			
Description:	Function for reading the controller mode (not for LSTEP express)		
Delphi:	function LSX_GetController(LSID: Integer; var XC, YC, ZC, AC: Integer): Integer;		
C++:	int GetController(int *plXC, int *plYC, int *plZC, int *plAC);		
LabView:			
Parameter:	X, Y, Z, A	Set controller mode 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:	LSX.GetController(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?ctr	-

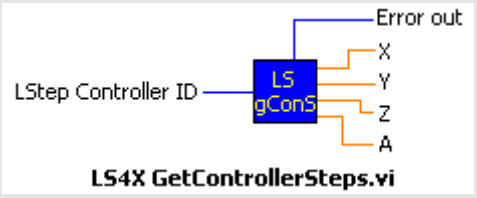
LSX_SetController			
Description:	Function for setting the controller mode. (not for LSTEP express)		
Delphi:	function LSX_SetController(LSID: Integer; XC, YC, ZC, AC: Integer): Integer;		
C++:	int SetController(int lXC,int lYC,int lZC,int lAC);		
LabView:			
Parameter:	X, Y, Z, A	Set controller mode 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:	LSX.SetController(1, 2, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!ctr	-

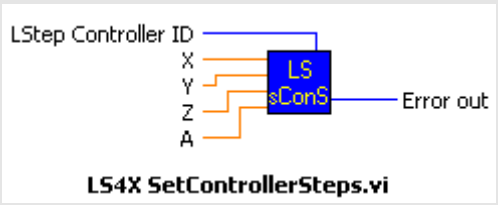
LSX_GetControllerCall			
Description:	The function delivers the set controller call time (not for LSTEP express)		
Delphi:	function LSX_GetControllerCall(LSID: Integer; var CtrCall: Integer): Integer;		
C++:	int GetControllerCall(int *plCtrCall);		
LabView:			
Parameter:	CtrCall	Controller call time in ms	
Example:	LSX.GetControllerCall(&CtrCall); // CtrCall = 10 means: Controller call every 10 ms		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?ctrc	-

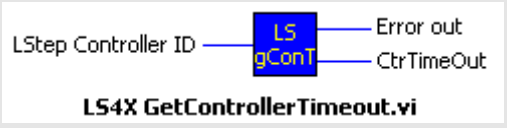
LSX_SetControllerCall			
Description:	Function for setting the controller call time (not for LSTEP express)		
Delphi:	function LSX_SetControllerCall(LSID: Integer; CtrCall: Integer): Integer;		
C++:	int SetControllerCall(int lCtrCall);		
LabView:			
Parameter:	CtrCall	Controller call time in ms	
Example:	LSX.SetControllerCall(10);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!ctrc	-

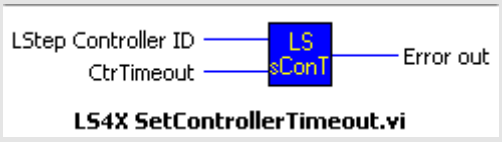
LSX_GetControllerFactor			
Description:	Function for reading the controller factors. See documentation of LSTEP 2000 series (not for LSTEP express)		
Delphi:	function LSX_GetControllerFactor(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetControllerFactor(double *pdX, double *pdY, double *pdZ, double *pdR);		
LabView:			
Parameter:	X, Y, Z, A	Set controller factors	
Example:	LSX.GetControllerFactor(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?ctrf	-

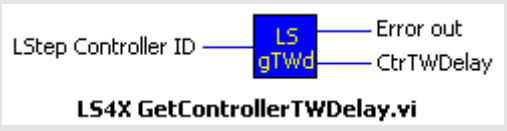
LSX_SetControllerFactor			
Description:	Function for setting the controller factors. See documentation of LSTEP 2000 series (not for LSTEP express)		
Delphi:	function LSX_SetControllerFactor(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetControllerFactor(double dX,double dY,double dZ,double dA);		
LabView:			
Parameter:	X, Y, Z, A	Controller factors to be set	
Example:	LSX.SetControllerFactor(X, Y, Z, A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!ctrf	-

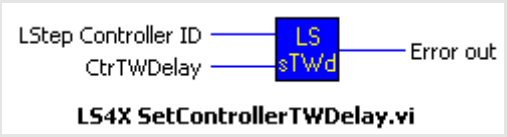
LSX_GetControllerSteps			
Description:	This function delivers the set controller step length (not for LSTEP express)		
Delphi:	function LSX_GetControllerSteps(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetControllerSteps(double *pdX, double *pdY, double *pdZ, double *pdR);		
LabView:			
Parameter:	X, Y, Z, A	Controller step length in the set axis dimension.	
Example:	LSX.GetControllerSteps(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?ctrs	-

LSX_SetControllerSteps			
Description:	Function for setting the controller step length (not for LSTEP express)		
Delphi:	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, A	Controller step length in the set axis dimension.	
Example:	LSX.SetControllerSteps(4, 5, 7, 9);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!ctrs	-

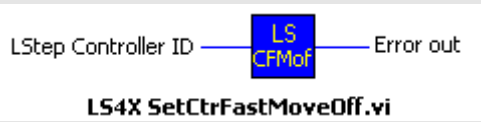
LSX_GetControllerTimeout			
Description:	Function for reading the controller timeouts after which a travel command returns with an error message (error code 4013), if the controller could not definitively find a position (not for LSTEP express)		
Delphi:	function LSX_GetControllerTimeout(LSID: Integer; var ACtrTimeout: Integer): Integer;		
C++:	int GetControllerTimeout(int *plACtrTimeout);		
LabView:			
Parameter:	ACtrTimeout	Set timeout in ms	
Example:	LSX.GetControllerTimeout(&ACtrTimeout);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?ctrt	-

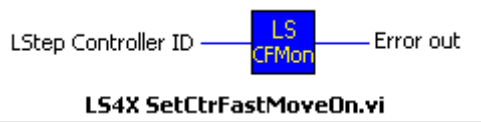
LSX_SetControllerTimeout			
Description:	Function for setting the controller timeouts after which a travel command returns with an error message (error code 4013), if the controller could not definitively find a position (not for LSTEP express)		
Delphi:	function LSX_SetControllerTimeout(LSID: Integer; ACtrTimeout: Integer): Integer;		
C++:	int SetControllerTimeout(int ACtrTimeout);		
LabView:			
Parameter:	ACtrTimeout	Timeout to be set in ms	
Example:	LSX.SetControllerTimeout(500);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!ctrt	-

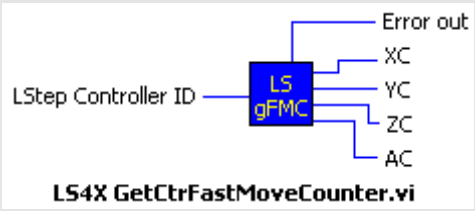
LSX_GetControllerTWDelay			
Description:	Reading of controller delay (not for LSTEP express)		
Delphi:	function LSX_GetControllerTWDelay(LSID: Integer; var CtrTWDelay: Integer): Integer;		
C++:	int GetControllerTWDelay(int *plCtrTWDelay);		
LabView:			
Parameter:	CtrTWDelay	Controller delay in ms	
Example:	LSX.GetControllerTWDelay(&CtrTWDelay);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?ctrd	-

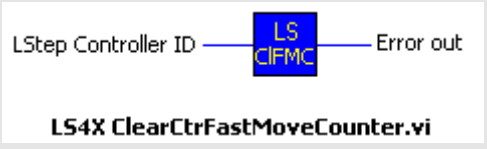
LSX_SetControllerTWDelay			
Description:	Function for setting the controller delay (not for LSTEP express)		
Delphi:	function LSX_SetControllerTWDelay(LSID: Integer; CtrTWDelay: Integer): Integer;		
C++:	int SetControllerTWDelay(int lCtrTWDelay);		
LabView:			
Parameter:	CtrTWDelay	Controller delay to be set in ms	
Example:	LSX.SetControllerTWDelay(0); // Controller delay Off		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!ctrd	-

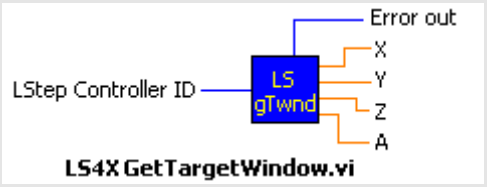
LSX_GetCtrFastMove			
Description:	Reads the setting of the Fast Move function (not for LSTEP express)		
Delphi:	function LSX_GetCtrFastMoveOff(LSID: Integer; var bActive: LongBool): Integer;		
C++:	int GetCtrFastMove(BOOL *pbActive);		
LabView:			
Parameter:	bActive	Set value True = Fast Move Function is active False = Fast Move Function is inactive	
Example:	LSX.GetCtrFastMoveOff(&bActive);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?ctrfm	-

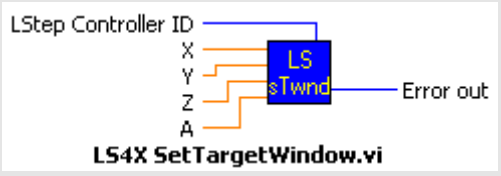
LSX_SetCtrFastMoveOff			
Description:	Function for deactivating the Fast Move function (not for LSTEP express)		
Delphi:	function LSX_SetCtrFastMoveOff(LSID: Integer): Integer;		
C++:	int SetCtrFastMoveOff();		
LabView:			
Parameter:	-		
Example:	LSX.SetCtrFastMoveOff();		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!ctrfm	-

LSX_SetCtrFastMoveOn			
Description:	Activation of Fast Move function, i.e. a new vector is started if a controller difference is larger than the capture range (not for LSTEP express)		
Delphi:	function LSX_SetCtrFastMoveOn(LSID: Integer): Integer;		
C++:	int SetCtrFastMoveOn();		
LabView:			
Parameter:	-		
Example:	LSX.SetCtrFastMoveOn();		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!ctrfm	-

LSX_GetCtrFastMoveCounter			
Description:	This function delivers the values of the Fast Move counter. If the controller difference is larger than the capture range, a new vector is started and the corresponding counter is extended by one. (not for LSTEP express)		
Delphi:	function LSX_GetCtrFastMoveCounter(LSID: Integer; var XC, YC, ZC, AC: Integer): Integer;		
C++:	int GetCtrFastMoveCounter(int *plXC, int *plYC, int *plZC, int *plAC);		
LabView:			
Parameter:	XC, YC, ZC, AC	Number of performed Fast Move functions	
Example:	LSX.SetCtrFastMoveCounter(&XC, &YC, &ZC, &AC);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	?ctrfmc	-

