Programming Interface

LSTEP-API

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

0 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. Three DLLs are available: LSTEP4DLL, LSTEP4XDLL or LSTEP64DLL.

For new developments, only LSTEP4XDLL or LSTEP64DLL for 64-bit applications should now be used, as they permit the parallel control of multiple LSTEPs. LSTEP4DLL is only being developed further with restrictions, e.g. LSTEP4DLL is no longer available for applications under LabVIEW.

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-PCIexpress, and LSTEPexpress.
- 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 LSTEP-API as well as with LSTEP4X-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

LSTEP-API and LSTEP4X-API have been tested with the following development and runtime environments:

Borland/Inprise Delphi 3-7, Embarcadero Delphi XE2
Microsoft Visual C++ 6.0, 2010
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 LSTEP-APIs is the file LSTEP4.DLL. You use this DLL for developing your own programs, to configure an LSTEP, to transmit commands, to inquire position values or inputs/outputs, etc. **Please only** use **LSTEP4XDLL or LSTEP64DLL** for new development.

2.2 LSTEP4X-API

The main component of the LSTEP4x-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.

2.3 General notes

2.3.1 LSTEP4.DLL

The DLL LSTEP4.DLL implements the commands of the LSTEP-API. All functions are declared with a 32-bit integer as the return value. A return value of 0 indicates the error-free execution of the function; if errors (e.g. timeouts) occur, the relevant error code (see Section 6 Error codes) is returned.

For functions such as LS_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.

2.3.2 LSTEP4X.DLL or LSTEP64.DLL

The DLL LSTEP4X.DLL or LSTEP64.DLL implements the commands of the LSTEP4X-API. All functions are declared with a 32-bit integer as the return value. A return value of 0 indicates 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 can 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.

2.3.3 Differences between LSTEP4.DLL and LSTEP4X.DLL or LSTEP64.DLL

The functional scope of LSTEP4.DLL is identical to that of LSTEP4X.DLL or LSTEP64.DLL. The LSTEP4.DLL is continued so that the existing source code does not have to be modified.

The LSTEP4X-API or LSTEP64-API opens an own protocol window for each LSTEP, and the Log files are also written separately for each LSTEP.

Since the LSTEP4XAPI or LSTEP64-API supports multi-threading, programs made by the customer can access the LSTEP controllers from several threads through the API.

The parallel control of several LSTEP controls is possible.

The function names received different prefixes. For the LSTEP4X.DLL or LSTEP64.DLL "LSX_" instead of "LS_" is used, just as for the LSTEP4.DLL.

For all functions calls of the LSTEP4X API, an integer value identifying the controller is used as an additional parameter. The LSTEP controllers are numbered from 1 through 32.

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

2.4 Integration in Delphi

2.4.1 LSTEP4-API

All function names of the LSTEP4-API start with "LS_" for easier differentiation. In order to be able to use the functions of the LSTEP4 API, the LSTEP4.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 control of an LSTEP

Required files: LSTEP4.DLL and LSTEP4.pas

```
uses … LSTEP4, …
...
var LStep1: Integer;
…
begin
  LSX_ConnectSimple(1, 'COM1', 9600, True);
  LSX_MoveAbs(10.0, 20.0, 30.0, 0.0, True);
  LSX_Disconnect();
end;
```

2.4.2 LSTEP4X API or LSTEP64 API

All function names of the LSTEP4X-API start with "LSX_" for easier differentiation (see LStep4x.pas). In order to be able to use the functions of the LSTEP4X APIs, 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;
```

2.5 Integration in Visual C++

2.5.1 LSTEP4-API

For Visual C++, an encapsulation of the LSTEP4.DLL has been created. The class CLStep4 loads the DLL and all pointers in response to function calls dynamically. The methods of the LSTEP object is not preceded by "LS_", since differentiation is enabled by the LSTEP object.

(Example: LS.Calibrate() instead of LS_Calibrate())

From class CLStep4, only an entity should be created, in particular the LSTEP4-API can only address one controller.

Visual C++ example for the control of an LStep

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

```
...
CLStep4 LS1;
…
LS1.ConnectSimple(1, "COM1", 9600, true);
LS1.MoveAbs(10.0, 20.0, 30.0, 0.0, true);
LS1.Disconnect();
delete LS1;
```

2.5.2 LSTEP4X-API

For Visual C++, an encapsulation of the LSTEP4X.DLL or LSTEP64.DLL has been created. The class CLStep4X orCLStep64 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;
```

2.6 Integration in LabVIEW

NI LabVIEW is a development environment based on the graphic programming language C. Is 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 (LSTEP4.DLL, LSTEP4X.DLL or LSTEP64.DLL) VI libraries (LSTEP4.LLB, 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 LSTEP4.DLL or LSTEP4X.DLL is used by means of the “Call Library Function” (calling ext. libraries).

2.6.1 Differences between LSTEP4.LLB and LSTEP4X.LLB

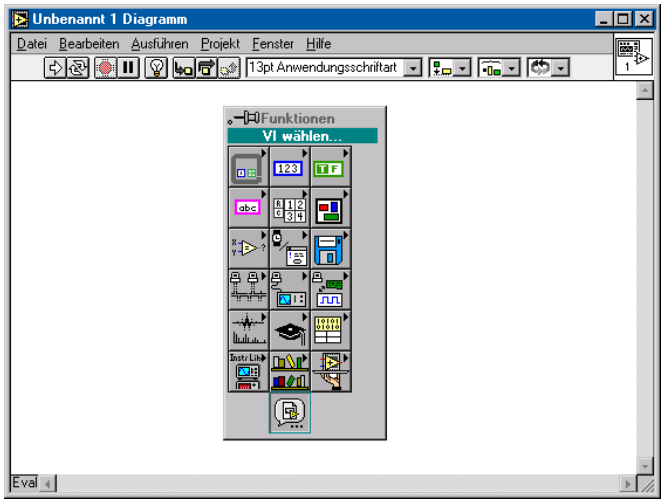
In new software projects with the LSTEP-API under LabView, you should exclusively use the VI-Library LSTEP4X.LLB or LSTEP64.LLB. The reason for this: LSTEP4.LLB supports a maximum of one LSTEP; apart from this, all VIs only have a Boolean variable as return value, i.e. no error code to be interpreted.

In the (newer) LSTEP4X.LLB or LSTEP64.LLB, the VIs have a 32-bit integer value as return value (designated as "error out"). If this is 0, it signals that the LSTEP-API function was carried out error-free. Otherwise, the meaning of the error code may be taken from Section 6 Error codes in this documentation. Apart from this, several LSTEP controllers may be controlled in parallel due to the LSTEP4X.DLL or LSTEP64.DLL called in the VIs. The VIs for the LSTEP4X.DLL or LSTEP64.DLL differ in their file name from those of the LSTEP4.DLL by starting with the prefix "LS4X" instead of "LS4". In LabView, the background colour of the VIs for the LSTEP4X.DLL or LSTEP64.DLL is purple; the background colour of the LSTEP4.DLL VIs is blue. The designation of the VIs in the symbols is identical. An additional terminal supplements all VIs. This is a 32-bit integer value and indicates the number of the LSTEP to which the command, e.g. a travel command or the reading out of the current position refers ("LSTEPController ID"). You can either assign these numbers yourself (e.g. "0" for the LSTEP at the serial interface COM1, "1" for the LSTEP at COM2 etc.), or have it created by using the VI "LS4X CreateLSID". LSTEP ID-numbers created using the VI "LS4X FreeLSID" can be "released" again. If you only use one LSTEP in your LabView project, you can leave the terminal "LSTEP Controller ID“ open for all used VIs from the LSTEP4X.LLB or LSTEP64.LLB, since its value is 1 by default.

Files required in LabView: LSTEP4X.DLL and LSTEP4X.LLB or LSTEP64.DLL and LSTEP64.LLB or LSTEP4.DLL and LSTEP4.LLB

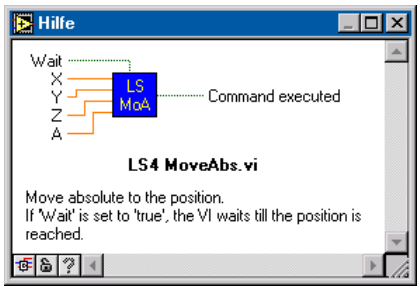
2.6.2 Procedure for using an LSTEP4 VI

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

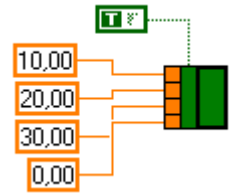


4. Select VI...
5. Open the supplied VI Library LSTEP4.LLB in the file dialog, and then select the required command (e.g. LS4 MoveAbs.vi)
6. Place VI in the diagram

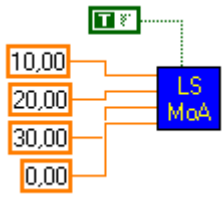
Press Ctrl+H to open a help window which gives you information about the VI on which the mouse pointer is currently located



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 is: 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.

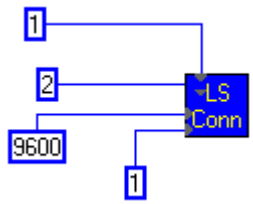


For details of the VI terminal assignment, please refer to the documentation for the API function in question, where you will find a diagram which is also shown in the Help window of LabVIEW. The parameters are more or less identical to those of the DLL function: Only are there some differences as regards 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.

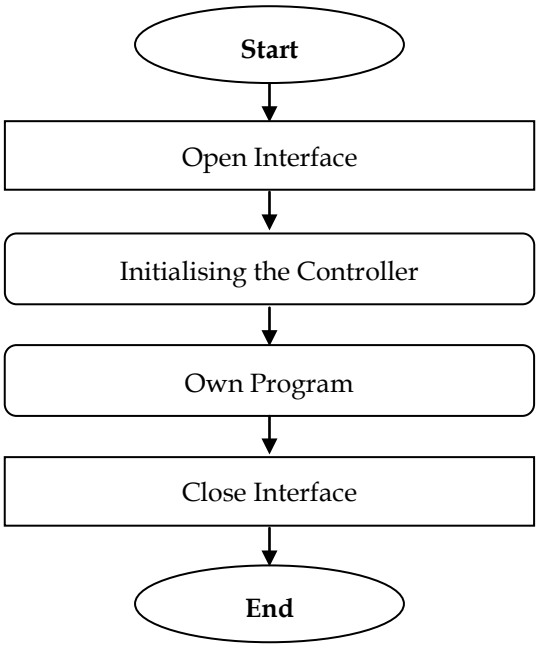
Example for RS232 (COM2 and 9600 Baud):



3 Notes regarding the development of own programs for programming the controller via 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. The 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

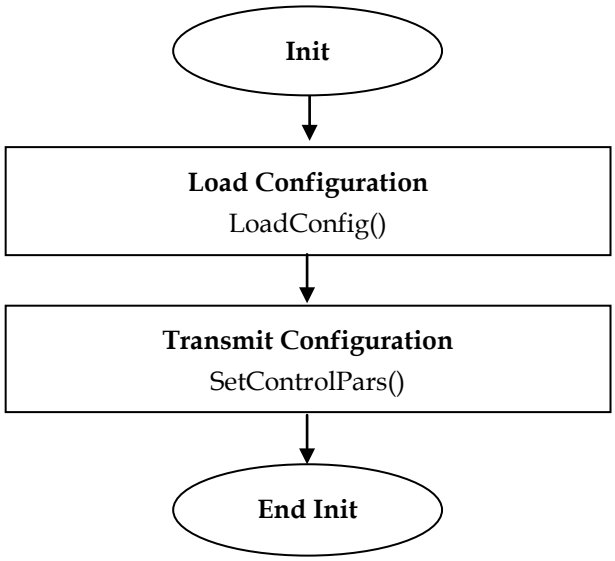
The interface is opened using one of the functions Connect, ConnectEx or ConnectSimple, with ConnectSimple being recommended for establishing the connection. The interface to be opened for connecting to a controller depend on the control type and the interface settings. The default settings of the interface 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	11
	MCL	Register	x					
LSTEP Series	LSTEP-xx	Register	x					
	LSTEP-PC	Register			x			
LSTEP 2000 Series	LSTEP-PCI	Ipreter Register	x			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					
LSTEPexpress Series	LSTEP-PCIexpress	Ipreter					x	x
	LSTEPexpress	Ipreter					x	x