LSX_ClearCtrFastMoveCounter			
Description:	If the controller difference is larger than the capture range, a new vector is started and the corresponding counter is extended by one. This function sets the Fast Move Counters of all axes to Zero. (not for LSTEP express)		
Delphi:	function LSX_ClearCtrFastMoveCounter(LSID: Integer): Integer;		
C++:	int ClearCtrFastMoveCounter();		
LabView:			
Parameter:			
Example:	LSX.ClearCtrFastMoveCounter;		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	1	!ctrfmc	-

LSX_GetTargetWindow			
Description:	Reads the target windows of all axes.		
Delphi:	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, A	Target window depends on the set axis dimension.	
Example:	LSX.GetTargetWindow(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?twi	-

LSX_SetTargetWindow			
Description:	Function for setting the target window.		
Delphi:	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, A	Target window to be set in the axis dimension.	
Example:	LSX.SetTargetWindow(1.0, 0.002, 1.0, 1.0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!twi	ValidPar / ValidConfig

LSX_GetDeviationRange			
Description:	Shows the maximal permittable contouring error (position controller deviation)		
Delphi:	function LSX_GetDeviationRange(LSID: Integer;var X,Y,Z,A: Double): Integer;		
C++:	int GetDeviationRange (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Preset contouring error	
Example:	LSX.GetDeviationRange(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?deviationrange	-

LSX_SetDeviationRange			
Description:	Sets the maximal permittable contouring error (position controller deviation)		
Delphi:	function LSX_SetDeviationRange(LSID: Integer;X,Y,Z,A : Double): Integer;		
C++:	int SetDeviationRange (double dX, double dY, double dZ, double dA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Preset contouring error	
Example:	LSX.SetDeviationRange(1.0, 0.002, 1.0, 1.0);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!deviationrange	ValidPar / ValidConfig

LSX_GetDeviationTime			
Description:	Shows the time after which the contouring error check is tripped when the maximal permittable contouring error is exceeded.		
Delphi:	function LSX_GetDeviationTime(LSID: Integer;var X,Y,Z,A: Integer): Integer;		
C++:	int GetDeviationTime (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Time for contouring error check	
Example:	LSX.GetDeviationTime(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?deviationtime	-

LSX_SetDeviationTime			
Description:	Sets the time after which the contouring error check is tripped when the maximal permittable contouring error is exceeded.		
Delphi:	function LSX_SetDeviationTime(LSID: Integer; X,Y,Z,A : Integer): Integer;		
C++:	int SetDeviationTime(int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Time for contouring error check	
Example:	LSX.SetDeviationTime(1.0, 0.002, 1.0, 1.0);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!deviationtime	ValidPar / ValidConfig

LSX_GetDeviationCheck			
Description:	Shows if the contouring error check is active		
Delphi:	function LSX_GetDeviationCheck (LSID: Integer;var X,Y,Z,A: LongBool):Integer;		
C++:	int GetDeviationCheck(BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	State of the contouring error check	
Example:	LSX.GetDeviationCheck(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?deviationcheck	-

LSX_SetDeviationCheck			
Description:	Activates/Deactivates the contouring error check		
Delphi:	function LSX_SetDeviationCheck (LSID: Integer;X,Y,Z,A: LongBool):Integer;		
C++:	int SetDeviationCheck (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	State of the contouring error check	
Example:	LSX.SetDeviationCheck(True,True,True,True);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!deviationcheck	ValidPar / ValidConfig

LSX_GetDeviationValue			
Description:	Reads the deviation value of the axes.		
Delphi:	function LSX_GetDeviationValue (LSID: Integer;X,Y,Z,A: Double):Integer;		
C++:	int GetDeviationValue (BOOL bX, BOOL bY, BOOL bZ, BOOL bA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Deviationvalue	
Example:	LSX.GetDeviationValue(&X,&Y,&Z,&A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?deviationvalue	-

LSX_GetPoscon			
Description:	Shows the state of the position controllers dependant on the axis-specific settings		
Delphi:	function LSX_GetPoscon(LSID: Integer;var Enable: Boolean): Integer;		
C++:	int GetPoscon (BOOL *pbEnable);		
LabView:	Not supported		
Parameter:	Enable	State of the position controller	
Example:	LSX.GetPoscon(&Enable);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?poscon	-

LSX_SetPoscon			
Description:	Activates/Deactivates the position controllers dependant on the axis-specific settings		
Delphi:	function LSX_SetPoscon(LSID: Integer; Enable: Boolean): Integer;		
C++:	int SetPoscon (BOOL bEnable);		
LabView:	Not supported		
Parameter:	Enable	State of the position controller	
Example:	LSX.SetPoscon(True);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!poscon	ValidPar/ ValidConfig

LSX_GetPosConKP			
Description:	Command for reading the position controller proportional part in % for servo and stepping motor		
Delphi:	function LSX_GetPosConKP (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetPosConKP (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Position controller proportional part, range: 0 to 5000%	
Example:	LSX.GetPosConKP (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?posconkp	-

LSX_SetPosConKP			
Description:	Command for setting the position controller proportional part in % for servo and stepping motor		
Delphi:	function LSX_SetPosConKP (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetPosConKP (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Position controller proportional part, range: 0 to 5000%	
Example:	LSX.SetPosConKP (40, 20, 80, 80);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!posconkp	ValidPar / ValidConfig

LSX_GetPosConAdaptiveKP			
Description:	Command for reading the adaptive position controller proportional part in % for stepping motor only		
Delphi:	function LSX_GetPosConAdaptiveKP (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetPosConAdaptiveKP (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Position controller proportional part, range: 0 to 5000%	
Example:	LSX.GetPosConAdaptiveKP (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?posconadaptivekp	-

LSX_SetPosConAdaptiveKP			
Description:	Command for setting the adaptive position controller proportional part in % for stepping motor only		
Delphi:	function LSX_SetPosConAdaptiveKP (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetPosConAdaptiveKP (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Position controller proportional part, range: 0 to 5000%	
Example:	LSX.SetPosConAdaptiveKP (40, 20, 80, 80);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!posconkp	ValidPar / ValidConfig

LSX_GetPosConKI			
Description:	Command for reading the position controller integral part in % for servo and stepping motor		
Delphi:	function LSX_GetPosConKI (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetPosConKI (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Position controller integral part, range: 0 to 5000%	
Example:	LSX.GetPosConKI (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?posconki	-

LSX_SetPosConKI			
Description:	Command for setting the position controller integral part in % for servo and stepping motor		
Delphi:	function LSX_SetPosConKI (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetPosConKI (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Position controller integral part, range: 0 to 5000%	
Example:	LSX.SetPosConKI (75, 60, 90, 90);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!posconki	ValidPar / ValidConfig

LSX_GetPosConAdaptiveKI			
Description:	Command for reading the adaptive position controller integral part in % for stepping motor only		
Delphi:	function LSX_GetPosConAdaptiveKI (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetPosConAdaptiveKI (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Preset position controller integral part, range: 0 to 5000%	
Example:	LSX.GetPosConAdaptiveKI (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?posconadaptiveki	-

LSX_SetPosConAdaptiveKI			
Description:	Command for setting the adaptive position controller integral part in % for stepping motor only		
Delphi:	function LSX_SetPosConAdaptiveKI (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetPosConAdaptiveKI (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Preset position controller integral part, range: 0 to 5000%	
Example:	LSX.SetPosConAdaptiveKI (40, 20, 80, 80);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!posconki	ValidPar / ValidConfig

LSX_GetPosConOutPass			
Description:	Command for reading the position controller output filter time constant for servo and stepping motor		
Delphi:	function LSX_GetPosConOutPass (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetPosConOutPass (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	position controller output filter time constant, range: 0 bis 100000 µs	
Example:	LSX.GetPosConOutPass (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?posconoutpass	-

LSX_SetPosConOutPass			
Description:	Command for setting the position controller output filter time constant for servo and stepping motor		
Delphi:	function LSX_SetPosConOutPass (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetPosConOutPass (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	position controller output filter time constant, range: 0 bis 100000 µs	
Example:	LSX.SetPosConOutPass (200, 500, 200, 500);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!posconoutpass	ValidPar / ValidConfig

LSX_GetPosConAdaptiveOutPass			
Description:	Command for reading the position controller output filter time constant for stepping motor only		
Delphi:	function LSX_GetPosConAdaptiveOutPass (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetPosConAdaptiveOutPass (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	position controller output filter time constant, range: 0 bis 100000 µs	
Example:	LSX.GetPosConAdaptiveOutPass (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?posconadaptiveoutpass	-

LSX_SetPosConAdaptiveOutPass			
Description:	Command for setting the position controller output filter time constant for stepping motor only		
Delphi:	function LSX_SetPosConAdaptiveOutPass (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetPosConAdaptiveOutPass (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	position controller output filter time constant, range: 0 bis 100000 µs	
Example:	LSX.SetPosConAdaptiveOutPass (200, 500, 200, 500);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!posconadaptiveoutpass	ValidPar / ValidConfig

LSX_GetPosConNominalSpeed			
Description:	Command for reading the nominal velocity/speed. Unit depends on the selected dimension. Parameters are only required for adaptive stepping motor position controller, PosCon… control parameters (e.g. PosConKP) relate to PosConNominalSpeed		
Delphi:	function LSX_GetPosConNominalSpeed (LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetPosConNominalSpeed (double *plX, double *plY, double *plZ, double *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	nominal velocity/speed	
Example:	LSX.GetPosConNominalSpeed (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?posconnominalspeed	-

LSX_SetPosConNominalSpeed			
Description:	Command for setting the nominal velocity/speed. Unit depends on the selected dimension. Parameters are only required for adaptive stepping motor position controller, PosCon…control parameters (e.g. PosConKP) relate to PosConNominalSpeed		
Delphi:	function LSX_SetPosConNominalSpeed (LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetPosConNominalSpeed (double lX, double lY, double lZ, double lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	nominal velocity/speed	
Example:	LSX.SetPosConNominalSpeed (2.5, 2.5, 2.5, 2.5);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!posconnominalspeed	ValidPar / ValidConfig

LSX_GetPosConAdaptiveSpeed			
Description:	Command for reading the adaptive velocity/speed. Unit depends on the selected dimension. Parameters are only required for adaptive stepping motor position controller, PosConAdaptive… control parameters (e.g. PosConAdaptiveKP) relate to PosConAdaptiveSpeed		
Delphi:	function LSX_GetPosConAdaptiveSpeed (LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetPosConAdaptiveSpeed (double *plX, double *plY, double *plZ, double *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	adaptive velocity/speed	
Example:	LSX.GetPosConAdaptiveSpeed (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?posconadaptivespeed	-