3.2 Initialising the Controller

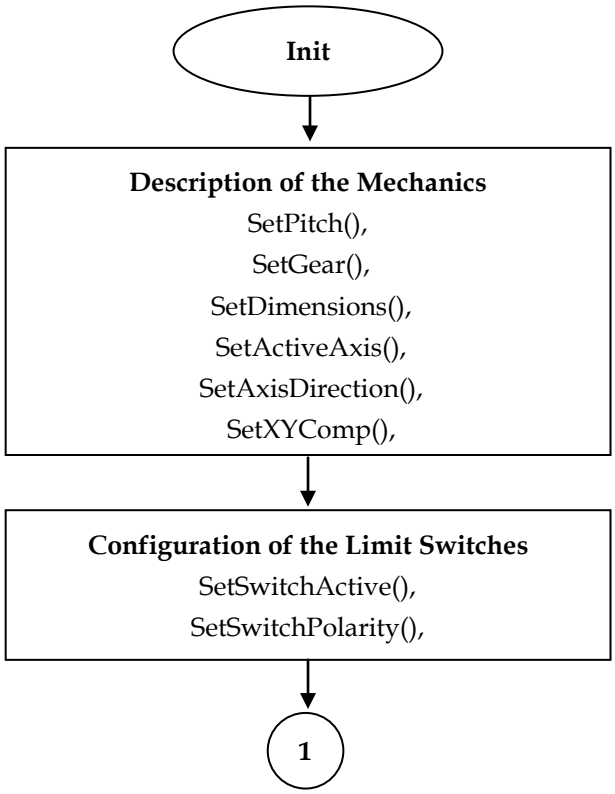
3.2.1 General

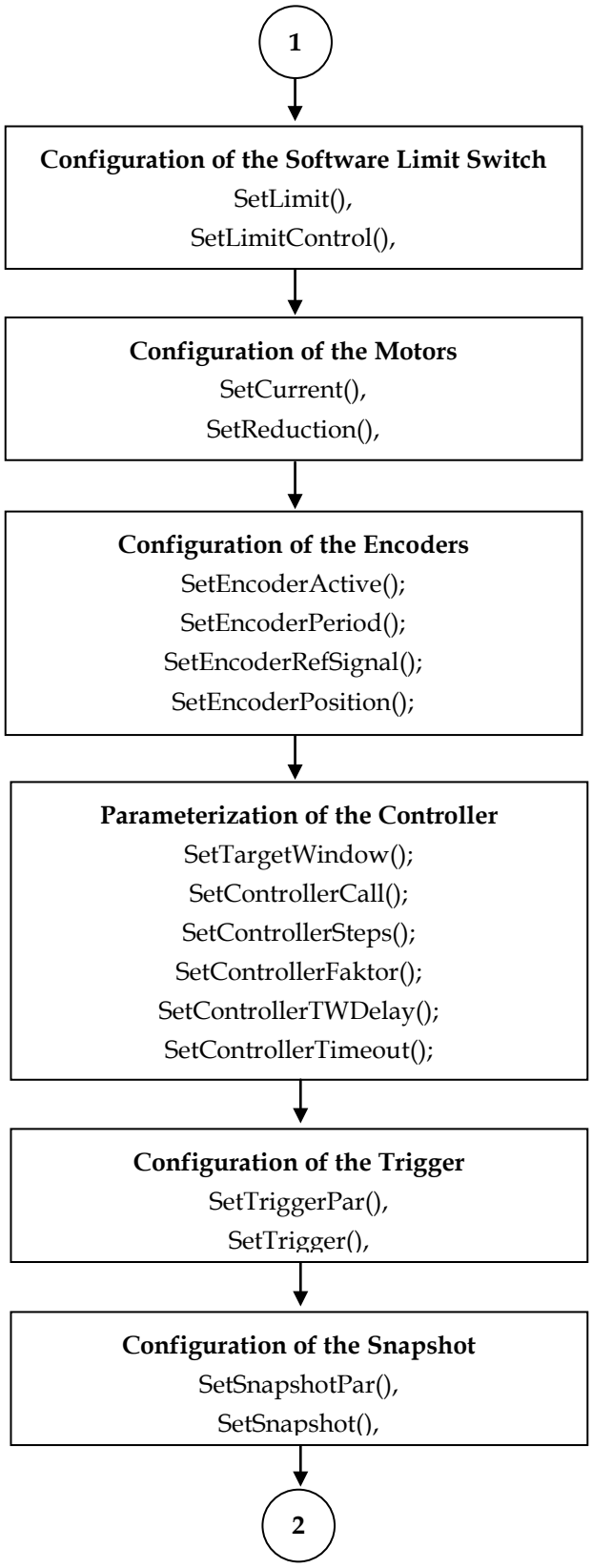
Prior to starting the own program part, the LSTEP controller should be configured. For this purpose, e.g. the commissioning software WIN-Commander may be used to create a configuration file. This file may subsequently be loaded by the API and transmitted to the controller.

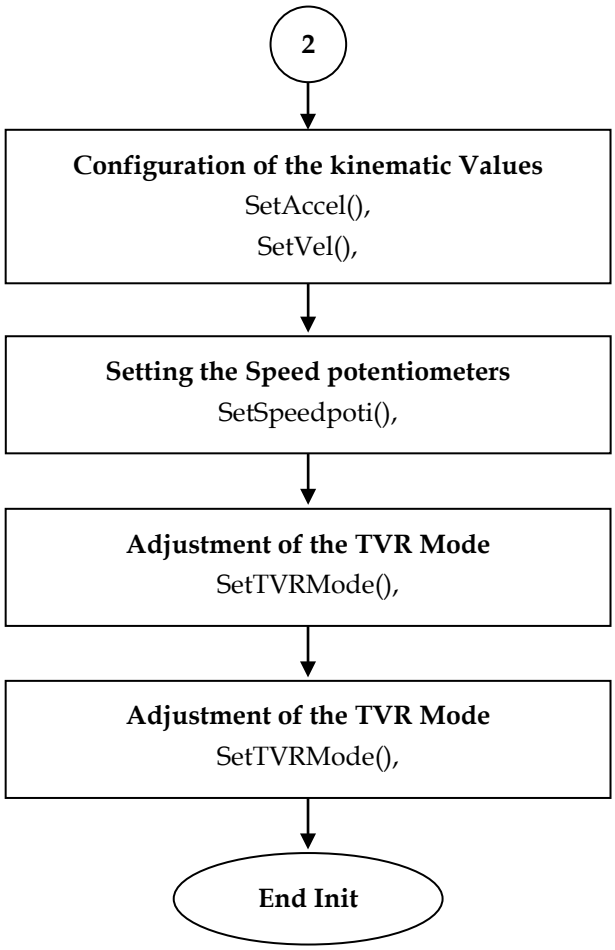


The settings in the controller may be saved if the API makes no general modifications of the control settings. You can use the commissioning software WIN-Commander for this purpose, too.

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.


Init
Description of the Mechanics
SetPitch(),
SetGear(),
SetDimensions(),
SetActiveAxis(),
SetAxisDirection(),
SetXYComp(),
Configuration of the Limit Switches
SetSwitchActive(),
SetSwitchPolarity(),
1


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 WinCommander 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.

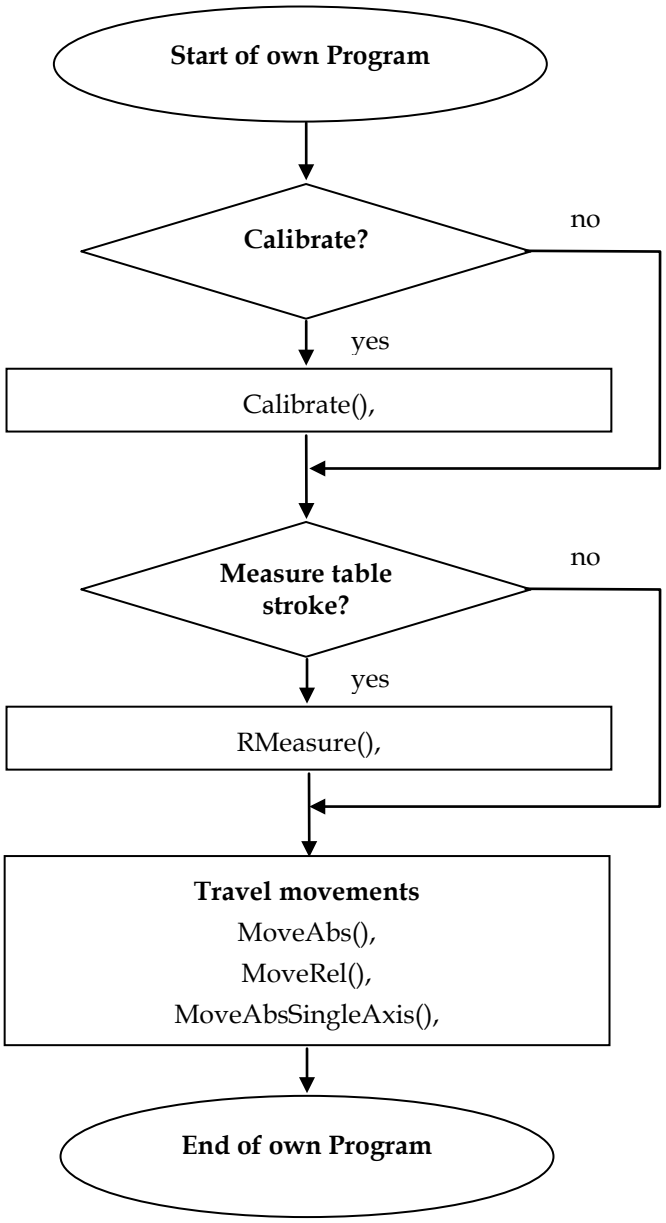
3.2.3 LSTEPexpress / LSTEP-PCIexpress controllers

LSTEPexpress 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 LSTEPexpress by using the WIN-Commander 5. This configuration 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 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 column "X": 1 = LSTEP; 2 = LSTEPexpress; 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	Creates an ID No. for using the LSTEP4X APIs	3	31
FreeLSID	Releases the created ID No again	3	32
Connect	Connect with LSTEP	3	32
ConnectEx ConnectExW	Connect with LSTEP	3	33
ConnectSimple ConnectSimpleW	Connect with LSTEP (recommended)	3	34
Disconnect	Disconnect LSTEP	3	35
SetExtValue	Activate extensions	3	36
SetProcessMessagesProc	Allows the substitution of the internal message dispatching procedure of the LSTEP-API	3	37
SetOsziCallBackFct	Transfers a global callback function for asynchronous results to the API. (see Section 5.1)	2	37
SetExtCallBackFct	Transfers a global callback function for asynchronous results to the API. (See 5.2)	2	38
SetLanguage SetLanguageW	Set language for LSTEP-API (log/messages)	3	39
TranslateErrMsg	Translates an error message sent by the controller	2	39

4.1.2 Control and API information

Command	Brief description	X	Page
GetAPIVersion GetAPIVersionW	Shows version number of LSTEP-API	3	49
GetSerialNr GetSerialNrW	Read out controller serial number	1	49
GetVersionStr GetVersionStrW	Feeds back the current firmware version number	3	50
GetVersionStrDet GetVersionStrDetW	Read out firmware configuration	1	51
GetVersionStrInfo GetVersionStrInfoW	Read out supplement of current version number	3	51

4.1.3 Status requests

Command	Brief description	X	Page
GetError	Shows the current error number	3	52
GetSecurityErr	Reads all statuses and results of GAL safety monitoring (only with LS44-controllers)	1	52
GetSecurityStatus	Reads the current statuses of safety monitoring system	1	54
GetStatus GetStatusW	Shows the current status of the controller	3	55
GetStatusAxis GetStatusAxisW	Shows the present status of the individual axes	3	55
SetAutoStatus	AutoStatus On/Off	3	56
GetStatusLimit GetStatusLimitW	Shows the current state of the software limits of each axis.	3	57

4.1.4 Parameter handling

Command	Brief description	X	Page
SetControlPars	Transmits the parameters loaded with LS_LoadConfig to the LSTEP.	3	58
LstepSave	Saves current configuration in LSTEP (EEPROM)	3	58
SoftwareReset	Restarts the controller	3	59

4.1.5 Axis and motor configuration

Command	Brief description	X	Page
ConfigMaxAxes	Sets the number of axes used	3	60
GetDimensions	Inquires the axis dimensions	3	60
SetDimensions	Sets the axis dimensions	3	61
GetActiveAxes	Shows the released axes	3	61
SetActiveAxes	Shows the released axes	3	62
GetAxisDirection	Inquires the reversal of direction	3	63
SetAxisDirection	Sets the reversal of direction	3	63
GetGear	Reads the gear factor	1	64
SetGear	Sets gear factor	1	64
GetGearDenominator	Inquires the denominator of the gear ratio	2	65
SetGearDenominator	Sets the denominator of the gear ratio	2	65
GetGearNumerator	Inquires the numerator of the gear ratio	2	66
SetGearNumerator	Sets the numerator of the gear ratio	2	66
GetMotorCurrent	Inquires the motor current	1	67
SetMotorCurrent	Sets the motor current	1	67
GetReduction	Inquires the current reduction	3	68
SetReduction	Sets the current reduction	3	68
GetCurrentDelay	Indicates the time delay for the current reduction	3	69
SetCurrentDelay	Sets the time delay for the current reduction	3	69
GetMotorType	Shows the set motor type	2	70
SetMotorType	Sets the motor type	2	70
GetMotorFieldDir	Inquires the motor field direction	2	71
SetMotorFieldDir	Sets the motor field direction	2	72
GetMotorMaxVel	Shows the set max. motor speed	2	72
SetMotorMaxVel	Sets the max. adjustable motor speed	2	73
GetMotorTablePatch	Indicates whether the correction table is activated	1	73
SetMotorTablePatch	Activates or deactivates the correction table	1	74
GetPitch	Shows the spindle pitches	3	74
SetPitch	Sets the spindle pitch	3	75
GetXYAxisComp	Inquires the XY axis overlay	1	75
SetXYAxisComp	Activates the XY axis overlay	1	76
GetStopPolarity	Reads the stop polarity	3	76
SetStopPolarity	Sets the stop polarity	3	77