LSX_SetPosConAdaptiveSpeed			
Description:	Command for setting the adaptive velocity/speed. Unit depends on the selected dimension. Parameters are only required for adaptive stepping motor position controller, PosConAdaptive… control parameters (e.g. PosConAdaptiveKP) relate to PosConAdaptiveSpeed		
Delphi:	function LSX_SetPosConAdaptiveSpeed (LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetPosConAdaptiveSpeed (double lX, double lY, double lZ, double lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	adaptive velocity/speed	
Example:	LSX.SetPosConAdaptiveSpeed (2.5, 2.5, 2.5, 2.5);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!posconadaptivespeed	ValidPar / ValidConfig

LSX_GetPosConEnable			
Description:	Command for reading the axis-specific selection of position controller operation mode		
Delphi:	function LSX_GetPosConEnable (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetPosConEnable (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	position controller operation mode 0: position controller disabled 1: position controllerenabled (servomotor and stepping motor) 2: adaptiver position controller enabled (for stepping motor only)	
Example:	LSX.GetPosConEnable (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?posconenable	-

LSX_SetPosConEnable			
Description:	Command for setting the axis-specific selection of position controller operation mode		
Delphi:	function LSX_SetPosConEnable (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetPosConEnable (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	position controller operation mode 0: position controller disabled 1: position controller enabled (servomotor and stepping motor) 2: adaptiver position controller enabled (for stepping motor only)	
Example:	LSX.SetPosConEnable (0, 2, 1, 1);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!posconenable	ValidPar / ValidConfig

LSX_GetSpeedConKP			
Description:	Command for reading the speed controller proportional part in %		
Delphi:	function LSX_GetSpeedConKP (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetSpeedConKP (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Speed controller proportional part, range: 0 to 5000%	
Example:	LSX.GetSpeedConKP (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?speedconkp	-

LSX_SetSpeedConKP			
Description:	Command for setting the speed controller proportional part in %		
Delphi:	function LSX_SetSpeedConKP (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetSpeedConKP (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Speed controller proportional part, range: 0 to 5000%	
Example:	LSX.SetSpeedConKP (40, 20, 80, 80);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!speedconkp	ValidPar / ValidConfig

LSX_GetSpeedConKI			
Description:	Command for reading the speed controller integral part in %		
Delphi:	function LSX_GetSpeedConKI (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetSpeedConKI (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Speed controller integral part, range: 0 to 5000%	
Example:	LSX.GetSpeedConKI (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?speedconki	-

LSX_SetSpeedConKI			
Description:	Command for setting the speed controller integral part in %		
Delphi:	function LSX_SetSpeedConKI (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetSpeedConKI (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Speed controller integral part, range: 0 to 5000%	
Example:	LSX.SetSpeedConKI (75, 60, 90, 90);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!speedconki	ValidPar / ValidConfig

LSX_GetSpeedConKD			
Description:	Command for reading the speed controller differential part in %		
Delphi:	function LSX_GetSpeedConKD (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetSpeedConKD (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Speed controller differantial part, range: 0 to 5000%	
Example:	LSX.GetSpeedConKD (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?speedconkd	-

LSX_SetSpeedConKD			
Description:	Command for setting the speed controller differential part in %		
Delphi:	function LSX_SetSpeedConKD (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetSpeedConKD (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Speed controller differantial part, range: 0 to 5000%	
Example:	LSX.SetSpeedConKD (75, 60, 90, 90);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!speedconkd	ValidPar / ValidConfig

LSX_GetSpeedConOutPass			
Description:	Command for reading the speed controller output filter time constant.		
Delphi:	function LSX_GetSpeedConOutPass (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetSpeedConOutPass (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Speed controller output filter time constant, range: 0 bis 100000 µs	
Example:	LSX.GetSpeedConOutPass (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?speedconoutpass	-

LSX_SetSpeedConOutPass			
Description:	Command for setting the speed controller output filter time constant.		
Delphi:	function LSX_SetSpeedConOutPass (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetSpeedConOutPass (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Speed controller output filter time constant, range: 0 bis 100000 µs	
Example:	LSX.SetSpeedConOutPass (200, 500, 200, 500);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!speedconoutpass	ValidPar / ValidConfig

LSX_GetActSpeedFilterConst			
Description:	Command for reading the speed actual value filter time constant		
Delphi:	function LSX_GetActSpeedFilterConst (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetActSpeedFilterConst (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	speed actual value filter time constant, range: 0 bis 10000 µs	
Example:	LSX.GetActSpeedFilterConst (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?actspeedfilterconst	-

LSX_SetActSpeedFilterConst			
Description:	Command for setting the speed actual value filter time constant		
Delphi:	function LSX_SetActSpeedFilterConst (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetActSpeedFilterConst (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	speed actual value filter time constant, range: 0 bis 10000 µs	
Example:	LSX.SetActSpeedFilterConst (200, 500, 200, 500);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!actspeedfilterconst	ValidPar / ValidConfig

LSX_GetSpeedFeedForward			
Description:	Command for reading the speed or velocity feed forward in %		
Delphi:	function LSX_GetSpeedFeedForward (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetSpeedFeedForward (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Speed or velocity feed forward, range: 0 to 400%	
Example:	LSX.GetSpeedFeedForward (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?speedfeedforward	-

LSX_SetSpeedFeedForward			
Description:	Command for setting the speed or velocity feed forward in %		
Delphi:	function LSX_SetSpeedFeedForward (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetSpeedFeedForward (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Speed or velocity feed forward, range: 0 to 400%	
Example:	LSX.SetSpeedFeedForward (75, 60, 90, 90);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!speedfeedforward	ValidPar / ValidConfig

LSX_GetActAccelFilterConst			
Description:	Command for reading the acceleration actual value filter time constant		
Delphi:	function LSX_GetActAccelFilterConst (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetActAccelFilterConst (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	acceleration actual value filter time constant, range: 0 to 100 ms	
Example:	LSX.GetActAccelFilterConst (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?actaccelfilterconst	-

LSX_SetActAccelFilterConst			
Description:	Command for setting the acceleration actual value filter time constant		
Delphi:	function LSX_SetActAccelFilterConst (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetActAccelFilterConst (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	acceleration actual value filter time constant, range: 0 to 100 ms	
Example:	LSX.SetActAccelFilterConst (20, 5, 20, 5);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!actaccelfilterconst	ValidPar / ValidConfig

LSX_GetAccelFeedForward			
Description:	Command for reading the acceleration feed forward in %		
Delphi:	function LSX_GetAccelFeedForward (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetAccelFeedForward (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	acceleration feed forward, range: 0 to 400%	
Example:	LSX.GetAccelFeedForward (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?accelfeedforward	-

LSX_SetAccelFeedForward			
Description:	Command for setting the acceleration feed forward in %		
Delphi:	function LSX_SetAccelFeedForward (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetAccelFeedForward (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	acceleration feed forward, range: 0 to 400%	
Example:	LSX.SetAccelFeedForward (50, 80, 70, 70);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!accelfeedforward	ValidPar / ValidConfig

LSX_GetAccelFeedForwardOutPass			
Description:	Command for reading the acceleration feed forward output filter time constant		
Delphi:	function LSX_GetAccelFeedForwardOutPass (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetAccelFeedForwardOutPass (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	acceleration feed forward output filter time constant, range 0 to 100000 µs	
Example:	LSX.GetAccelFeedForwardOutPass (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?accelfeedforwardoutpass	-

LSX_SetAccelFeedForwardOutPass			
Description:	Command for setting the acceleration feed forward output filter time constant		
Delphi:	function LSX_SetAccelFeedForwardOutPass (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetAccelFeedForwardOutPass (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	acceleration feed forward output filter time constant, range 0 to 100000 µs	
Example:	LSX.SetAccelFeedForwardOutPass (200, 500, 200, 500);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!accelfeedforwardoutpass	ValidPar / ValidConfig

LSX_GetSpeedConTN			
Description:	Command for reading the effective reset time TN of the speed controller in [ms]		
Delphi:	function LSX_GetSpeedConTN (LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetSpeedConTN (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Effective reset time TN	
Example:	LSX.GetSpeedConTN (&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?speedcontn	-

4.2.18 Deviation Criterion

LSX_GetDevCritMode			
Description:	Reads if the deviation criterion is queried over each move or continouosly.		
Delphi:	LSX_GetDevCritMode (LSID: Integer; var X: integer; var Y: integer; var Z: integer; var R: integer): Integer;		
C++:	int GetDevCritMode (int *plX, int *plY, int *plZ, int *plR);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Assigned mode of the deviation criterion 0 = The deviation criterion is monitored from the start of a move until its end. A new move start resets the criterion to 0 1 = The deviation criterion is measured continouosly	
Example:	LSX.GetDevCritMode(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?devcritmode	-

LSX_SetDevCritMode			
Description:	Command for setting if the deviation criterion is queried over each move or continouosly		
Delphi:	function LSX_SetDevCritMode(LSID: Integer; X: integer; Y: integer; Z: integer; R: integer): Integer;		
C++:	SetDevCritMode (int lX, int lY, int lZ, int lR);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	Assignable modes of the deviation criterion 0 = The deviation criterion is monitored from the start of a move until its end. A new move start resets the criterion to 0 1 = The deviation criterion is measured continouosly	
Example:	LSX.SetDevCritMode (1, 0, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!devcritmode	LSX_SetDevCritEnable

LSX_GetDevCritType			
Description:	The command queries whether the deviation criterion is determined based on the absolute value of the deviation or on the square of the deviation.		
Delphi:	function LSX_GetDevCritType (LSID: Integer; var X: integer; var Y: integer; var Z: integer; var R: integer): Integer;		
C++:	int GetDevCritType (int *plX, int *plY, int *plZ, int *plR);		
LabView:	Not supported		
Parameter:	X, Y, Z, R	Assigned type of the deviation criterion 0 = The deviation criterion is measured based on the square of the deviation 1 = The deviation criterion is measured based on the sum of the deviation criterion	
Example:	LSX.GetDevCritType(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?devcrittype	-

LSX_SetDevCritType			
Description:	The command specifies whether the deviation criterion is determined based on the arithmetic mean of the deviation or on the root mean square of the deviation.		
Delphi:	function LSX_SetDevCritType (LSID: Integer; X: integer; Y: integer; Z:integer; R: integer): Integer;		
C++:	int SetDevCritType (int lX, int lY, int lZ, int lR);		
LabView:	Not supported		
Parameter:	X, Y, Z, R	Assignable types of the deviation criterion 0 = The deviation criterion is measured based on the arithmetic mean of the deviation 1 = The deviation criterion is measured based on the root mean square of the deviation	
Example:	LSX.SetDevCritType(1, 0, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!devcrittype	LSX_SetDevCritEnable