Command	Brief description	X	Page
GetPowerAmplifier	Inquires whether the power amplifiers are activated or deactivated (only for LS44 controller and LSTEPexpress)	3	77
SetPowerAmplifier	Activates or deactivates the power amplifiers (only for LS44 controller and LSTEPexpress)	3	78

4.1.6 Kinematics

Command	Brief description	X	Page
GetAccel	Inquires the acceleration	3	79
SetAccel	Sets the acceleration	3	79
SetAccelSingleAxis	Sets the acceleration of a single axis	3	80
GetAccelJerk	Inquires the jerk for acceleration	2	80
SetAccelJerk	Sets the jerk for acceleration	2	81
GetDeceleration	Inquires the deceleration	2	81
SetDeceleration	Sets the deceleration	2	82
SetDecelSingleAxis	Sets the deceleration of a single axis	2	82
GetDecelJerk	Inquires the jerk during deceleration	2	83
SetDecelJerk	Sets the jerk during deceleration	2	83
GetVel	Inquires the velocity of all axes	3	84
SetVel	Sets the velocity of all axes	3	84
SetVelSingleAxis	Sets the velocity for a single axis	3	85
GetVelFac	Inquires the velocity reduction of all axes	1	85
SetVelFac	Sets the velocity reduction of all axes	1	86
SetVelFacSingleAxis	Sets the velocity reduction of a single axis	1	86
GetVLevel	Shows the hidden speed	1	87
SetVLevel	Sets the hidden speeds at which resonances occur	1	87
GetSpeedPoti	Indicates whether the speed potentiometer is activated or deactivated	1	88
SetSpeedPoti	Activates or deactivates the speed potentiometer	1	88
GetStopAccel	Shows the braking acceleration for an active stop	1	89
SetStopAccel	Sets the braking acceleration for an active stop	1	89
GetStopDecel	Reads the value at which the axis shall decelerate in case of a Stop signal	2	90
SetStopDecel	Sets the value at which the axis shall decelerate in case of a Stop signal	2	90
GetStopDecelJerk	Inquires the jerk during system delay in case of an Emergency Stop signal	2	91

Command	Brief description	X	Page
SetStopDecelJerk	Sets the jerk during system delay in case of an Emergency Stop signal	2	91

4.1.7 Limit switches and software limits

Command	Brief description	X	Page
GetLimitControl	Reads whether if travel range control is active	3	93
SetLimitControl	Sets the travel range control	3	93
GetLimitControlMode	Shows the mode for controlling the software limits	1	94
SetLimitControlMode	Sets the mode for controlling the software limits	1	94
GetAutoLimitAfterCalibRM	Indicates whether the internal software limits are set during calibration and table stroke measuring	3	95
SetAutoLimitAfterCalibRM	Prevents that the internal software limits are set during calibration and table stroke measuring	3	96
GetLimit	Shows travel range limits	3	93
SetLimit	Sets travel range limits	3	93
GetSwitchActive	Indicates whether limit switches are activated	3	98
SetSwitchActive	Sets the status of activated or deactivated limit switches	3	98
GetSwitchPolarity	Reads the limit switch polarity	3	99
SetSwitchPolarity	Sets the limit switch polarity	3	100
GetSwChange	Shows the limit switch settings	2	100
SetSwChange	Sets the value if limit switches are to be changed	2	101
GetSwitches	Reads the status of all limit switches	3	101

4.1.8 Reference travel

Command	Brief description	X	Page
GetCalibRMAccel	Inquires the acceleration during the calibration procedure	2	103
SetCalibRMAccel	Sets the acceleration during the calibration procedure	2	103
GetCalibRMJerk	Inquires the jerk during calibration	2	104
SetCalibRMJerk	Sets the jerk during calibration	2	104
GetCalibBackSpeed	Shows the speed at which limit switches are left	1	105
SetCalibBackSpeed	Sets the positioning speeds for moving out of the limit switches during the calibration procedure	1	105
GetCalibRMBackSpeed	Shows the speed at which limit switches are left	2	106
SetCalibRMBackSpeed	Sets the positioning speeds for moving out of the limit switches during the calibration procedure	2	106
GetRefSpeed	Inquires the speed at which the reference mark is searched during calibration	1	108

Command	Brief description	X	Page
SetRefSpeed	Sets the speed at which the reference mark is searched during calibration	1	108
GetRMOffset	Inquires the set RM offset	3	110
SetRMOffset	Sets the RM offset	3	110
GetCalibRMVel	Inquires the travel velocity during the calibration procedure	2	107
SetCalibRMVel	Sets the travel velocity during the calibration procedure	2	107
GetCalibOffset	Inquires the calibration offset	3	109
SetCalibOffset	Sets the calibration offset	3	109
GetCalibrateDir	Shows whether the sign reversal is set for calibration	3	111
SetCalibrateDir	Sets the sign reversal for calibration	3	111
Calibrate	Starts calibration for all active axes	3	112
CalibrateEx	Starts calibration for specific axes	3	112
RMeasure	Starts table stroke measuring for all active axes	3	113
RmeasureEx	Starts table measuring for specific axes	3	113

4.1.9 Travel commands and position administration

Command	Brief description	X	Page
GetPos	Inquires the current position of all axes	3	114
GetPosEx	Inquires the current encoder or position values of all axes	3	114
GetPosSingleAxis	Inquires the current position of a single axis	3	115
SetPos	Sets the position of all axes	3	116
ClearPos	Sets the position values to zero (for infinite rotation axes)	1	116
GetDelay	Shows the delay of the vector start	1	117
SetDelay	Sets the delay of the vector start	1	117
GetDistance	Shows the distance started by LS_MoveRelShort	3	118
SetDistance	Sets the distance for LS_MoveRelShort	3	118
GetInputTrigMove	Shows the configuration of Pin1 on the MFP	1	119
SetInputTrigMove	Configures Pin 1 on the MFP	1	119
MoveAbs	Approaches an absolute position with all axes	3	120
MoveAbsSingleAxis	Moves to an absolute position with a single axis	3	121
MoveEx	Enables extended travel commands	3	121
MoveRel	Moves to a relative vector with all axes	3	123
MoveRelShort	Moves to a relative vector as a short command	3	123
MoveRelSingleAxis	Moves to a relative vector with a single axis	3	124

Command	Brief description	X	Page
StopAxes	Stops all travel movements	3	124
WaitForAxisStop	Waits for the movement stop of specific axes	3	125

4.1.10 Joystick and handwheel

Command	Brief description	X	Page
GetHandwheel	Reads the hand wheel status	1	126
SetHandWheelOff	Deactivates the handwheel	1	126
SetHandwheelOn	Activates the handwheel	1	127
GetDigJoySpeed	Reads the speed of the digital joystick	1	128
SetDigJoySpeed	Sets the speed of the digital joystick	1	128
SetDigJoyOff	Deactivates the digital joystick	1	129
GetJoystick	Reads the status of the analogue joystick	3	129
SetJoystickOff	Deactivates the analogue joystick	3	130
SetJoystickOn	Activates the analogue joystick	3	130
GetJoystickFilter	Indicates whether the filtering is active in joystick operation	1	131
SetJoystickFilter	Activates or deactivates the filtering in joystick operation	1	132
GetJoystickWindow	Reads the joystick window	3	132
SetJoystickWindow	Sets the joystick window	3	133
GetJoyChangeAxis	Reads the joystick axis assignment	1	133
JoyChangeAxis	Sets the joystick axis assignment	1	134
GetJoystickAxes	Shows the axes for which the joystick is activated	2	134
SetJoystickAxes	Activates the joystick for the specified axes	2	135
GetJoystickDir	Reads the set joystick direction	3	136
SetJoystickDir	Sets the joystick direction	3	136
GetJoyVel	Inquires the maximum positioning speed in joystick operation	2	137
SetJoyVel	Sets the max. positioning speed in joystick operation	2	138

4.1.11 Control panel with trackball and joystick keys

Command	Brief description	X	Page
GetBPZ	Reads the control panel status	1	139
SetBPZ	Activates or deactivates the control panel	1	139
GetBPZJoyspeed	Reads the control panel joystick speed	1	140
SetBPZJoyspeed	Sets the control panel joystick speed	1	140
GetBPZTrackballBacklash	Reads the control panel trackball backlash	1	141
SetBPZTrackballBacklash	Sets the control panel trackball backlash	1	141
GetBPZTrackballFactor	Reads the control panel trackball factor	1	142
SetBPZTrackballFactor	Sets the control panel trackball factor	1	142

4.1.12 Digital and analogue inputs and outputs

Command	Brief description	X	Page
GetAnalogInput	Reads the current status of an analogue channel	3	143
GetAnalogInputs2	Reads the current statuses of the analogue channels PT100, MV and V24	1	143
SetAnalogOutput	Sets and analogue channel	3	144
GetDigitalInputs	Reads the digital inputs (0-15)	3	144
GetDigitalInputsE	Reads the additional digital inputs (16-31)	1	145
SetDigitalOutput	Sets a digital output	3	145
SetDigitalOutputs	Reads the digital outputs (0-15)	3	146
SetDigitalOutputsE	Sets the additional digital outputs (16-31)	1	147
SetDigIO_Distance	Sets the activation of an output dependent on the set distance before/behind of the target position	1	147
SetDigIO_EmergencyStop	Sets the assignment of digital I/O to Emergency Stop pin	1	148
SetDigIO_Off	Deactivates function of the digital inputs/outputs	1	149
SetDigIO_Polarity	Sets the polarity to the functions of the digital outputs	1	149

4.1.13 Cycle Forward / Back In

Command	Brief description	X	Page
GetFactorTVR	Reads the Forward/ Back cycle factor	3	151
SetFactorTVR	Sets the Forward/ Back cycle factor	3	151
GetTVRMode	Reads the Cycle Forward/ Back mode	1	152
SetTVRMode	Sets the Forward/ Back cycle mode	1	152
SetTVRInPulse	Sets the Forward/Back pulse via the interface	1	153

4.1.14 Cycle Forward / Back outputs for additional axes

Command	Brief description	X	Page
GetAccelTVRO	Reads all adjusted acceleration values of the TVRO	1	154
SetAccelTVRO	Sets the TVRO acceleration values	1	154
SetAccelSingleAxisTVRO	Sets the individual TVRO acceleration values	1	155
GetVelTVRO	Reads all adjusted speeds of the TVRO	1	155
SetVelTVRO	Sets all speeds of the TVRO	1	156
SetVelSingleAxisTVRO	Sets the speeds of a single TVRO axis	1	156
GetPosTVRO	Shows the TVRO position values dependent on the dimension	1	157
SetPosTVRO	Sets the TVRO position	1	157
GetStatusTVRO GetStatusTVROW	Shows the current status of the axes	1	158
GetTVROutMode	Reads the set TVRO mode	1	159
SetTVROutMode	Sets the TVRO offset	1	159
GetTVROutPitch	Reads the TVRO spindle pitch	1	160
SetTVROutPitch	Sets the TVRO spindle pitch	1	160
GetTVROutResolution	Shows the resolution of the TVRO power amplifier to be controlled	1	161
SetTVROutResolution	Sets the resolution of the TVRO power amplifier to be controlled	1	161
MoveAbsTVRO	Moves to an absolute position with all TVRO axes	1	162
MoveAbsTVROSingleAxis	Moves to an absolute position with a single TVRO axis	1	162
MoveRelTVRO	Moves to a relative vector with all TVRO axes	1	164
MoveRelTVROSingleAxis	Moves to a relative vector with a single TVRO axis	1	164

4.1.15 Encoder settings

Command	Brief description	X	Page
ClearEncoder	Sets the encoder position to zero	1	165
GetEncoder	Reads all encoder positions	1	165
GetEncoderActive	Reads, which encoders are activated after calibration	1	166
SetEncoderActive	Sets the encoders to be activated after calibration	1	166
GetEncoderMask	Reads the encoder statuses	3	167
SetEncoderMask	Activates or deactivates the encoders	1	168
GetEncoderPeriod	Reads the set encoder period lengths	3	168
SetEncoderPeriod	Sets the encoder period lengths	3	169
GetEncoderPosition	Shows the set position source	3	169
SetEncoderPosition	Sets the position source	3	170
GetEncoderRefSignal	Reads whether the encoder reference signal is to be evaluated during calibration	3	170
SetEncoderRefSignal	Sets whether the encoder reference signal is to be evaluated during calibration	3	171