LSX_GetDevCritEnable			
Description:	The command queries whether the deviation criterion measurement is active.		
Delphi:	function LSX_GetDevCritEnable (LSID: Integer; var X: LongBool; var Y: LongBool; var Z: LongBool; var R: LongBool): Integer;		
C++:	int GetDevCritEnable (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbR);		
LabView:	Not supported		
Parameter:	X, Y, Z, R	True = The measurement of the deviation criterion is active False = The measurement of the deviation criterion is inactive	
Example:	LSX.GetDevCritType(&X, &Y, &Z, &R);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?devcritenable	-

LSX_SetDevCritEnable			
Description:	The command activates the measurement of the deviation criterion		
Delphi:	function LSX_SetDevCritEnable (LSID: Integer; X: LongBool; Y: LongBool; Z: LongBool; R: LongBool): Integer;		
C++:	int SetDevCritEnable (BOOL bX, BOOL bY, BOOL bZ, BOOL bR);		
LabView:	Not supported		
Parameter:	X, Y, Z, A	True = Activate the measurement of the deviation criterion False = Deactivate the measuremend of the deviation criteiron	
Example:	LSX.SetDevCritEnable(True, True, True, True);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!devcritenable	-

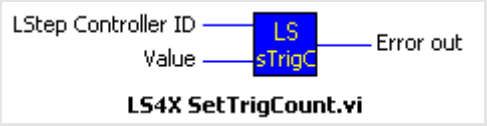
LSX_GetDevCritValue			
Description:	The command queries the deviation criterion measurement.		
Delphi:	function LSX_GetDevCritEnable (LSID: Integer; var X: LongBool; var Y: LongBool; var Z: LongBool; var R: LongBool): Integer;		
C++:	int GetDevCritEnable (BOOL *pbX, BOOL *pbY, BOOL *pbZ, BOOL *pbR);		
LabView:	Not supported		
Parameter:	X, Y, Z, R	Value of the deviation criterion	
Example:	LSX.GetDevCritValue (&X,&Y,&Z,&R);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?devcritvalue	-

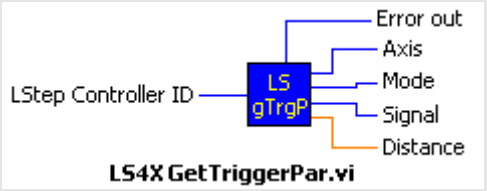
4.2.19 Trigger output

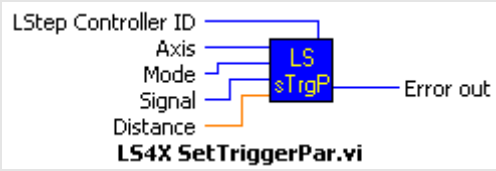
LSX_GetTrigger			
Description:	This function delivers the status of the trigger function.		
Delphi:	function LSX_GetTrigger(LSID: Integer; var ATrigger: Integer): Integer; function LSX_GetTrigger(LSID: Integer; var ATrigger: LongBool): Integer; (deprecated)		
C++:	int GetTrigger (int *plATrigger);		
LabView:			
Parameter:	ATrigger	Status of trigger function True = Trigger “On” False = Trigger “Off”	
		Status of triggerfunction 0 = Trigger function deactivated 1 = Trigger function computed with position values 2 = Trigger function computed with position and speed values	
Example:	LSX.GetTrigger(&ATrigger);		
Other:	Compatibility	“SendString” command	Activation (LSTEP express series)
	3	?trig	-

LSX_SetTrigger			
Description:	Function for activating/deactivating the trigger function.		
Delphi:	function LSX_SetTrigger(LSID: Integer; ATrigger: Integer): Integer; function LSX_SetTrigger(LSID: Integer; ATrigger: LongBool): Integer; (deprecated)		
C++:	int SetTrigger (int lATrigger);		
LabView:			
Parameter:	ATrigger	Status of trigger function True = Trigger "On" False = Trigger "Off"	
		Status of triggerfunction 0 = Trigger function deactivated 1 = Trigger function computed with position values 2 = Trigger function computed with position and speed values	
Example:	LSX.SetTrigger(1);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!trig	-

LSX_GetTrigCount			
Description:	Function for reading the trigger counter.		
Delphi:	function LSX_GetTriggerCount(LSID: Integer; var Value: Cardinal): Integer; function LSX_GetTrigCount(LSID: Integer; var Value: Integer): Integer; (deprecated)		
C++:	int GetTriggerCount (unsigned long *plValue); int GetTrigCount(int *pValue);		
LabView:			
Parameter:	Value	Number of executed trigger events	
Example:	LSX.GetTrigCount(&Value);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?trigcount	-

LSX_SetTrigCount			
Description:	Function for setting the trigger counter reading.		
Delphi:	function LSX_SetTriggerCount(LSID: Integer; Value: Cardinal): Integer; function LSX_SetTrigCount(LSID: Integer; Value: Integer): Integer; (deprecated)		
C++:	int SetTriggerCount (unsigned long lValue); int SetTrigCount(int Wert);		
LabView:			
Parameter:	Value	Intended value of trigger counter	
Example:	LSX.SetTrigCount(0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!trigcount	-

LSX_GetTriggerPar		
Description:	Delivers the set parameters of the Trigger function.	
Delphi:	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 allocation of trigger function X = 1 Y = 2 …
	Mode	Trigger mode (see controller manual, trigm command)
	Signal	Trigger signal length in µs
	Distance	Trigger distance in the set unit of the axis
Example:	LSX.GetTriggerPar(&Axis, & Mode, & Signal, & Distance);	
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?triga / ?trigm ?trigs / ?trigd	-

LSX_SetTriggerPar			
Description:	Sets the trigger parameters for the trigger function		
Delphi:	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 allocation of trigger function X = 1 Y = 2 …	
	Mode	Trigger mode (see controller manual, trigm command)	
	Signal	Trigger signal length in µs	
	Distance	Trigger distance in the set unit of the axis	
Example:	LSX.SetTriggerPar(1, 3, 2, 5.0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!triga / !trigm !trigs / !trigd	SetTrigger

LSX_GetTriggerDim			
Description:	Shows the selected unit for parametertransfer of triggerdistance and trig-gerhysteresis		
Delphi:	function LSX_GetTriggerDim(LSID: Integer;var Dimension: Integer): Integer;		
C++:	int GetTriggerDim (int *plDimension);		
LabView:	Not supported		
Parameter:	Dimension	0 = Parameter are transfered in userunit (see command "dim") 1 = Parameter are transfered regardless of userunit in microsteps	
Example:	LSX.GetTriggerDim (&Dimension);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?trigdim	-

LSX_SetTriggerDim			
Description:	Selects the unit for parametertransfer of triggerdistance and triggerhysteresis		
Delphi:	function LSX_SetTriggerDim(LSID: Integer; Dimension: Integer): Integer;		
C++:	int SetTriggerDim (int lDimension);		
LabView:	Not supported		
Parameter:	Dimension	0 = Parameter are transfered in userunit (see command "dim") 1 = Parameter are transfered regardless of userunit in microsteps	
Example:	LSX.SetTriggerDim(0);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!trigdim	-

LSX_GetTriggerSource			
Description:	Shows the trigger source. Triggering is possible to the target position or to the encoder value of the selected axis (see command "triga"). Triggering to the encoder value requires parameterization of a suitable trigger hysteresis (see command "trigh").		
Delphi:	function LSX_GetTriggerSource (LSID: Integer; var Source: integer): Integer;		
C++:	int GetTriggerSource (int *plSource);		
LabView:	Not supported		
Parameter:	Dimension	0 = Triggers to target position 1 = Triggers to encoder values	
Example:	LSX.GetTriggerSource (&Source);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?trigsource	-

LSX_SetTriggerSource			
Description:	Selects the trigger source. Triggering is possible to the target position or to the encoder value of the selected axis (see command "triga"). Triggering to the encoder value requires parameterization of a suitable trigger hysteresis (see command "trigh").		
Delphi:	function LSX_SetTriggerSource (LSID: Integer; Source: Integer): Integer;		
C++:	int SetTriggerSource (int lSource);		
LabView:	Not supported		
Parameter:	Dimension	0 = Triggers to target position 1 = Triggers to encoder values	
Example:	LSX.SetTriggerSource(0);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!trigsource	SetTrigger

LSX_GetTriggerHysterese			
Description:	Shows the trigger hysteresis. Triggering to encoder values requires a hysteresis by the tripper position. This prevents the unintended output of trigger signals due to errors (e.g. deviation of axis by external forces / moments through trigger position or noise on the encoder signal, ...). After a trigger pulse output, another pulse can only be given after covering the trigger hysteresis; for this reason, the trigger hysteresis has to be shorter than 50% of the trigger distance.		
Delphi:	function LSX_GetTriggerHysterese (LSID: Integer; var Hysterese: Double): Integer;		
C++:	int GetTriggerHysterese (double *pdHysterese);		
LabView:	Not supported		
Parameter:	Hysterese	0 to a maximum with up to 8 decimal places in the set dimension	
Example:	LSX.GetTriggerHysterese (&Hysterese);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?trigh	-

LSX_SetTriggerHysterese			
Description:	Sets a trigger hysteresis. Triggering to encoder values requires a hysteresis by the tripper position. This prevents the unintended output of trigger signals due to errors (e.g. deviation of axis by external forces / moments through trigger position or noise on the encoder signal, ...). After a trigger pulse output, another pulse can only be given after covering the trigger hysteresis; for this reason, the trigger hysteresis has to be shorter than 50% of the trigger distance.		
Delphi:	function LSX_SetTriggerHysterese (LSID: Integer; Hysterese: Double): Integer;		
C++:	int SetTriggerHysterese (double dHysterese);		
LabView:	Not supported		
Parameter:	Hysterese	0 to a maximum with up to 8 decimal places in the set dimension	
Example:	LSX.SetTriggerHysterese(0.0022);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!trigh	SetTrigger

LSX_GetTriggerPLength			
Description:	Shows the trigger signal period length after which the next trigger is output (burst mode)		
Delphi:	function LSX_GetTriggerPLength (LSID: Integer; var Length: Integer): Integer;		
C++:	int GetTriggerPLength (int *plLength);		
LabView:	Not supported		
Parameter:	Length	Trigger signal period length	
Example:	LSX.GetTriggerPLength (&Length);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?trigp	-