4.1.16 Controller settings

Command	Brief description	X	Page
GetController	Reads the set controller mode	1	172
SetController	Sets the controller mode	1	172
GetControllerCall	Reads the controller call setting	1	173
SetControllerCall	Sets the controller call	1	174
GetControllerFactor	Reads the controller factor setting	1	174
SetControllerFactor	Sets the controller factor	1	175
GetControllerSteps	Reads the controller steps	1	175
SetControllerSteps	Sets the controller steps	1	176
GetControllerTimeout	Shows the controller monitoring timeout setting	1	176
SetControllerTimeout	Sets the controller monitoring timeout	1	177
GetControllerTWDelay	Reads the controller delay	1	177
SetControllerTWDelay	Sets the controller delay	1	178
GetCtrFastMove	Reads setting of the Fast Move Function	1	178
SetCtrFastMoveOff	Sets the fast move function "OFF"	1	179
SetCtrFastMoveOn	Sets the Fast Move function "ON"	1	179
GetCtrFastMoveCounter	Reads the number of Fast Move functions carried out	1	180
ClearCtrFastMoveCounter	Sets the number of Fast Move functions carried out to 0	1	180

Command	Brief description	X	Page
GetTargetWindow	Shows the controller target window	3	181
SetTargetWindow	Sets the controller target window	3	181

4.1.17 Trigger output

Command	Brief description	X	Page
GetTrigger	Reads the trigger setting	3	182
SetTrigger	Activates or deactivates the trigger	3	182
GetTrigCount	Reads the trigger counter	3	183
SetTrigCount	Sets the trigger counter	3	183
GetTriggerPar	Reads the trigger parameters	3	183
SetTriggerPar	Sets the trigger parameters	3	184

4.1.18 Snapshot input

Command	Brief description	X	Page
GetSnapshot	Reads the snapshot setting	3	186
SetSnapshot	Activates or deactivates the snapshot	3	186
GetSnapshotFilter	Reads the input filter	3	187
SetSnapshotFilter	Sets the input filter	3	187
GetSnapshotPar	Reads the snapshot parameters	3	187
SetSnapshotPar	Sets the snapshot parameters	3	188
GetSnapshotCount	Reads the snapshot counter	3	189
GetSnapshotPos	Reads the snapshot position	3	189
GetSnapshotPosArray	Reads the snapshot position array	3	190

4.2 Detailed functional description

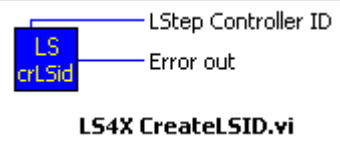
The detailed functional descriptions below contain a descriptive text for the function, a description of the function parameters, an example call of the function as well as the prototypes for the programming languages Delphi and C++. In addition, the 'Other' column includes notes as to which controller 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 LSTEPexpress controller series.

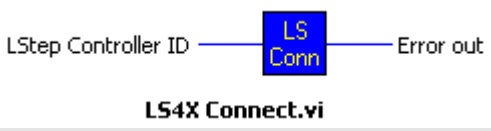
A table excerpt showing the 'Other' column is

Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	(compatible with X, see Section 4.1)	(used command from the controller command set)	(activation possible by the indicated "SendSting" commands)

4.2.1 API-Configuration/Interface Configuration

LSX_CreateLSID (only LSTEP4X- and LSTEP64-API)			
Description:	Creates an LSTEPID number. This is used as an additional parameter in the LSTEP4X-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 (LSTEPexpress series)
	3	-	-

LSX_FreeLSID (only LSTEP4X- and LSTEP64-API)			
Description:	Releases a created LSTEPID number. This is used as an additional parameter in the LSTEP4X-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.		
Delphi:	function LSX_FreeLSID(LSID: Integer): Integer;		
C++:	-		
LabView:			
Parameter:	LSID	LSTEP ID number to be released This may no longer be used after FreeLSID.	
Example:	var LStep1: Integer; ... LSX_CreateLSID(&LStep1); LSX_ConnectSimple(LStep1,...); ... LSX_Disconnect(LStep1); LSX_FreeLSID(LStep1);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

LS_Connect	
Description:	This function establishes a connection to an LSTEP controller. For this purpose, the interface parameters loaded from the INI file via LS_LoadConfig are used. (One of the functions LS_Connect, LS_ConnectSimple or LS_ConnectEx must be called for initialising of the interface so that the communication with the LSTEP is possible.)
Delphi:	function LS_Connect(): Integer; function LSX_Connect(LSID: Integer): Integer;
C++:	int Connect();
LabView:	

Parameter:	-		
Example:	LS.LoadConfig("C:\LStepTest\LStep.INI"); LS.Connect();		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

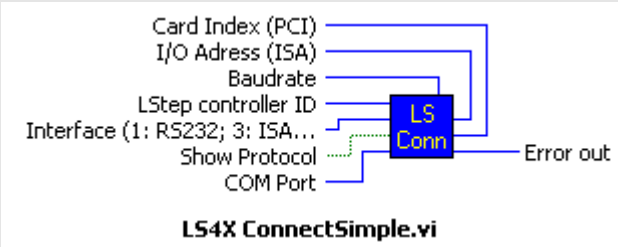
LS_ConnectEx, LS_ConnectExW			
Description:	This function establishes a connection to an LSTEP controller. A pointer is transferred to a data structure containing the interface parameters. This record also returns information on the identified controller (version number...). (One of the functions LS_Connect, LS_ConnectSimple or LS_ConnectEx must be called for initialising of the interface so that the communication with the LSTEP is possible.)		

Delphi:	function LS_ConnectEx(var AControlInitPar: TLS_ControlInitPar): Integer;
	function LS_ConnectExW(var AControlInitPar: TLS_ControlInitPar): Integer;
	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:	LS.ConnectEx(&ControlInitPar1);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

LS_ConnectSimple, LS_ConnectSimpleW		
Description:	This function establishes a connection to an LSTEP controller. The settings of the interface are delivered as parameters. (One of the functions LS_Connect, LS_ConnectSimple or LS_ConnectEx must be called for initialising of the interface so that the communication with the LSTEP is possible.)	

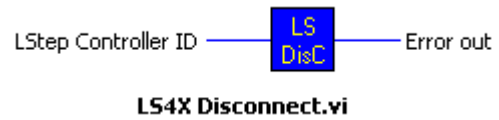
Delphi:	function LS_ConnectSimple(AnInterfaceType: Integer; AComName: PAnsiChar; ABR: Integer; AShowProt: LongBool): Integer;
	function LS_ConnectSimpleW(AnInterfaceType: Integer; AComName: PWideChar; ABR: Integer; AShowProt: LongBool): Integer;
	function LSX_ConnectSimple(LSID: Integer; AnInterfaceType: Integer; AComName: PAnsiChar; ABaudRate: Integer; AShowProt: LongBool): Integer;
	function LSX_ConnectSimpleW(LSID: Integer; AnInterfaceType: Integer; AComName: PWideChar; ABaudRate: Integer; AShowProt: LongBool): Integer;	

C++:	int ConnectSimple (int lAnInterfaceType, char *pcAComName, int lABR, BOOL AShowProt); int ConnectSimpleW (int lAnInterfaceType, TCHAR *pcAComName, int lABR, BOOL AShowProt);	
LabView:		
Parameter:	AnInterfaceType	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
	AComName	Name of the COM interface, e.g. 'COM2'; to be set to ZERO with ArcNet or DPRAM

	ABR	The meaning is independent of the interface type RS232 = baud rate, z. B. 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	
	AShowProt	Show interface log True = indicate False = do not hide	

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

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

LS_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 LS_Disconnect: Integer; function LSX_Disconnect(LSID: Integer): Integer;		
C++:	int Disconnect ();		
LabView:			
Parameter:	-		
Example:	LS.Disconnect();		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

<table>
<tr><th colspan="4">LS_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 LS_SetExtValue(AName: Integer; AValue: Integer): Integer;
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 renamed 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">LS.SetExtValue(3, 1); // Filter for move commands On
LS.SetExtValue(4, 10000); // maximum length of log file = 10000 lines
LS.SetExtValue(5, THREAD_PRIORITY_HIGHEST);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEPexpress series)</td></tr>
<tr><td>3</td><td>-</td><td>-</td></tr>
</table>

LS_SetProcessMessagesProc			
Description:	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 the 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.		
Delphi:	function LS_SetProcessMessagesProc(Proc: Pointer): Integer; function LSX_SetProcessMessagesProc(LSID: Integer; Proc: Pointer): Integer;		
C++:	int SetProcessMessagesProc(void* pProc);		
LabView:			
Parameter:	pProc	Pointer to substitute function This must be a pointer to a stdcall procedure without a parameter: void MyProcessMessages ()	
Example:	LS.SetProcessMessagesProc(&MyProcessMessages);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

LS_SetOsziCallBackFct	
Description:	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 distinguish any LSTEP entities. If a controller-specific CallBack function is to be called, the function LS_SetExtCallBackFct has to be used. As long as no controller-related CallBack function has been assigned via the function LS_SetExtCallBackFct the function assigned by LS_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 LSTEP-API".
Delphi:	LS_SetOsziCallBackFct(Fct: Pointer): Integer; LSX_SetOsziCallBackFct(LSID: Integer; Fct: Pointer): Integer;
C++:	int SetOsziCallBackFct (void* pFct)
LabView:	-

Parameter:	pFct	Pointer to CallBack function. For function declaration, please refer to "5 CallBack functions of LSTEP-API".	

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

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

LS_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 LS_SetExtCallBackFct the function assigned by LS_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 LSTEP-API".		
Delphi:	LS_SetExtCallBackFct(Fct: Pointer; pprivate: Pointer): Integer; 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 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)
	{
	…
	}
	…
	LS.SetExtCallBackFct(LSTEPExtCallBackFct);
	//activate extended log:
	LS.SendString("!errorchannel 2\r", 0, 0, false, 1000);		

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

LS_SetLanguage, LS_SetLanguageW			
Description:	Set language for LSTEP-API (log/messages)		

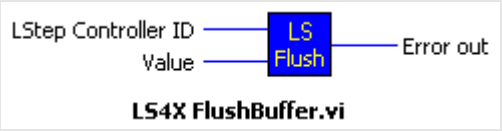
Delphi:	function LS_SetLanguage(PLN: PAnsiChar): Integer;
	function LS_SetLanguageW(PLN: PWideChar): Integer;
	function LSX_SetLanguage(LSID: Integer; PLN: PAnsiChar): Integer;
	function LSX_SetLanguageW(LSID: Integer; PLN: PWideChar): Integer;		

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:	LS.SetLanguage('ENG');		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

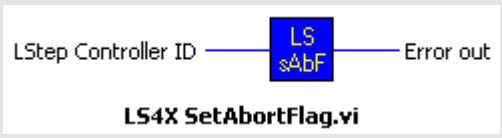
LS_TranslateErrMsg		
Description:	Translates error message transferred by a CallBack function.	
Delphi:	LS_TranslateErrMsg(MsgIn: PWideChar; MsgOut: PWideChar; MaxLen: Integer): Integer; LSX_TranslateErrMsg(LSID: Integer; MsgIn: PWideChar; MsgOut: PWideChar; MaxLen: Integer): Integer;	
C++:	int TranslateErrMsg (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));
	LS.TranslateErrMsg(InputString, TranslatedString, 255);
	
	… // process TranslatedString		

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

LS_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 LS_FlushBuffer(AValue: Integer): Integer; function LSX_FlushBuffer(LSID: Integer; AValue: Integer): Integer;		
C++:	int FlushBuffer (int lAValue);		
LabView:			
Parameter:	AValue	Currently not used, can be set to 0	
Example:	LS.FlushBuffer(0);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

LS_SetAbortFlag	
Description:	Set flag so that communication with LSTEP is disrupted. A function which awaits for a feedback from the controller when LS_SetAbortFlag is called (e.g. travel commands) and returns with 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 LS_SetAbortFlag: Integer; function LSX_SetAbortFlag(LSID: Integer): Integer;
C++:	int SetAbortFlag ();

LabView:			
Parameter:	-		
Example:	LS.SetAbortFlag(); LS.StopAxes(); (terminate communication with the LSTEP and send the command to stop all axes)		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

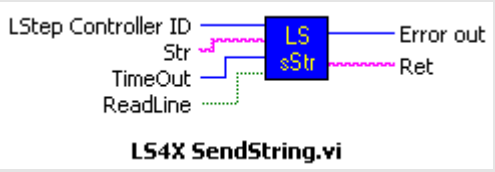
LS_EnableCommandRetry			
Description:	This function is used to activate/deactivate the repeated sending of commands in case errors (activated by default).		
Delphi:	function LS_EnableCommandRetry(AValue: LongBool): Integer; function LSX_EnableCommandRetry(LSID: Integer; AValue: LongBool): Integer;		
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:	LS.EnableCommandRetry(false) ;		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

LS_SetCommandTimeout			
Description:	Sets timeouts for waiting for acknowledgement with regard to general API calls, positioning and calibration. The default value for the timeout is 1000 ms.		
Delphi:	function LS_ SetCommandTimeout (AtoRead, AtoMove, AtoCalibrate: Integer): Integer; LSX_SetCommandTimeout(LSID: Integer; AtoRead, AtoMove, AtoCalibrate: Integer): Integer;		
C++:	int SetCommandTimeout (int lAtoRead, int lAtoMove, int lAtoCalibrate) ;		

LabView:			
Parameter:	AtoRead	Timeout for waiting for general acknowledgement of an API call in ms	
	AtoMove	Timeout for positioning in ms	
	AtoCalibrate	Timeout for calibration in ms	
Example:	LS. SetCommandTimeout (int lAtoRead, int lAtoMove, int lAtoCalibrate) ;		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

LS_SendString, LS_SendStringW	
Description:	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.

Delphi:	function LS_SendString(Str, Ret: PAnsiChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer;
	function LS_SendStringW(Str, Ret: PWideChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer;
	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;

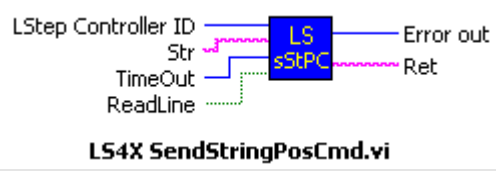
C++:	int SendString (char *pcStr,char *pcRet,int lMaxLen,BOOL ReadLine,int lTimeOut); int SendStringW (TCHAR *pcStr,TCHAR *pcRet,int lMaxLen,BOOL ReadLine,int lTimeOut);
LabView:	

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 that can be copied into the buffer for acknowledgement.
	ReadLine	Waiting for acknowledgement from LSTEP controller True = Acknowledgement expected False = No acknowledgement
	TimeOut	Maximum waiting time for acknowledgement in ms.
Example:	LS.SendString("?ver\r", pcLStepVer, 256, true, 1000); // Read version number, timeout 1s	

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

LS_SendStringPosCmd, LS_SendStringPosCmdW	
Description:	Sends a string as a moving command to the controller, which may await the axis, mask (see LS_GetStatusAxis, LS_GetStatusAxisW) as acknowledgement from the controller.

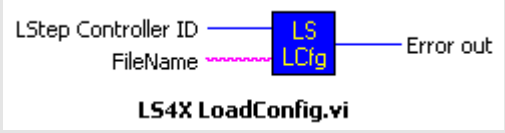
Delphi:	function LS_SendStringPosCmd(Str, Ret: PAnsiChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer;
	function LS_SendStringPosCmdW(Str, Ret: PWideChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer;
	function LSX_SendStringPosCmd(LSID: Integer; Str, Ret: PAnsiChar; 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 [ms]	
Example:	LS.SendStringPosCmd("!moa 1 2\r", pcLStepVer, 256, true, 100000);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

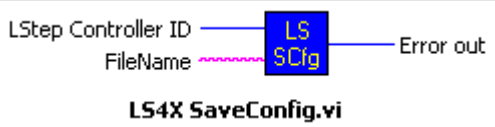
LS_LoadConfig, LS_LoadConfigW		
Description:	Loads LSTEP configuration (interface, axis settings, controllers) from an INI file. The loaded configuration is used in the functions LS_Connect and LS_SetControlPars. The format of the INI file for LSTEP controllers is compatible with the WIN-CommanderINI file, i.e. the settings can be taken over from the - WINCommander (Wincom4.ini). This does not apply to WIN-Commander 5 and the controllers of the LSTEPexpress series **Attention!** For LSTEP-PCIexpress or LSTEPexpress, you can save the settings with WIN-Commander 5 in the Controller/Export configuration menu, and can load them with LoadConfig. LS_SetControlPars is used to transmit this configuration to the controller.	

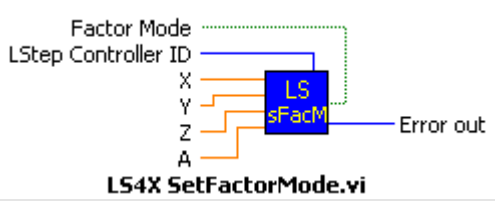
Delphi:	function LS_LoadConfig(FileName: PAnsiChar): Integer;
	function LS_LoadConfigW(FileName: PWideChar): Integer;
	function LSX_LoadConfig(LSID: Integer; FileName: PAnsiChar): Integer;
	function LSX_LoadConfigW(LSID: Integer; FileName: PWideChar): Integer;	

C++:	int LoadConfig (char *pcFileName); int LoadConfigW (TCHAR *pcFileName);	
LabView:		
Parameter:	FileName	File name of INI file as zero-terminated string
Example:	LS.LoadConfig("C:\LStepTest\LStep.INI");	
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

LS_SaveConfig, LS_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. (not for LSTEPexpress)		

Delphi:	function LS_SaveConfig(FileName: PAnsiChar): Integer;
	function LS_SaveConfigW(FileName: PWideChar): Integer;
	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:	LS.SaveConfig("C:\LStepTest\LStep.INI");		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	-	-

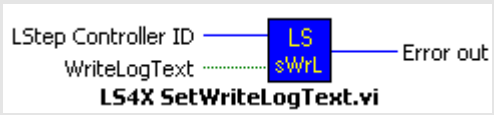
LS_SetFactorMode	
Description:	Position value conversion for 'odd' spindle pitches **SetPitch can only be called with the actual, physical spindle pitch after SetFactorMode.** All moving commands use a factor for conversion after calling SetFactorMode and SetPitch, so that the LSTEP is positioned correctly. The resulting vector is as follows: Sent position vector = sent position vector * spindle pitch LSTEP / physical spindle pitch
Delphi:	function LS_SetFactorMode(AFactorMode: LongBool; X, Y, Z, A: Double): Integer; function LSX_SetFactorMode(LSID: Integer; AFactorMode: LongBool; X, Y, Z, A: Double): Integer;
C++:	int SetFactorMode (BOOL bAFactorMode, double dX, double dY, double dZ, double dA);
LabView:	

Parameter:	AFactorMode	This parameter activates an API-internal conversion of the position values/spindle pitch in order to avoid rounding errors with 'odd' spindle pitches.	
	X, Y, Z, A	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)	
Example:	LS.SetFactorMode(true, 1, 1, 1, 0); LS.SetPitch(1.234, 1.234, 2.345, 0); LS.MoveAbs(1.234, 2.468, 2.345, 0, true);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	-	-

LS_Initialize, LS_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	LS.Initialize("C:\Users\All Users\MyProg\LS1");		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

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

LS_SetShowProt			
Description:	Interface log On/Off		
Delphi:	function LS_SetShowProt(ShowProt: LongBool): Integer; function LSX_SetShowProt(LSID: Integer; ShowProt: LongBool): Integer;		
C++:	int SetShowProt (BOOL ShowProt);		
LabView:			
Parameter:	ShowProt	Specifies whether or not the window "Interface Protocol" is to be shown	
Example:	LS.SetShowProt(true);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

LS_SetWriteLogText	
Description:	Writing of log file LSTEP4.log On/Off (writing is deactivated in LSTEP4.log by default)
Delphi:	function LS_SetWriteLogText(AWriteLogText: LongBool): Integer; function LSX_SetWriteLogText(LSID: Integer; AWriteLogText: LongBool): Integer;
C++:	int SetWriteLogText (BOOL AWriteLogText);
LabView:	

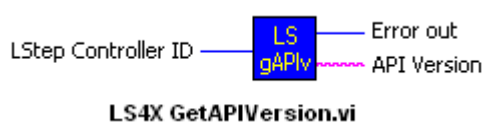
Parameter:	-		
Example:	LS.SetWriteLogText (true);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

LS_SetWriteLogTextFN, LS_SetWriteLogTextFNW			
Description:	Activation/deactivation of writing the interface log in a specific file (writing is deactivated by default)		

Delphi:	function LS_SetWriteLogTextFN(AWriteLogText: LongBool; ALogFN: PAnsiChar): Integer;
	function LS_SetWriteLogTextFNW(AWriteLogText: LongBool; ALogFN: PWideChar): Integer;
	function LSX_SetWriteLogTextFN(LSID: Integer; AWriteLogText: LongBool; ALogFN: PAnsiChar): Integer;
	function LSX_SetWriteLogTextFNW(LSID: Integer; AWriteLogText: LongBool; ALogFN: PWideChar): Integer;		

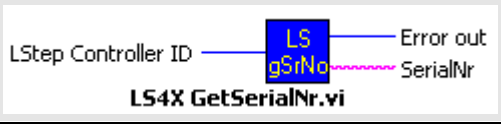
C++:	int SetWriteLogTextFN (BOOL bAWriteLogText, char *pcALogFN); int SetWriteLogTextFNW (BOOL bAWriteLogText, TCHAR *pcALogFN);		
LabView:			
Parameter:	AWriteLogText		If True, then write log file
	ALogFN		File name of log file
Example:	LS.SetWriteLogTextFN(true, „C:\Temp\prot.txt");		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

4.2.2 Control and API information

LS_GetAPIVersion, LS_GetAPIVersionW			
Description:	Shows version number of LSTEP-API		
Delphi:	LS_GetAPIVersion(APIVer: PAnsiChar; MaxLen: Integer): Integer; LS_GetAPIVersionW(APIVer: PWideChar; MaxLen: Integer): Integer; 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:	LS.GetAPIVersion(pcVers, 100);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	-	-

LS_GetSerialNr, LS_GetSerialNrW	
Description:	Reads serial number of controller. (not for LSTEPexpress)

Delphi:	function LS_GetSerialNr(SerialNr: PAnsiChar; MaxLen: Integer): Integer;
	function LS_GetSerialNrW(SerialNr: PWideChar; MaxLen: Integer): Integer;
	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:	LS.GetSerialNr(pcSerialNr, 256);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?readsn	-

LS_GetVersionStr, LS_GetVersionStrW			
Description:	Returns the current firmware version number. For additional information, GetVersionStrDet and GetVersionStrInfo should also be used.		

Delphi:	function LS_GetVersionStr(Vers: PAnsiChar; MaxLen: Integer): Integer;
	function LS_GetVersionStrW(Vers: PWideChar; MaxLen: Integer): Integer;
	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:	LS.GetVersionStr(pcVers, 64);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?ver	-

LS_GetVersionStrDet, LS_GetVersionStrDetW			
Description:	Function for reading the firmware configuration. For additional information, GetVersionStr and GetVersionStrInfo should also be used. (not for LSTEPexpress)		

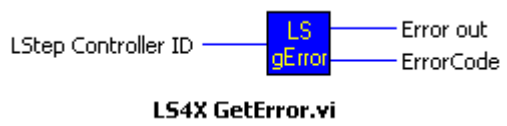
Delphi:	function LS_GetVersionStrDet(VersDet: PAnsiChar; MaxLen: Integer): Integer;
	function LS_GetVersionStrDetW(VersDet: PWideChar; MaxLen: Integer): Integer;
	function LSX_GetVersionStrDet(LSID: Integer; VersDet: PAnsiChar; MaxLen: Integer): Integer;
	function LSX_GetVersionStrDetW(LSID: Integer; 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:	LS.GetVersionStrDet(pcVersDet, 64);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?det	-

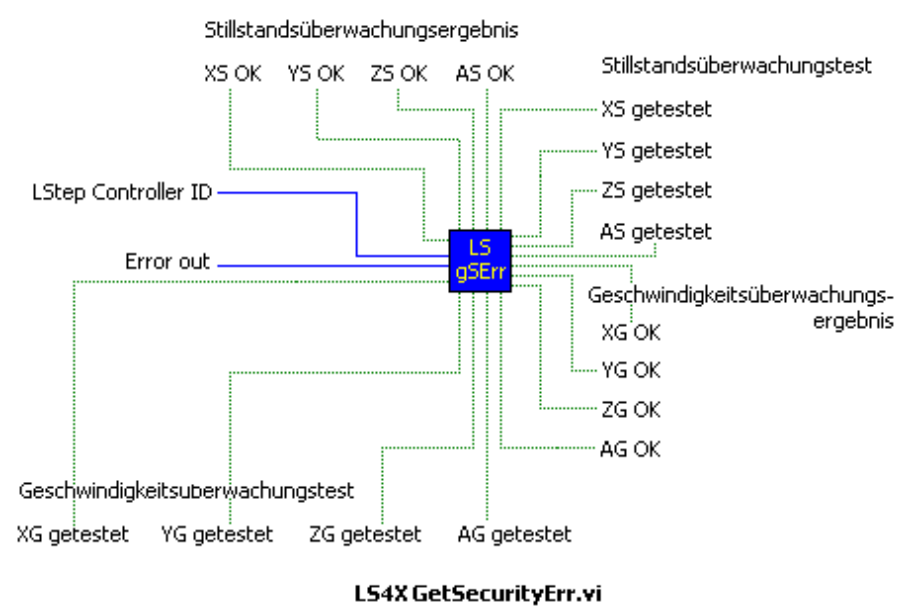
LS_GetVersionStrInfo, LS_GetVersionStrInfoW			
Description:	Provides detailed information about version number. For additional information, GetVersionStr and GetVersionStrDet should also be used.		
Delphi:	function LS_ GetVersionStrInfo (VersInfo: PAnsiChar; MaxLen: Integer): Integer; function LS_ GetVersionStrInfoW (VersInfo: PWideChar; MaxLen: Integer): Integer; 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:	LS.GetVersionStrInfo (pcVersInfo, 64);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?iver	-

4.2.3 Status requests

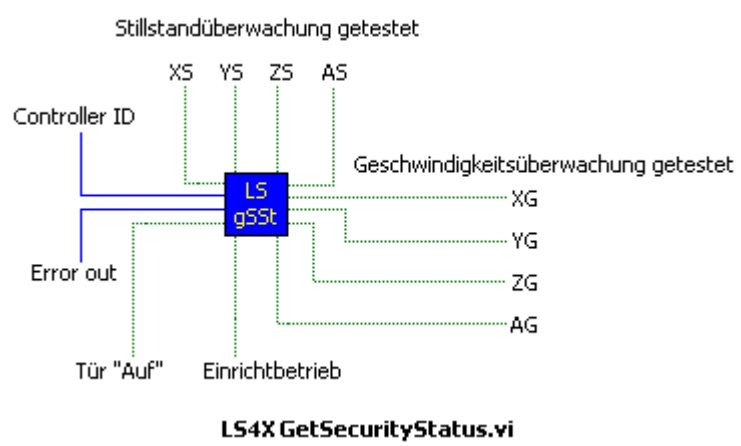
LS_GetError			
Description:	This function requests the current error number from the controller.		
Delphi:	function LS_GetError(var ErrorCode: Integer): Integer; function LSX_GetError(LSID: Integer; var ErrorCode: Integer): Integer;		
C++:	int GetError(int *plErrorCode);		
LabView:			
Parameter:	ErrorCode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

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

<table>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>Value</td><td colspan="2">32-Bit LongWord without sign; contains the bit mask in the bits 0-15 after activating the function.