LSX_SetTriggerPLength			
Description:	Sets the trigger signal period length after which the next trigger is output (burst mode)		
Delphi:	function LSX_SetTriggerPLength (LSID: Integer; Length: Integer): Integer;		
C++:	int SetTriggerPLength (int lLength);		
LabView:	Not supported		
Parameter:	Length	Trigger signal period length	
Example:	LSX.SetTriggerPLength(0.0022);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!trigp	SetTrigger

LSX_GetTriggerPulsCount			
Description:	Shows the number of pulses to be output in burst mode		
Delphi:	function LSX_GetTriggerPulsCount (LSID: Integer;var Count : Integer): integer;		
C++:	int GetTriggerPulsCount (int *plCount);		
LabView:	Not supported		
Parameter:	Count	Trigger puls number	
Example:	LSX.GetTriggerPulsCount (&Count);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?trigc	-

LSX_SetTriggerPulsCount			
Description:	Sets the number of pulses to be output in burst mode		
Delphi:	function LSX_SetTriggerPulsCount (LSID: Integer; Count: Integer):Integer;		
C++:	int SetTriggerPulsCount (int lCount);		
LabView:	Not supported		
Parameter:	Count	Trigger puls number	
Example:	LSX.SetTriggerPulsCount(0.0022);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!trigc	SetTrigger

LSX_GetTrigger_Two			
Description:	This function delivers the status of the second trigger function.		
Delphi:	function LSX_GetTrigger_Two(LSID: Integer; var Status: Integer): Integer;		
C++:	int GetTrigger_Two (int *plStatus);		
LabView:			
Parameter:	Status	Status of triggerfunction 0 = Trigger function deactivated 1 = Trigger function computed with position values 2 = Trigger function computed with position and speed values	
Example:	LSX.GetTrigger(&Status);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?trig_two	-

LSX_SetTrigger_Two			
Description:	Function for activating/deactivating the second trigger function.		
Delphi:	function LSX_SetTrigger_Two(LSID: Integer; Status: Integer): Integer;		
C++:	int SetTrigger_Two (int lStatus);		
LabView:			
Parameter:	Status	Status of triggerfunction 0 = Trigger function deactivated 1 = Trigger function computed with position values 2 = Trigger function computed with position and speed values	
Example:	LSX.SetTrigger_Two(1);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!trig_two	-

LSX_GetTrigger_TwoCount			
Description:	Function for reading the second trigger counter.		
Delphi:	function LSX_GetTrigger_TwoCount(LSID: Integer; var Value: Cardinal): Integer;		
C++:	int GetTrigger_TwoCount (unsigned long *plValue);		
LabView:	Not supported		
Parameter:	Value	Number of executed trigger events	
Example:	LSX.GetTrigger_TwoCount(&Value);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	?trigcount_two	-

LSX_SetTrigger_TwoCount			
Description:	Function for setting the second trigger counter reading.		
Delphi:	function LSX_SetTrigger_TwoCount(LSID: Integer; Value: Cardinal): Integer;		
C++:	int SetTrigger_TwoCount (unsigned long lValue);		
LabView:	Not supported		
Parameter:	Value	Intended value of trigger counter	
Example:	LSX.SetTrig_TwoCount(0);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	2	!trigcount_two	-

<table>
<tr><th colspan="4">LSX_GetTrigger_TwoPar</th></tr>
<tr><td>Description:</td><td colspan="3">Delivers the set parameters of the second trigger function.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LSX_GetTrigger_TwoPar(LSID: Integer; var Axis, Mode, Signal: Integer; var Distance: Double): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int GetTrigger_TwoPar(int *plAxis, int *plMode, int *plSignal, double *pdDistance);</td></tr>
<tr><td>LabView:</td><td colspan="3">Not supported</td></tr>
<tr><td rowspan="4">Parameter:</td><td>Axis</td><td colspan="2">Axis allocation of trigger function
X = 1
Y = 2
…</td></tr>
<tr><td>Mode</td><td colspan="2">Trigger mode (see controller manual, trigm command)</td></tr>
<tr><td>Signal</td><td colspan="2">Trigger signal length in µs</td></tr>
<tr><td>Distance</td><td colspan="2">Trigger distance in the set unit of the axis</td></tr>
<tr><td>Example:</td><td colspan="3">LSX.GetTrigger_TwoPar(&Axis, & Mode, & Signal, & Distance);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEP express series)</td></tr>
<tr><td>2</td><td>?triga_two / ?trigm_two
?trigs_two / ?trigd_two</td><td>-</td></tr>
</table>

LSX_SetTrigger_TwoPar				
Description:	Sets the trigger parameters for the second trigger function			
Delphi:	function LSX_SetTrigger_TwoPar(LSID: Integer; Axis, Mode, Signal: Integer; Distance: Double): Integer;			
C++:	int SetTrigger_TwoPar(int lAxis, int lMode, int lSignal, double dDistance);			
LabView:	Not supported			
Parameter:	Axis	Axis allocation of trigger function X = 1 Y = 2 …		
	Mode	Trigger mode (see controller manual, trigm command)		
	Signal	Trigger signal length in µs		
	Distance	Trigger distance in the set unit of the axis		
Example:	LSX.SetTrigger_TwoPar(1, 3, 2, 5.0);			
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)	
	3	!triga_two / !trigm_two !trigs_two / !trigd_two	SetTrigger	

LSX_GetTrigger_TwoDim			
Description:	Shows the selected unit for parametertransfer of triggerdistance and triggerhysteresis of the second trigger		
Delphi:	function LSX_GetTrigger_TwoDim(LSID: Integer;var Dimension: Integer): Integer;		
C++:	int GetTrigger_TwoDim (int *plDimension);		
LabView:	Not supported		
Parameter:	Dimension	0 = Parameter are transfered in userunit (see command "dim") 1 = Parameter are transfered regardless of userunit in microsteps	
Example:	LSX.GetTrigger_TwoDim (&Dimension);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?trigdim_two	-

LSX_SetTrigger_TwoDim			
Description:	Selects the unit for parametertransfer of triggerdistance and triggerhysteresis of the second trigger		
Delphi:	function LSX_SetTrigger_TwoDim(LSID: Integer; Dimension: Integer): Integer;		
C++:	int SetTrigger_TwoDim (int lDimension);		
LabView:	Not supported		
Parameter:	Dimension	0 = Parameter are transfered in userunit (see command "dim") 1 = Parameter are transfered regardless of userunit in microsteps	
Example:	LSX.SetTrigger_TwoDim(0);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!trigdim_two	-

LSX_GetTrigger_TwoSource			
Description:	Shows the trigger source of the second trigger. Triggering is possible to the target position or to the encoder value of the selected axis (see command "triga_two"). Triggering to the encoder value requires parameterization of a suitable trigger hysteresis (see command "trigh_two").		
Delphi:	function LSX_GetTrigger_TwoSource (LSID: Integer; var Source: integer): Integer;		
C++:	int GetTrigger_TwoSource (int *plSource);		
LabView:	Not supported		
Parameter:	Dimension	0 = Triggers to target position 1 = Triggers to encoder values	
Example:	LSX.GetTrigger_TwoSource (&Source);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?trigsource_two	-

LSX_SetTrigger_TwoSource			
Description:	Selects the trigger source of the second trigger. Triggering is possible to the target position or to the encoder value of the selected axis (see command "triga_two"). Triggering to the encoder value requires parameterization of a suitable trigger hysteresis (see command "trigh_two").		
Delphi:	function LSX_SetTrigger_TwoSource (LSID: Integer; Source: Integer): Integer;		
C++:	int SetTrigger_TwoSource (int lSource);		
LabView:	Not supported		
Parameter:	Dimension	0 = Triggers to target position 1 = Triggers to encoder values	
Example:	LSX.SetTrigger_TwoSource(0);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!trigsource_two	SetTrigger_Two

LSX_GetTrigger_TwoHysterese			
Description:	Shows the trigger hysteresis of the second trigger. Triggering to encoder values requires a hysteresis by the tripper position. This prevents the unintended output of trigger signals due to errors (e.g. deviation of axis by external forces / moments through trigger position or noise on the encoder signal, ...). After a trigger pulse output, another pulse can only be given after covering the trigger hysteresis; for this reason, the trigger hysteresis has to be shorter than 50% of the trigger distance.		
Delphi:	function LSX_GetTrigger_TwoHysterese (LSID: Integer; var Hysterese: Double): Integer;		
C++:	int GetTrigger_TwoHysterese (double *pdHysterese);		
LabView:	Not supported		
Parameter:	Hysterese	0 to a maximum with up to 8 decimal places in the set dimension	
Example:	LSX.GetTrigger_TwoHysterese (&Hysterese);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?trigh_two	-

LSX_SetTrigger_TwoHysterese			
Description:	Sets a trigger hysteresis for the second trigger. Triggering to encoder values requires a hysteresis by the tripper position. This prevents the unintended output of trigger signals due to errors (e.g. deviation of axis by external forces / moments through trigger position or noise on the encoder signal, ...). After a trigger pulse output, another pulse can only be given after covering the trigger hysteresis; for this reason, the trigger hysteresis has to be shorter than 50% of the trigger distance.		
Delphi:	function LSX_SetTrigger_TwoHysterese (LSID: Integer; Hysterese: Double): Integer;		
C++:	int SetTrigger_TwoHysterese (double dHysterese);		
LabView:	Not supported		
Parameter:	Hysterese	0 to a maximum with up to 8 decimal places in the set dimension	
Example:	LSX.SetTrigger_TwoHysterese(0.0022);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!trigh_two	SetTrigger_Two

LSX_GetTrigger_TwoPLength			
Description:	Shows the trigger signal period length after which the next trigger is output (burst mode) of the second trigger.		
Delphi:	function LSX_GetTrigger_TwoPLength (LSID: Integer; var Length: Integer): Integer;		
C++:	int GetTrigger_TwoPLength (int *plLength);		
LabView:	Not supported		
Parameter:	Length	Trigger signal period length	
Example:	LSX.GetTrigger_TwoPLength (&Length);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?trigp_two	-

LSX_SetTrigger_TwoPLength			
Description:	Sets the trigger signal period length after which the next trigger is output (burst mode) of the second trigger.		
Delphi:	function LSX_SetTrigger_TwoPLength (LSID: Integer; Length: Integer): Integer;		
C++:	int SetTrigger_TwoPLength (int lLength);		
LabView:	Not supported		
Parameter:	Length	Trigger signal period length	
Example:	LSX.SetTrigger_TwoPLength(0.0022);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!trigp_two	SetTrigger_Two