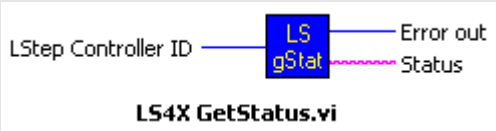
Value 0 for monitoring result = not OK
Value 1 for monitoring result = OK
Value 0 for monitoring test = not tested
Value 1 for monitoring test = tested

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="3">LS.GetSecurityErr(&Value);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEPexpress series)</td></tr>
<tr><td>1</td><td>?securityerror</td><td>-</td></tr>
</table>

<table>
<tr><th colspan="4">LS_GetSecurityStatus</th></tr>
<tr><td>Description:</td><td colspan="3">Shows the current status of safety monitoring
(only with LS44 controllers).</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LS_GetSecurityStatus(var Value: LongWord): Integer;
function LSX_GetSecurityStatus(LSID: Integer; var Value: LongWord): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int GetSecurityStatus(LongWord *pValue);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>Value</td><td colspan="2">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="3">LS.GetSecurityStatus(&Value);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEPexpress series)</td></tr>
<tr><td>1</td><td>?securitystatus</td><td>-</td></tr>
</table>

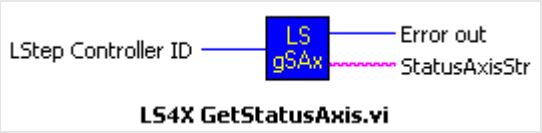
LS_GetStatus, LS_GetStatusW			
Description:	Shows the current controller status.		

Delphi:	function LS_GetStatus(Stat: PAnsiChar; MaxLen: Integer): Integer;
	function LS_GetStatusW(Stat: PWideChar; MaxLen: Integer): Integer;
	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:	LS.GetStatus(pcStat, 256);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?status	-

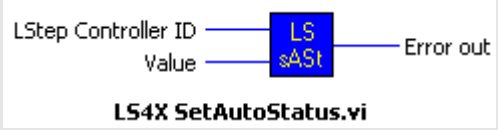
LS_GetStatusAxis, LS_GetStatusAxisW			
Description:	Shows the present status of the individual axes		

Delphi:	function LS_GetStatusAxis(StatusAxisStr: PAnsiChar; MaxLen: Integer): Integer;
	function LS_GetStatusAxisW(StatusAxisStr: PWideChar; MaxLen: Integer): Integer;
	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:	LS.GetStatusAxis(pcStatAxis, 256);	

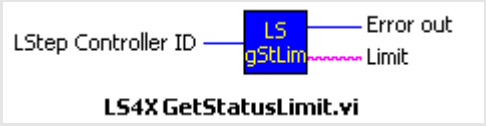
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?statusaxis	-

LS_SetAutoStatus	
Description:	This function activates or deactivates the auto status. **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.
Delphi:	function LS_SetAutoStatus(Value: Integer): Integer; function LSX_SetAutoStatus(LSID: Integer; Value: Integer): Integer;
C++:	int SetAutoStatus(int lValue);
LabView:	

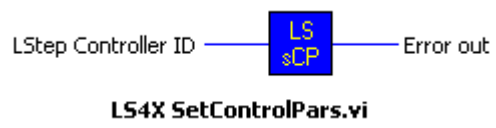
Parameter:	Value	AutoStatus-Modus 0 = No status is transmitted by the controller. 1 = "Position reached" signals are sent automatically by the controller. 2 = "Position reached" signals and status messages are sent automatically by the controller. * 3 = Only a Carriage Return is fed back for "Position reached". * (* only LSTEP 2000 series)	
Example:	LS.SetAutoStatus(3);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3 (see Value)	!autostatus	-

LS_GetStatusLimit, LS_GetStatusLimitW			
Description:	Shows the current state of the software limits of each axis.		

Delphi:	function LS_GetStatusLimit(Limit: PAnsiChar; MaxLen: Integer): Integer;
	function LS_GetStatusLimitW(Limit: PWideChar; MaxLen: Integer): Integer;
	function LSX_GetStatusLimit(LSID: Integer; Limit: PAnsiChar; MaxLen: Integer): Integer;
	function LSX_GetStatusLimitW(LSID: Integer; Limit: PWideChar; MaxLen: Integer): Integer;		

C++:	int GetStatusLimit(char *pcLimit, int lMaxLen); int GetStatusLimitW (TCHAR *pcLimit, int lMaxLen);		
LabView:			
Parameter:	pc Limit	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	
	MaxLen	Maximum number of characters which may be copied into the buffer	
Example:	LS.GetStatusLimit(pc Limit, 64);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?statuslimit	-

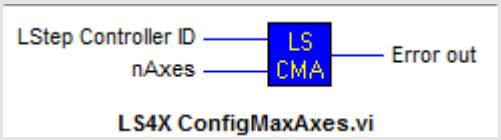
4.2.4 Parameter handling

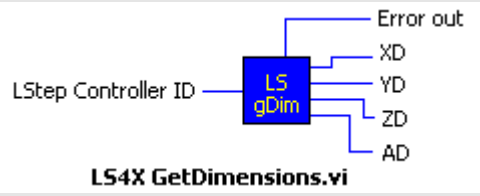
LS_SetControlPars			
Description:	Transmits the parameters loaded by LS_LoadConfig to the LSTEP.		
Delphi:	function LS_SetControlPars: Integer; function LSX_SetControlPars(LSID: Integer): Integer;		
C++:	int SetControlPars ();		
LabView:			
Parameter:	-		
Example:	LS.SetControlPars();		
Other:	Compatibility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

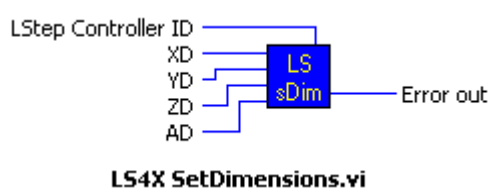
LS_LStepSave			
Description	This function triggers the saving of the current configuration in the non-volatile storage of the LSTEP controller. The controller feedback showing that saving is completed is awaited.		
Delphi	function LS_LStepSave(): Integer; function LSX_LStepSave(LSID: Integer): Integer;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

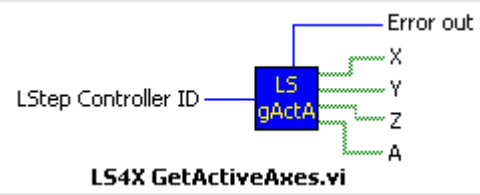
<table>
<tr><th colspan="4">LS_SoftwareReset</th></tr>
<tr><td>Description:</td><td colspan="3">The controller is restarted. The controller loads the data saved last.</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LS_SoftwareReset: Integer;
function LSX_SoftwareReset(LSID: Integer): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int SoftwareReset();</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td colspan="3">-</td></tr>
<tr><td>Example:</td><td colspan="3">LS.SoftwareReset();</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEPexpress series)</td></tr>
<tr><td>3</td><td>!reset</td><td>-</td></tr>
</table>

4.2.5 Axis and motor configuration

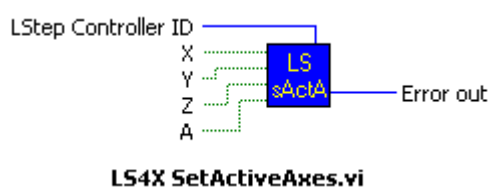
LS_ConfigMaxAxes			
Description:	Configures the number of axes.		
Delphi:	function LS_ConfigMaxAxes(nAxes: Integer): Integer; function LSX_ConfigMaxAxes(LSID: Integer; nAxes: Integer): Integer;		
C++:	int ConfigMaxAxes(int nAxes);		
LabView:			
Parameter:	nAxes	Number of axes	
Example:	LS.ConfigMaxAxes(2);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

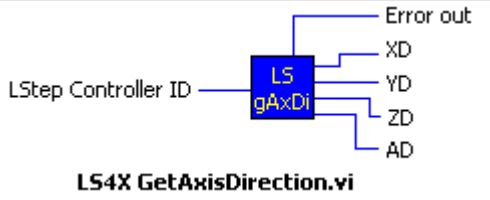
LS_GetDimensions			
Description:	Inquiry of set axes dimensions.		
Delphi:	function LS_GetDimensions(var XD, YD, ZD, AD: Integer): Integer; function LSX_GetDimensions(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;		
C++:	int GetDimensions(int *plXD, int *plYD, int *plZD, int *plAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Dimension values 0 = Microsteps 1 = µm 2 = Millimetres 3 = Degrees 4 = Revolutions	
Example:	LS.GetDimensions(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?dim	-

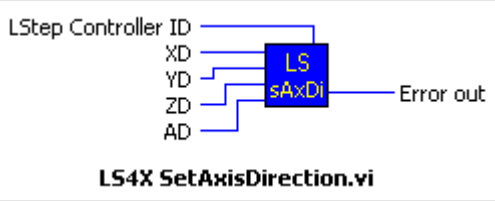
LS_SetDimensions			
Description:	Sets axes dimensions.		
Delphi:	function LS_SetDimensions(XD, YD, ZD, AD: Integer): Integer; function LSX_SetDimensions(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;		
C++:	int SetDimensions(int lXD,int lYD,int lZD,int lAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Dimension values 0 = Microsteps 1 = µm 2 = Millimetres 3 = Degrees 4 = Revolutions	
Example:	LS.SetDimensions(3, 2, 2); // X-axis in degrees; Y and Z in mm		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!dim	-

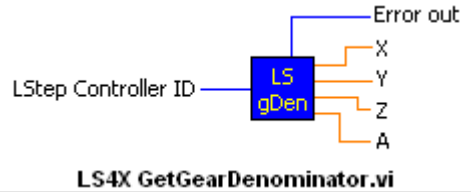
LS_GetActiveAxes	
Description:	Shows the released axes.
Delphi:	function LS_GetActiveAxes(var Flags: Integer): Integer; function LSX_GetActiveAxes(LSID: Integer; var Flags: Integer): Integer;
C++:	int GetActiveAxes(int *plFlags);
LabView:	

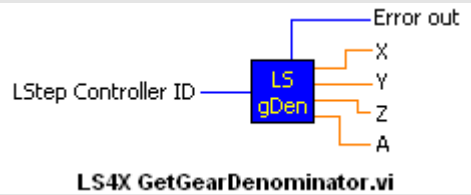
Parameter:	Flags	32-bit integer containing a bit mask after calling the function in the bits 0-4. Bit 0 = X-axis Bit 1 = Y-axis … Value 0 = Axis deactivated Value 1 = Axis activated	
Example:	LS.GetActiveAxes(&Flags);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?axis	-

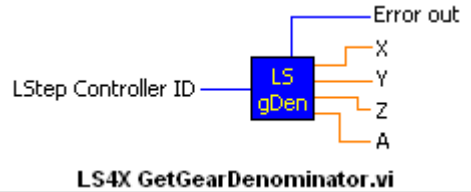
LS_SetActiveAxes			
Description:	Sets axes release.		
Delphi:	function LS_SetActiveAxes(Flags: Integer): Integer; function LSX_SetActiveAxes(LSID: Integer; Flags: Integer): Integer;		
C++:	int SetActiveAxes(int Flags);		
LabView:			
Parameter:	Flags	Bit mask for axis release Bit 0 = X-axis Bit 1 = Y-axis … Value 0 = Axis deactivated Value 1 = Axis activated	
Example:	LS.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 (LSTEPexpress series)
	3	!axis	-

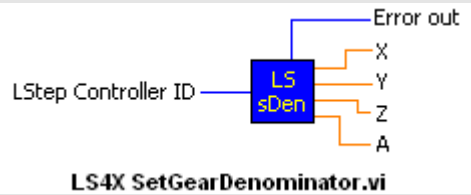
LS_GetAxisDirection			
Description	Function for change of direction inquiry.		
Delphi	function LS_GetAxisDirection(var XD, YD, ZD, AD: Integer): Integer; function LSX_GetAxisDirection(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;		
C++	int GetAxisDirection(int *plXD, int *plYD, int *plZD, int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

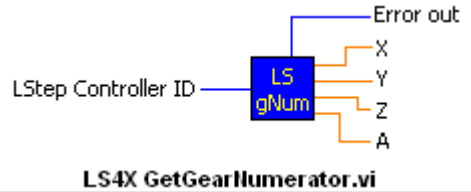
LS_SetAxisDirection			
Description	Function for setting the change of direction.		
Delphi	function LS_SetAxisDirection(XD, YD, ZD, AD: Integer): Integer; function LSX_SetAxisDirection(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;		
C++	int SetAxisDirection (int lXD, int lYD, int lZD, int lAD);		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

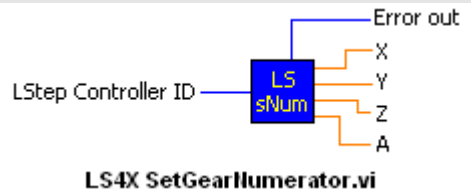
LS_GetGear			
Description:	Function for gear ratio inquiry. (not for LSTEPexpress)		
Delphi:	function LS_GetGear(var X, Y, Z, A: Double): Integer; function LSX_GetGear(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetGear (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView: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

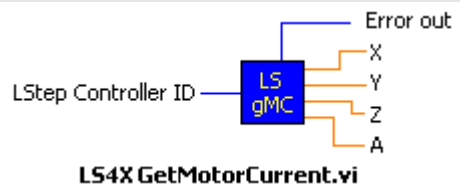
LS_SetGear			
Description:	Function for gear ratio setting. (not for LSTEPexpress)		
Delphi:	function LS_SetGear(var X, Y, Z, A: Double): Integer; function LSX_SetGear(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int SetGear (double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView: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

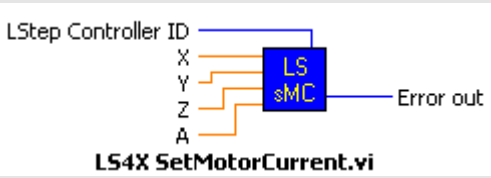
LS_GetGearDenominator			
Description:	Inquires denominator of gear ratio (only for LSTEPexpress).		
Delphi:	function LS_GetGearDenominator(var X, Y, Z, A: Integer): Integer; function LSX_GetGearDenominator(LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetGearDenominator(int *plX, int *plY,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

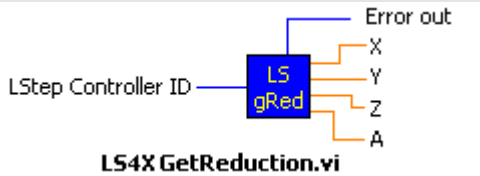
LS_SetGearDenominator			
Description:	Adjust denominator of gear ratio. (only for LSTEPexpress).		
Delphi:	function LS_SetGearDenominator(X, Y, Z, A: Integer): Integer; function LSX_SetGearDenominator(LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetGearDenominator(int lX, int lY, int lZ,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

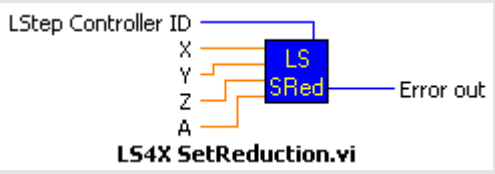
LS_GetGearNumerator			
Description:	Inquiry of gear ratio numerator (only for LSTEPexpress).		
Delphi:	function LS_GetGearNumerator(var X, Y, Z, A: Integer): Integer; function LSX_GetGearNumerator(LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetGearNumerator(int *plX, int *plY, int *plZ, int *plA);		
LabView:			
Parameter:	X, Y, Z, A	Gear ratio numerator (actuator side).	
Example:	LS.GetGearNumerator(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	?gearnumerator	-

LS_SetGearNumerator			
Description	Adjust numerator of gear ratio (only for LSTEPexpress).		
Delphi	function LS_SetGearNumerator(X, Y, Z, A: Integer): Integer; function LSX_SetGearNumerator(LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++	int SetGearNumerator(int lX, int lY, int lZ, int lA);		
LabView:			
Parameter	X, Y, Z, A	Value of gear ratio numerator (motor side)	
Example	LS.SetGearNumerator(4, 1, 1, 1);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	!gearnumerator	validconfig

LS_GetMotorCurrent			
Description:	Function for inquiring the set motor current.		
Delphi:	function LS_GetMotorCurrent(var X, Y, Z, A: Double): Integer; function LSX_GetMotorCurrent(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetMotorCurrent(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	X, Y, Z, A	Motor current in A	
Example:	LS.GetMotorCurrent(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	LSTEP 2000: ?cur LSTEPexpress: ?motorcurrent	-

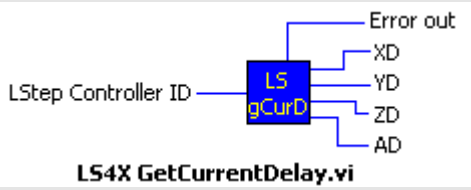
LS_SetMotorCurrent			
Description:	Function for setting the motor current.		
Delphi:	function LS_SetMotorCurrent(X, Y, Z, A: Double): Integer; function LSX_SetMotorCurrent(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetMotorCurrent (double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	X, Y, Z, A	Motor current in A	
Example:	LS.SetMotorCurrent(1.5, 1.5, 1.0, 1.0); // Motor current for X and Y is 1.5 amperes; for Z and A: 1.0 amperes		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	LSTEP2000: !cur LSTEPexpress: !motorcurrent	validpar / validconfig

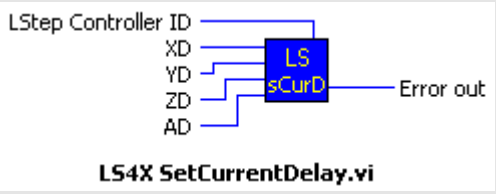
LS_GetReduction			
Description:	Inquiry of set current reduction.		
Delphi:	function LS_GetReduction(var X, Y, Z, R: Double): Integer; function LSX_GetReduction(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetReduction(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			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

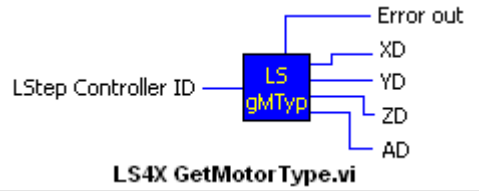
LS_SetReduction			
Description:	Setting the ratio by which the motor rated current is reduced if current reduction is active.		
Delphi:	function LS_SetReduction(X, Y, Z, A: Double): Integer; function LSX_SetReduction(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetReduction(double dX, double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

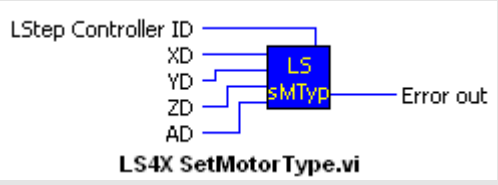
Example:	LS.SetReduction(0.1, 0.7, 0.5, 0.5); //LSTEP 2000 series
	/* X-axis bias current = <0.1*rated current; Y-axis = 0.7*rated current... */		

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

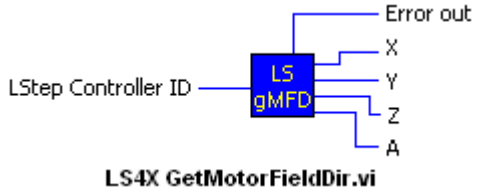
LS_GetCurrentDelay			
Description:	Indicates the time delay until the current reduction is activated.		
Delphi:	function LS_GetCurrentDelay(var X, Y, Z, A: Integer): Integer; function LSX_GetCurrentDelay(LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetCurrentDelay(int *plX, int *plY,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

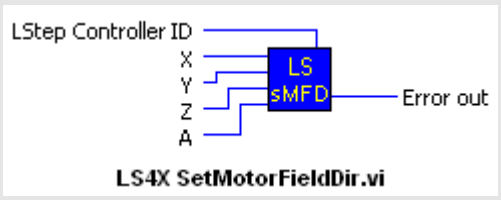
LS_SetCurrentDelay			
Description:	Indicates the time delay for activating the current reduction.		
Delphi:	function LS_SetCurrentDelay(X, Y, Z, A: Integer): Integer; function LSX_SetCurrentDelay(LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetCurrentDelay(int lX, int lY, int lZ, int lA);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

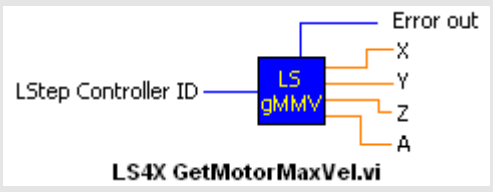
LS_GetMotorType			
Description:	Function for reading the set motor type (only for LSTEPexpress).		
Delphi:	function LS_GetMotorType(var X: Integer; var Y: Integer; var Z: Integer; var A: Integer): Integer; function LSX_GetMotorType(LSID: Integer; var X: Integer; var Y: Integer; var Z: Integer; var A: 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:	LS.GetMotorType(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	?motortype	-

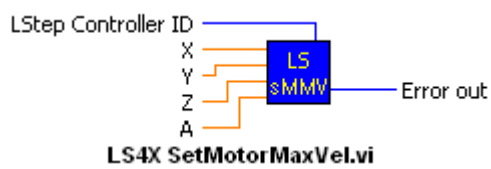
LS_SetMotorType			
Description:	Function for setting the motor type (only for LSTEPexpress).		
Delphi:	function LS_SetMotorType(X: Integer; var Y: Integer; var Z: Integer; var A: Integer): Integer; function LSX_SetMotorType(LSID: Integer; X: Integer; var Y: Integer; var Z: Integer; var A: Integer): Integer;		
C++:	int SetMotorType(int lX, int lY, int lZ, int lA);		
LabView:			

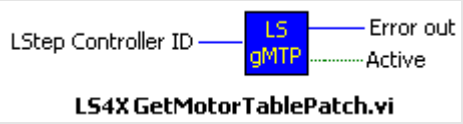
Parameter:	X, Y, Z, A	Motor type 0 = rotary 2-phase stepper motor 1 = linear 2-phase stepper motor 2 = rotary 3-phase stepper motor 3 = linear 3-phase stepper motor 4 = rotary 2-phase servomotor 5 = linear 2-phase servomotor 6 = rotary 3-phase servomotor 7 = linear 3-phase servomotor	
Example:	LS.SetMotorType(5, 4, 4, 4);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	!motortype	validconfig

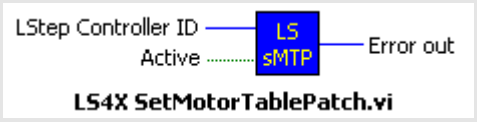
LS_GetMotorFieldDir			
Description:	Function for inquiring the field direction of the motor (only for LSTEPexpress).		
Delphi:	function LS_GetMotorFieldDir(var X, Y, Z, A: Integer): Integer; function LSX_GetMotorFieldDir(LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetMotorFieldDir(int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

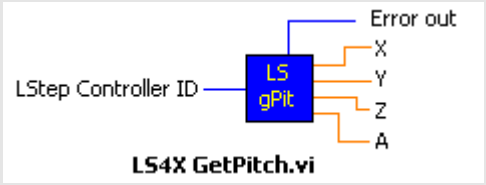
LS_SetMotorFieldDir			
Description:	Function for setting the field direction of the motor (only for LSTEPexpress).		
Delphi:	function LS_SetMotorFieldDir(X, Y, Z, A: Integer): Integer; function LSX_SetMotorFieldDir(LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetMotorFieldDir(int lX, int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

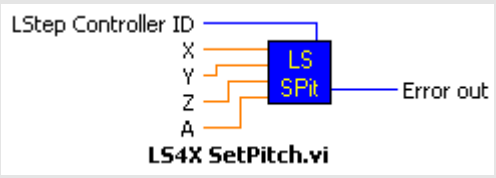
LS_GetMotorMaxVel			
Description:	Reading of maximum motor speed or velocity (only for LSTEPexpress).		
Delphi:	function LS_GetMotorMaxVel(var XD, YD, ZD, AD: Double): Integer; function LSX_GetMotorMaxVel(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;		
C++:	int GetMotorMaxVel(double *pdXD, double *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:	LS.GetMotorMaxVel(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	?motormaxvel	-