LSX_GetTrigger_TwoPulsCount			
Description:	Shows the number of pulses to be output in burst mode of the second trigger.		
Delphi:	function LSX_GetTrigger_TwoPulsCount (LSID: Integer;var Count : Integer): integer;		
C++:	int GetTrigger_TwoPulsCount (int *plCount);		
LabView:	Not supported		
Parameter:	Count	Trigger puls number	
Example:	LSX.GetTrigger_TwoPulsCount (&Count);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?trigc_two	-

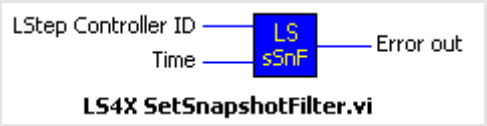
LSX_SetTrigger_TwoPulsCount			
Description:	Sets the number of pulses to be output in burst mode of the second trigger.		
Delphi:	function LSX_SetTrigger_TwoPulsCount (LSID: Integer; Count: Integer):Integer;		
C++:	int SetTrigger_TwoPulsCount (int lCount);		
LabView:	Not supported		
Parameter:	Count	Trigger puls number	
Example:	LSX.SetTrigger_twoPulsCount(0.0022);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!trigc_two	SetTrigger_Two

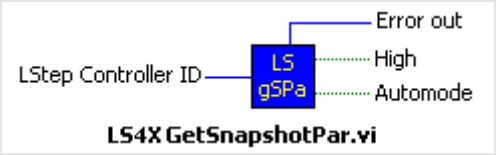
4.2.20 Snapshot input

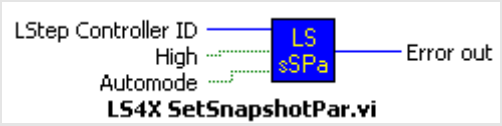
LSX_GetSnapshot			
Description:	Shows the current snapshot function status.		
Delphi:	function LSX_GetSnapshot(LSID: Integer; var ASnapshot: LongBool): Integer;		
C++:	int GetSnapshot(BOOL *pbASnapshot);		
LabView:			
Parameter:	ASnapshot	Status of snapshot function True = Snapshot function “On” False = Snapshot function “Off”	
Example:	LSX.GetSnapshot(&ASnapshot);		
Other:	Compatibility	“SendString” command	Activation (LSTEP express series)
	3	?sns	-

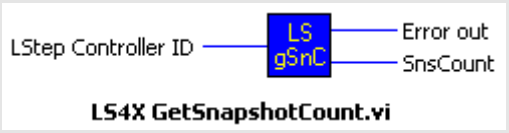
LSX_SetSnapshot			
Description:	Function for activating/deactivating the snapshot function.		
Delphi:	function LSX_SetSnapshot(LSID: Integer; ASnapshot: LongBool): Integer;		
C++:	int SetSnapshot (BOOL bASnapshot);		
LabView:			
Parameter:	ASnapshot	Status of snapshot function True = Snapshot function “On” False = Snapshot function “Off”	
Example:	LSX.SetSnapshot(true);		
Other:	Compatibility	“SendString” command	Activation (LSTEP express series)
	3	!sns	-

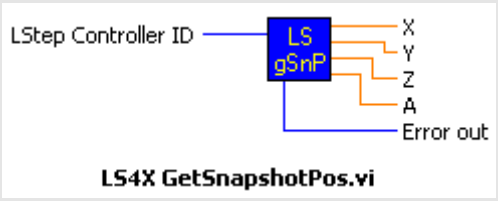
LSX_GetSnapshotFilter			
Description:	Reading of set input filter of snapshot function.		
Delphi:	function LSX_GetSnapshotFilter(LSID: Integer; var lTime: Integer): Integer;		
C++:	int GetSnapshotFilter(int *plTime);		
LabView:			
Parameter:	lTime	Filter time in ms	
Example:	LSX.GetSnapshotFilter(&lTime);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?snsf	-

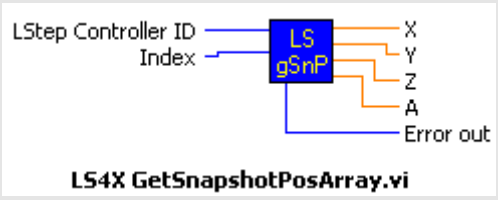
LSX_SetSnapshotFilter			
Description:	Function for setting the input filter of the snapshot function for bouncing switches.		
Delphi:	function LSX_SetSnapshotFilter(LSID: Integer; lTime: Integer): Integer;		
C++:	int SetSnapshotFilter(int lTime);		
LabView:			
Parameter:	lTime	Filter time in ms	
Example:	LSX.SetSnapshotFilter(0); // no Snapshot filter		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!snsf	SetSnapshot

<table>
<tr><th colspan="5">LSX_GetSnapshotPar</th></tr>
<tr><td>Description:</td><td colspan="4">Function for reading the snapshot parameters.</td></tr>
<tr><td>Delphi:</td><td colspan="4">function LSX_GetSnapshotPar(LSID: Integer; var High, AutoMode: LongBool): Integer;</td></tr>
<tr><td>C++:</td><td colspan="4">int GetSnapshotPar(BOOL *pbHigh, BOOL *pbAutoMode);</td></tr>
<tr><td>LabView:</td><td colspan="4"> 
</td></tr>
<tr><td rowspan="2">Parameter:</td><td>High</td><td colspan="3">Activation of the snapshot input
True = Snapshot is high-active
False = Snapshot is low-active</td></tr>
<tr><td>AutoMode</td><td colspan="3">Activation of automatic mode
True = Snapshot function in “Automatic mode”. The position is automatically approached after the first impulse.
False = Snapshot function is not in automatic mode</td></tr>
<tr><td>Example:</td><td colspan="4">LSX.GetSnapshotPar(&High, & AutoMode);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEP express series)</td></tr>
<tr><td>3</td><td>?snsl / !snsm</td><td colspan="2">-</td></tr>
</table>

LSX_SetSnapshotPar			
Description:	Function for setting the snapshot parameters.		
Delphi:	function LSX_SetSnapshotPar(LSID: Integer; High, AutoMode: LongBool): Integer;		
C++:	int SetSnapshotPar(BOOL bHigh, BOOL bAutoMode);		
LabView:			
Parameter:	High	Activation of the snapshot input True = Snapshot is high-active False = Snapshot is low-active	
	AutoMode	Activation of automatic mode True = Snapshot function in "Automatic mode". The position is automatically approached after the first impulse. False = Snapshot function is not in automatic mode	
Example:	LSX.SetSnapshotPar(true, false);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!snsl / !snsm	SetSnapShot

LSX_GetSnapshotCount			
Description:	Function for reading the snapshot counter.		
Delphi:	function LSX_GetSnapshotCount(LSID: Integer; var SnsCount: Integer): Integer;		
C++:	int GetSnapshotCount(int *plSnsCount);		
LabView:			
Parameter:	SnsCount:	Snapshot counter reading	
Example:	LSX.GetsnapshotCount(&SnsCount);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	?snsc	-

LSX_GetSnapshotPos			
Description:	Function for reading the snapshot position.		
Delphi:	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	Snapshot position in the set unit	
Example:	double X, Y, Z, A; LSX.GetSnapshotPos(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!snsp	-

LSX_GetSnapshotPosArray			
Description:	Function for reading the arrays of the snapshot positions.		
Delphi:	function LSX_GetSnapshotPosArray(LSID: Integer; Index: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetSnapshotPosArray(int lIndex, double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	Index	Index in snapshop position array	
	X, Y, Z, A	Position values in the set axis dimension.	
Example:	double X, Y, Z, A; LSX.GetSnapshotPos(2, &X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEP express series)
	3	!snsa	-

LSX_GetSnapshotSource			
Description:	Shows the position value source: The target position or the encoder values of the selected axes may be saved.		
Delphi:	function LSX_GetSnapShotSource (LSID: Integer;var X,Y,Z,A: Integer): Integer;		
C++:	int GetSnapShotSource (int *plX, int *plY, int *plZ, int *plA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	0 → Saves target positions 1 → Saves encoder values	
Example:	LSX.GetSnapshotSource(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	?snssource	-

LSX_SetSnapshotSource			
Description:	Selects the position value source: The target position or the encoder values of the selected axes may be saved.		
Delphi:	function LSX_SetSnapShotSource (LSID: Integer; X,Y,Z,A: Integer): Integer; function LSX_SetSnapShotSource (LSID: Integer;Axis: Integer;Source:Integer): Integer;		
C++:	int SetSnapShotSource (int lX, int lY, int lZ, int lA);		
LabView:	Not supported		
Parameter:	X,Y,Z,A	0 → Sollposition speichern 1 → Geberwert speichern	
	Axis,Source	Axis: 1 → X, 2 → Y, 3 → Z, 4 → A Source: 0 → Sollposition speichern 1 → Geberwert speichern	
Example:	LSX.SetSnapshotSource(1,1); LSX.SetSnapshotSource(1,1,0,0);		
Other:	Compatibility	"SendString" Command	Activation (LSTEP express series)
	2	!snssource	SetSnapShot

5 CallBack functions of the LSTEP-API

The LSTEP-API provides CallBack functions for asynchronous events. These are triggered by an LSTEP controller. The CallBack functions allow for processing an asynchronous event and signalling an appropriate action.

Since the CallBack function is activated during the processing of an incoming message, it has an influence on the processing speeds of messages. For this reason, the runtime should be observed with regard to the processing of asynchronous events. If extended processing is intended, this should not take place in the CallBack function, but be initiated by a signal.

The LSTEP-API offers a controller-related and a global CallBack function. If the controller-related function is used, the global CallBack function for this controller is deactivated.

The CallBack messages as well as CallBack applications are described below.

5.1 Standard CallBack-Function (OsziCallBackFct)

In the programming languages Delphi, C++ or C#, the standard CallBack function is declared as follows:

Declaration Standard-CallBack Function (OsziCallBackFct)		
Delphi	procedure OsziCallBackFct(Data: PAnsiChar; MaxLen: Integer; ChannelID: Integer); stdcall;	
C++	void CALLBACK OsziCallBackFct(char *pcData, int lMaxLen, int lChannelID)	
C#	void OsziCallBackFct (IntPtr Data, int MaxLen, int ChannelID)	
Parameter	Data	Pointer to ASCII string, zero-terminated.
	MaxLen	Maximum length of ASCII string in Data.
	ChannelID	Channel to which the ASCII string refers. For available channels, please refer to “5.3 Channel numbers”.
Return value	-	

This CallBack function is transferred to the API via the API function LSX_SetOsziCallBackFct. This is a global function, which does not allow any allocation to a controller or conclusions on the controller triggering the CallBack.

5.2 Extended CallBack function (ExtCallBackFct)

In the programming languages Delphi, C++ or C#, the extended CallBack function is declared as follows:

Declaration Extended-CallBack Function (ExtCallBackFct)		
Delphi	function ExtCallBackFct(Data: PAnsiChar; MaxLen: Integer; ChannelID: Integer; pObject: Pointer): Integer; stdcall;	
C++	int CALLBACK ExtCallBackFct(char *pcData, int lMaxLen, int lChannelID, void* pObject);	
C#	int ExtCallBackFunction(IntPtr Data, int MaxLen, int ChannelID, IntPtr pObject)	
Parameter	Data	Pointer to ASCII string, zero-terminated.
	MaxLen	Maximum length of ASCII string in Data.
	ChannelID	Channel to which the ASCII string refers. For available channels, please refer to “5.3 Channel numbers".
	pObject	Pointer to object which is transferred when the CallBack function is called.
Return value	The return value of the integer type is used to influence the communication input buffer of the API. 0x00000 = Do not perform any action 0x10000 = Delete current string of input buffer	

This CallBack function is transferred to the API via the API function LSX_SetExtCallBackFct. It is related to a controller or an object of the LSTEP-API.

The object behind the pointer pObject allows for entering conclusions on the activated controller.The object of the pointer pObject is transferred by the function LSX_SetExtCallBackFct.

By allocating the Extended CallBack function to a controller, this controller will not trigger any Standard CallBack function.

5.3 Channel numbers

The trigger of an API CallBack function is distinguished by the channel. The following channels have been defined:

1 = Oscilloscope channel

2 = Error/information channel

3 = Oscilloscope/information channel

4 = Digital input channel

…

10000 = Movement channel

5.3.1 Oscilloscope channel (1) and oscilloscope information channel (3).

These channels are used to implement a digital oscilloscope. More information is provided upon request. A model implementation may be viewed in WIN-Commander 5 Oscilloscope.

5.3.2 Error/information channel (2)

The data package of the error/information channel is a string divided in columns by tabs. The first column includes a sub-channel presented as an integer which may comprise a value range of 32 bit.A distinction is made between the sub-channels error channel (sub-channel 0) and information channel (sub-channel 99).

Error channel (sub-channel 0)

If the sub-channel is 0, the asynchronous data package includes an error message. The data package of the CallBack function for sub-channel 0 is composed as follows:

- Column 0 = Sub-channel
- Column 1 = Axis number (0 = X-axis, … 3 = A-axis, 5 = General message)
- Column 2 = Error number
- Column 3 = Operation time in seconds
- Column 4 = Millisecond portion of operation time

The error number may be looked up in the LSTEP controller documentation. In addition, the LSTEP-API offers the function LSX_TranslateErrMsg in order to make a translation. The source for the translations are language files enclosed to the LSTEP-API (e.g. LStep4deu.txt).

Information channel (sub-channel 99)

In this sub-channel, the column 0 with sub-channel number 99 is followed by an ASCII string with additional information. These include information, e.g. on which command has triggered an error. In WIN-Commander 5, this information is shown as a note in the error window.

5.3.3 Digital input channel (4)

This channel signals the status of the digital inputs of a controller. The signalization is triggered by a change in the digital inputs so that each change triggers a transmission through this channel.

For distinguishing the digital inputs from various plugs or input types of a controller, an identification is provided before the input status. The allocation of the identification can be taken from the controller documentation. It is separated from the input status by a tab.

The activation of this channel may be looked up in the controller documentation.

Control	For activation please refer to
LSTEP express Series	diginstatus
LSTEP 2000 Series	-
Others	-

5.3.4 Movement channel (10000)

This channel triggers a CallBack, if an axis mask is transmitted by a controller (see LSX_GetStatusAxis, LSX_GetStatusAxisW). The end of an asynchronous move may consequently be awaited in combination with the Autostatus functionality of the LSTEP controller without the need for a continuous inquiry of the axis mask.

In order that the synchronicity of the API communication input buffer is maintained, the last entry of the communication buffer has to be deleted in case of an asynchronous move with an active Autostatus. This entry corresponds to the axis mask. For deleting, the value 0x10000 is to be used as the return value of the Extended CallBack function.

5.4 Supported controller or interface types

The tables below show an overview of the controller systems and interfaces supporting the relevant channels.

Control	Supported channel					Activation
	1	2	3	4	10000	
LSTEP express Series	Yes	Yes	Yes	Yes	Yes	!errorchannel 2
LSTEP 2000 series	-	-	-	Yes[1)]	-	-
Others	-	-	-	-	-	-

1) Only for LStep PCI

Interface type		Supported channel					
No.	Type	1	2	3	4	…	10000
1	RS232	-	-	-	-	-	-
2	ArcNet	-	-	-	-	-	-
3	DPRAM / ISA-Bus	-	-	-	-	-	-
4	DPRAM / PCI-Bus	-	-	-	Yes	-	-
5	RS232 with RTS/CTS and extended log	Yes	Yes	Yes	Yes	-	Yes
11	RS232 with RTS/CTS	-	-	-	-	-	-

5.5 Application examples

The following project examples are intended to support the implementation of the CallBack functions:

C#_LogFile - program example for using a log file or the error channel
C++_MFC_LogFile - program example for using a log file or the error channel

The project examples are enclosed to the LSTEP-API archive.

6 Error codes

Two methods are available for error identification when using a Lang controller. One of them is the inquiry of the error number from the controller by means of the function LSX_GetError, the other one is the evaluation of the return value of the API functions.

6.1 Error numbers inquired from the controller (GetError)

The last error occurred is read from the LSX_GetErrorcontroller by means of the API function. It is indicated as a 32-bit integer and has a value below 4000. If the value is Zero, no error has occurred since the last inquiry. Apart from the functionLSX_GetError, these error numbers are also transmitted from the controller to the application through the Error/information channel (2) (see section 5.3.2). The list below shows the allocation of the error numbers: This allocation is enclosed to the LSTEP-API as text file in different languages (e.g. LStep4deu.txt):

0 = No error
1 = Valid axis designation missing
2 = Non-executable function
3 = Command string has too many characters
4 = Invalid command
5 = Not within valid numerical range
6 = Incorrect number of parameters
7 = Command must start with ! 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 = Move command not possible, as joystick is in use
12 = Limit switch tripped
13 = Function cannot be carried out because Encoder was recognized
14 = Error during calibration! Limit switch was not left correctly
15 = Error during calibration on reference mark
16 = Save command has failed
17 = Axis still in use
18 =Axis not ready
19 = Axis not calibrated
20 =Driver relay defective (safety circle K3/K4)
21 =Only single vectors may be driven (setup mode)
22 =No calibration, measuring table stroke or joystick operation (door open or setup mode)
23 = SECURITY Error X-axis
24 = SECURITY Error Y-axis
25 = SECURITY Error Z-axis
26 = SECURITY Error A-axis
27 = Emergency STOP
28 = Error in the door switch safety circle
29 = Power amplifiers are not activated

30 = GAL security error
31 = Joy-stick cannot be activated because Move is active
32 = Vector outside the travel range

1009 = Axes cannot be switched on or off because TVR is active
1010 = Other manual mode is active
1011 = Servo and step motor cannot be coupled (joystick)
1012 = Output already allocated to other function (digital output)
1013 = Hadware layer has been assigned multiple times
1014 = Axis mapping is not possible
1015 = trigger output frequency exceeded

1030 = Configuration is active
1031 = Axis not configured
1032 = Internal error
1033 = Axis still in use
1034 = Axis in error state
1035 = Axis not calibrated
1036 = Axis without RoomMeasure
1037 = Min. limit unknown
1038 = Max. limit unknown
1039 = Emergency-stop tripped
1040 = Limit switch reached
1041 = Travel distance too small
1042 = Speed too low
1043 = Jerk too small
1044 = No Trigger limit switch in
1045 = No Trigger limit switch out
1046 = Travel clipped
1047 = Limit switch override

1052 = Velocity too high
1053 = Acceleration too high
1054 = Jerk too high

1064 = Travel distance too long
1065 = Brake and power supply for limit switch not possible at the same time
1066 = No commutation necessary
1067 = Axis not commuted
1068 = Adaptive position controller not in servo operation permitted
1069 = Filter time constant is too small [1)]
1070 = Frequencies of band gap are not possible
1071 = time constant position control output filter too low

1072 = time constant speed control output filter too low
1073 = time constant acceleration feed forward filter too low
1074 = time constant speed actual value filter too low
1075 = time constant acceleration actual value filter too low
1076 = time constant velocity monitoring filter too low
1077 = time constant manual operation filter too low

1096 = Min. limit switch active
1097 = Max. limit 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 = Power amplifier overcurrent
1103 = Overcurrent upon activation
1104 = Overvoltage
1105 = Intermediate voltage fuse defect
1106 = Encoder error: amplitude too small
1107 = Encoder error: Amplitude too large
1108 = Position error too large
1109 = Speed too high
1110 = Motor blocked
1111 = Motor brake failure
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
1117 = Axis is moved
1118 = Switch for min. moving range activated
1119 = Switch for max. moving range activated
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 = Encoder track error
1125=Amplitude of encoder too small, maybe no transmitter connected
1126 = Angle in auto commutation outside tolerance; axis may be jammed
1127 = No rotational axis
1128=No Zero limit switch / no encoder reference mark
1129 = No encoder interface
1130 = Encoder input has more than one allocations
1131 = eQep encoder inputs not configured (hardware configuration MFP)

1132 = Target window not reached within permissible time
1133 = Encoder input not available
1134 = Auto commutation system larger than rated current
1135 = Auto commutation system is zero

1139 = Lag error not activated
1140 = Engine I^2T-current is smaller or equal to engine Ampacity

1160 = Missing OTP data

1162 = Computation time window exceeded (level 4)
1163 = Computation time window exceeded (level 3)
1164 = Computation time window exceeded (level 2)
1165 = Computation time window exceeded (level 1)
1166 = Device in “System Error” status
1167 = I^2T device error

1171 = Amplitude error commutation encoder (Input ENC1 to ENC8)
1172 = Frequencie error position encoder 1 phase error (Input ENC1 to ENC8)

1174 = Amplitude error of position encoder 1 (Input ENC1 to ENC8)
1175 = Frequencie error of position encoder 1 (Input ENC1 to ENC8)

1177 = Amplitude error of position encoder 2 (Input ENC1 to ENC8)
1178 = Frequencie error of position encoder 2 (Input ENC1 to ENC8)

1193 = Warning: Power amplifier overtemperature
1194 = Warning: Motor temperature too high
1195 = Driver fallen short of
1196 = Axis deactivated
1197 = Intermediate voltage too low
1198 = Intermediate voltage too high

1250 = Oscilloscope pretrigger position exceeds oscilloscope data size

The commands for error message 1069 are listed below. The minimum filter times are listed in the command descriptions:

- LSX_SetAccelFeedForwardOutPass / accelfeedforwardoutpass
- LSX_SetPosConOutPass / posconoutpass
- LSX_SetSpeedConOutPass / speedconoutpass
- LSX_SetActSpeedFilterConst / actspeedfilterconst
- LSX_SetActAccelFilterConst / actaccelfilterconst
- LSX_SetMonitoringVelFilter / monitoringvelfilter
- LSX_SetJoyOutPass /joyoutpass
- LSX_SetTippOutPass / tippoutpass

- LSX_SetTrackBallOutPass / tboutpass

6.2 Return value of LSTEP-API functions

All commands of the LSTEP-API deliver a 32-bit integer return value. If this value is 0 or 4100, the API command was executed without any error. If an error has occurred, an error number > 4000 is generated in the API and may be processed through the return value. The list below shows the generated error numbers:

4001 = Internal error
4002 = Internal error
4003 = Undefined error
4004 = Interface type unknown (may occur with Connect…)
4005 = Interface initialization error
4006 = No connection to 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 LSTEP
4011 = Joystick active (may occur with SetJoystickOn/Off)
4012 = Move command not possible, as joystick is active
4013 = Controller timeout with move command
4014 = Error during calibration, limit switch not left correctly
4015 = Limit switch activated in moving direction
4016 = Repeated vector start!! (control)
4031 = Joy-stick cannot be activated because Move is active!
4032 = Software limits undefined
4100 = No error

Error number as from 4100

These error numbers are generated by the LSTEP-API, if an error has occurred during the execution of an API command and this is signalled by the controller. For instance, if an "F" occurs in the axis mask or the status string includes "ERR ErrorNo" (e.g. "ERR 27").

In order to allocate a descriptive error text to these errors, the value 4100 has to be deducted from the error number, and the resulting number has to be looked up in " 6.1 Error numbers inquired from the controller (GetError)".

7 Frequent questions & answers

Overview
How is the LSTEP4X.DLL embedded in a MS Visual C++ project?
How do I initialise the connection to LSTEP with the LSTEP-API?
Which of the Connect commands should be used?

How do I initialise the driver for theLSTEP-PCI?
Why is my program with the LSTEP4X.DLL not able to establish a connection to the LSTEP-PCI?
A fault has occurred during the process, in theLSTEP4X.DLL or in my program. What is the cause for this problem, and how can I solve it?
Can I inquire the status of inputs, the current position, etc. during moving commands?
Why are messages processed during the execution of LSTEP-API functions and how can I deactivate this?
When should Moves be used with or without Wait?
How can I move single axes of the LSTEP independently with the LSTEP-API?
How can I use severalLSTEP-PCI cards in a single PC?
Is the LSTEP-API compatible with MCL or to the former register command-set?
Why does the message “fatal error C1010” occur in MS Visual C++ upon embedding LStep4X.cpp?
How can I use a special/new LSTEP command for which there is no suitable LSTEP-API-function?
Why do I see the message “First chance exception”, “Exception: Timeout read RS232!”, etc. in the debugger of my development environment when using the LSTEP-API?
How can I simulate a sort of joystick with the LSTEP-API, i.e. move an axis until a key is released?
How can I permanently save the settings of the LSTEP?
How many entries will fit into the log window of the LSTEP-APIs?

How is the LSTEP4X.DLL embedded in a MS Visual C++ project?

- Create project
- Copy LSTEP4X.DLL, LSTEP4X.h, LSTEP4X.cpp in project folder
- Insert LSTEP4X.h and LSTEP4X.cpp in the project
- Menu: Project\Adjustment\C/C++ Option: [do not use pre compiled headers]
- Embed "stdafx.h" inLSTEP4X.h #include
- Insert "LSTEP4X.h" in the project name_Dlg.h #include
- Embed the required entity in public
  Example: CLStep* MyLStep = new CLStep();

How do I initialize the connection to LSTEP with the LSTEP-API?

The connection to the LSTEP-API is initialized with one of the Connect commands (Connect, ConnectEx, ConnectSimple).

Which of the Connect commands should be used?

Except for some special cases, ConnectSimple should always be used.

Function	Intended use:
ConnectSimple	For the direct transfer of the interface parameters
Connect	After loading of the interface parameters out of an .ini file using Load-Config
ConnectEx	When loading the interface parameters out of a data structure

How do I initialise the driver for theLSTEPPCI?

After the correct installation of the LSTEP-PCI, Windows requests a driver for a device of the type "network controller" during the start. Click on the "Search" button or the like in this dialog window and then change into the directory to which the files of the LSTEP-API where unpacked. The sub-directory "LStep-PCI" contains the driver-files and the Inf-files required for driver installation.

Why is my program with the LSTEP4X.DLL not able to establish a connection to the LSTEP-PCI?

You should first check the Windows device manager as to whether the installed LSTEP-PCI is registered as a device there. In addition, **the file DRVX40.DLL must be contained in the directory 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.

A fault has occurred during the process, in the LSTEP4X.DLL or in my program. What is the cause for this problem, and how can I solve it?

In order to perform a fault diagnosis you should absolutely activate the log function of the LSTEP-API by SetWriteLogText. Subsequently, you should attempt to reproduce the fault while recording it. You can then send the log file (LStep4.log) to us for analysis.

Can I inquire the status of inputs, the current position, etc. during moving commands?

Yes, for example by calling GetDigitalInputs during the moving command via a Windows timer or a second Thread function like GetPos. However, it is not possible to activate the command WaitForAxisStop during a moving command with Wait=True?

Why are messages processed during the execution of LSTEP-API functions, and how can I deactivate this?

While waiting for feedback from the LSTEP in the main thread, the LSTEP-API processes messages e.g. for performing interruptions or axis stops. If you wish to deactivate message dispatching or replace it by your own code, you can use the functionSetProcessMessagesProc for setting a callback procedure.

When should moves be used with or without Wait?

Move commands with WaitForAxisStop are to be used, if all axes are to be moved synchronously and linearly interpolated. The controller only accepts new Move commands after all axes are stopped.

Move commands without WaitForAxisStop are to be used, if all axes are to be moved asynchronously. In this case the user has to make sure that only the axis, which receives a move command, is at a standstill.

How can I move single axes of the LSTEP independently from each other?

The LSTEP-API move commands offer two different possibilities:

If the Parameter Wait =True is set for move commands, the function only returns after the axes have reached there target position.

If, however, the parameter Wait =False is set, the LSTEP-API function only sends the move command and immediately returns without performing the movement.

For this reason, an independent movement can be performed by executing a MoveAbsSingleAxis with Wait=False for e.g. the X-axis; a MoveAbsSingleAxis with Wait=False for the Y-axis will called a little later. Subsequently, both axes will be moved asynchronously. You can use the Command WaitForAxisStop **in order to find out whether** the axes have reached their target positions.

Example:

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

However, it is **not possible to use Move commands with Wait=True and those with Wait =False simultaneously**. This leads to permanent or sporadic errors in communication.

Example (not permissible):

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

How can I use several LSTEP-PCI cards in a single PC?

The procedure for the installation is the same as for a single card. After the start, Windows requests the driver for all LSTEP-PCI cards.

However, it is **difficult to identify which physical card is assigned to a specific index number**. It is not guaranteed thatLSX_ConnectSimple(4, nil, 0, true) will establish a connection to the LSTEP-PCI in the first PCI slot of the mainboard , LSX_ConnectSimple(4, nil, 1, true) will establish a connection to the LSTEP-PCI in the second PCI slot, etc. For this reason, the serial number should be inquired by **GetSerialNr** for clear identification.

Is the LSTEP-API compatible with MCL or to the former register command-set?

In principle, the LSTEP-API is down-compatible with the register command set with which other MCL and previous LSTEP controllers communicate. However, this command set does not offer many of the possibilities, which the LSTEP-API can use with a new command set. For this reason, some LSTEP-API commands such as WaitForAxisStop can generally not be used by controllers with an old command-set

Why does the message "fatal error C1010" occur in MS Visual C++ upon embedding LStep4X.cpp?

This is not a fault in the file LStep4X.cpp. The message usually appears, if the compiler is looking for a pre-compiled header file and cannot find it. Should the message "fatal error C1010 precompiled header files" appear in MS Visual C++, the option "pre compiled Header-file" for LStep4X.cpp must be disabled. If you do not wish to use the MFC in your project, you should delete the line #include "stdafx.h" from LStep4X.cpp.

How can I use a special/new LSTEP command for which there is no suitable LSTEP-API function?

The LSTEP-API-function **SendString**offers the possibility to use new LSTEP commands not provided in the LSTEP-API. Please note that all commands close with #13 or \r!

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

Internal exceptions of the LSTEP4.DLL, which are only visible in the debugger, have no meaning. They serve the internal process control. An exception frequently appears in ConnectSimple because the LSTEP-API attempts to find out the command set. This leads to a timeout if the controller does not support the tested command set. The exceptions occurring may mostly be ignored in the development environment.

How can I simulate a sort of joystick with the LSTEP-API, i.e. move an axis until a key is released?

Such a key joystick can be implemented as follows:

Upon pressing the key, the axis is started with a very long vector, which is still in the travel range of the axis.Subsequently, a **MoveRelSingleAxis(Xaxis, 100000, false)** is to be executed. It is important to set the parameter Wait=False so that the program will not await the end of the movement. When the key is released, the Command **StopAxes** is to be called.

How can I permanently save the settings of the LSTEP ?

The LSTEP-API command **LstepSave** can be used to maintain settings (spindle pitches, gear factors, axes currents, etc.) made once, even after a reset of the LSTEP. Please refer to the documentation to see whether your LSTEP supports this command.

How many entries will fit into the log window of the LSTEP-API?

The log window of LSTEP-API has a capacity of over 20,000 logs. If more entries exist, the log window is deleted and re-filled.

How many logs can be written into the log file?

You can write into the log file until the programme is terminated or the hard disk is full. This behavior can be changed by using the function SetExtValue.