LS_SetMotorMaxVel			
Description:	Setting of maximum motor speed or velocity (only for LSTEPexpress).		
Delphi:	function LS_SetMotorMaxVel(XD, YD, ZD, AD: Double): Integer; 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:	LS.GetMotorMaxVel(1000, 1000, 500, 250);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	!motormaxvel	validconfig

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

LS_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 LSTEPexpress).		
Delphi:	function LS_SetMotorTablePatch(bActive: LongBool): Integer; 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:	LS.SetMotorTablePatch(True);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!motorpatch	-

LS_GetPitch			
Description:	Shows the set value of the spindle pitch.		
Delphi:	function LS_GetPitch(var X, Y, Z, A: Double): Integer; function LSX_GetPitch(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetPitch(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	X, Y, Z, A	Spindle pitches in mm/revolution	
Example:	LS.GetPitch(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?pitch	-

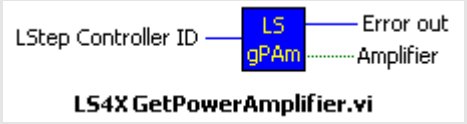
LS_SetPitch			
Description:	Sets the axis spindle pitch.		
Delphi:	function LS_SetPitch(X, Y, Z, R: Double): Integer; function LSX_SetPitch(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetPitch(double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	X, Y, Z, A	Spindle pitches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

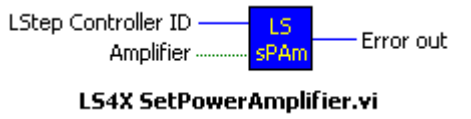
LS_GetXYAxisComp			
Description	Inquiry of XY-axis overlay (not for LSTEPexpress).		
Delphi	function LS_GetXYAxisComp(var Value: Integer): Integer; function LSX_GetXYAxisComp(LSID: Integer; var Value: Integer): Integer;		
C++	int GetXYAxisComp (int *plValue);		
LabView:			
Parameter	Value	Mode of axis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

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

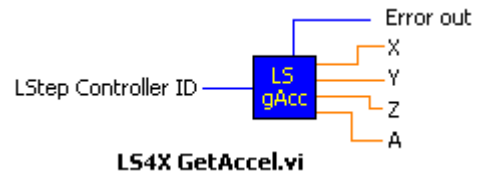
LS_GetStopPolarity			
Description:	Reading of stop input polarity.		
Delphi:	function LS_GetStopPolarity(var bHighActiv: LongBool): Integer; function LSX_GetStopPolarity(LSID: Integer; var bHighActiv: LongBool): Integer;		
C++:	int GetStopPolarity(BOOL *pbHighActiv);		
LabView:			
Parameter:	bHighActiv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

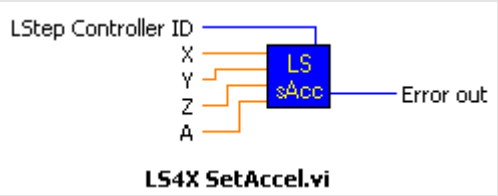
LS_SetStopPolarity			
Description:	Function for setting the stop input polarity.		
Delphi:	function LS_SetStopPolarity(bHighActiv: LongBool): Integer; function LSX_SetStopPolarity(LSID: Integer; bHighActiv: LongBool): Integer;		
C++:	int SetStopPolarity(BOOL bHighActiv);		
LabView:			
Parameter:	bHighActiv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

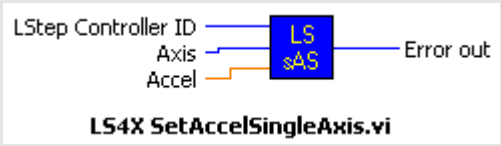
LS_GetPowerAmplifier			
Description:	Indicates whether the power amplifiers of the LS44 are activated or deactivated (only for LS44 controller and LSTEPexpress).		
Delphi:	function LS_GetPowerAmplifier(bAmplifier: var LongBool): Integer; function LSX_GetPowerAmplifier(LSID: Integer; var bAmplifier: LongBool): 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:	LS.GetPowerAmplifier(&Amplifier);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3 (see description)	?pa	-

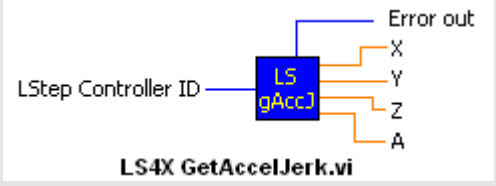
LS_SetPowerAmplifier			
Description:	Activates or deactivates the power amplifiers of the controller (only for LS44 controller and LSTEPexpress).		
Delphi:	function LS_SetPowerAmplifier (bAmplifier: LongBool): Integer; function LSX_SetPowerAmplifier (LSID: Integer; bAmplifier: LongBool): Integer;		
C++: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

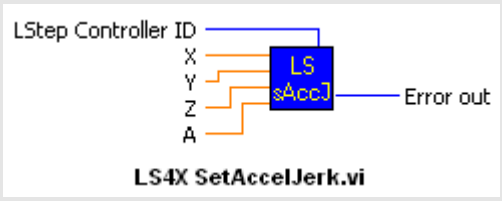
4.2.6 Kinematics

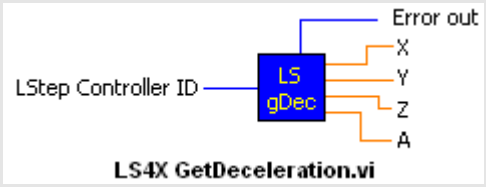
LS_GetAccel			
Description:	Function for inquiring the acceleration for positioning processes.		
Delphi:	function LS_GetAccel(var X, Y, Z, A: Double): Integer; function LSX_GetAccel(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetAccel(double *pdX, double *pdY, double *pdZ, double *pdA);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

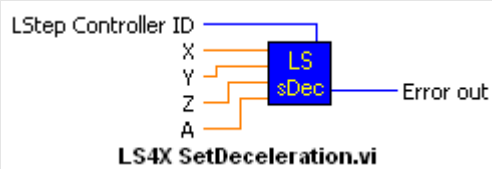
LS_SetAccel			
Description:	Function for setting the acceleration for positioning processes.		
Delphi:	function LS_SetAccel(X, Y, Z, A: Double): Integer; function LSX_SetAccel(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetAccel(double dX, double dY, double dZ, double dA);		
LabView:			
Parameter: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

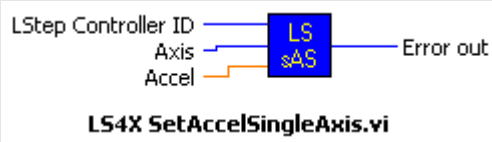
LS_SetAccelSingleAxis			
Description:	Function for setting the acceleration of a single axis for positioning processes.		
Delphi:	function LS_SetAccelSingleAxis(Axis: Integer; Accel: Double): Integer; function LSX_SetAccelSingleAxis(LSID: Integer; Axis: Integer; Accel: Double): Integer;		
C++:	int SetAccelSingleAxis(int lAxis,double dAccel);		
LabView:			
Parameter:	Axis	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 LSTEPexpress series = set dimension/s², maximum value limited by controller	
Example:	LS.SetAccelSingleAxis(4, 1.0); // Accelerate A-axis 1.0 m/s²		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!accel	-

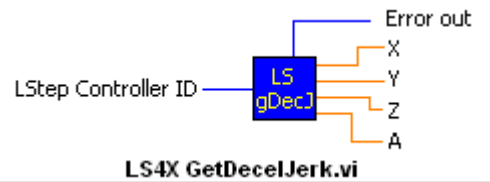
LS_GetAccelJerk			
Description:	Function for inquiring the jerk used during the acceleration phase of positioning processes (only for LSTEPexpress).		
Delphi:	function LS_GetAccelJerk(var XD, YD, ZD, AD: Double): Integer; function LSX_GetAccelJerk(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;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

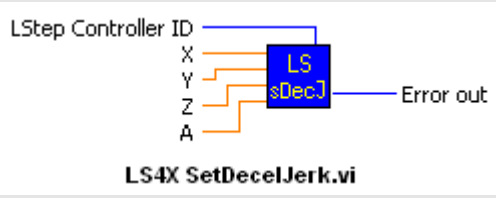
LS_SetAccelJerk			
Description:	Function for setting the jerk used during the acceleration phase of positioning processes (only for LSTEPexpress).		
Delphi:	function LS_SetAccelJerk(XD, YD, ZD, AD: Double): Integer; function LSX_SetAccelJerk(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int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

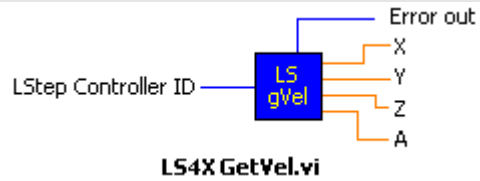
LS_GetDeceleration			
Description:	Inquiry of deceleration applied for movements (only for LSTEPexpress).		
Delphi:	function LS_GetDeceleration(var XD, YD, ZD, AD: Double): Integer; function LSX_GetDeceleration(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetDeceleration(double *pdXD,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

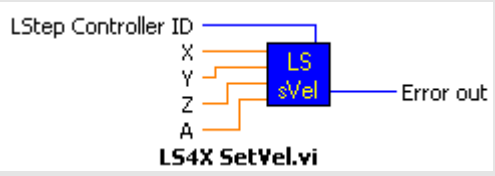
LS_SetDeceleration			
Description:	Setting of deceleration to be applied for movements (only for LSTEPexpress).		
Delphi:	function LS_SetDeceleration(XD, YD, ZD, AD: Double): Integer; function LSX_SetDeceleration(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetDeceleration(double dXD,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

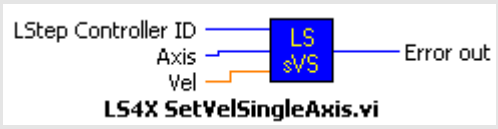
LS_SetDecelSingleAxis			
Description:	Sets the deceleration applied for the movement of a single axis (only for LSTEPexpress).		
Delphi:	function LS_SetDecelSingleAxis(Axis: Integer; Decel: Double): Integer; function LSX_SetDecelSingleAxis(LSID: Integer; Axis: Integer; Decel: Double): Integer;		
C++:	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:	LS.SetDecelSingleAxis(1, 1000.0);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	!decel	-

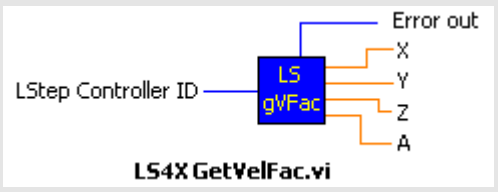
LS_GetDecelJerk			
Description:	Inquiry of the jerk to be used during the deceleration phase of a movement (only for LSTEPexpress).		
Delphi:	function LS_GetDecelJerk(var XD, YD, ZD, AD: Double): Integer; function LSX_GetDecelJerk(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;		
C++:	int GetDecelJerk(double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

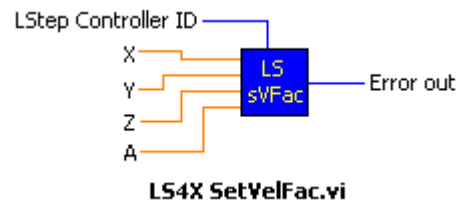
LS_SetDecelJerk			
Description:	Setting of the jerk to be used during the deceleration phase of a movement (only for LSTEPexpress).		
Delphi:	function LS_SetDecelJerk(XD, YD, ZD, AD: Double): Integer; function LSX_SetDecelJerk(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetDecellJerk(double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

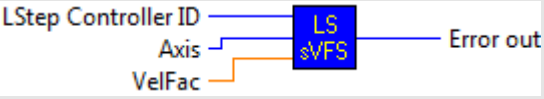
LS_GetVel			
Description:	Inquiry of the set velocity used for positioning processes.		
Delphi:	function LS_GetVel(var X, Y, Z, A: Double): Integer; function LSX_GetVel(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetVel(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:	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

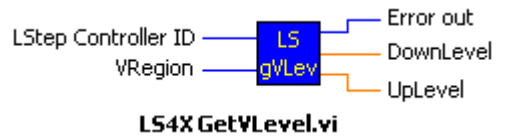
LS_SetVel			
Description:	Setting of velocity used for positioning processes.		
Delphi:	function LS_SetVel(X, Y, Z, R: Double): Integer; function LSX_SetVel(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetVel(double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	X, Y, Z, A: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

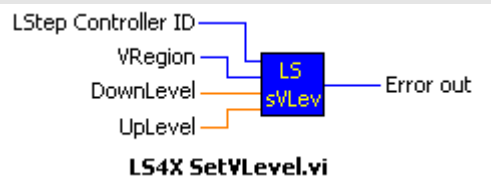
LS_SetVelSingleAxis			
Description:	Function for setting the velocity for a single axis		
Delphi:	function LS_SetVelSingleAxis(Axis: Integer; Vel: Double): Integer; function LSX_SetVelSingleAxis(LSID: Integer; Axis: Integer; Vel: Double): Integer;		
C++:	int SetVelSingleAxis(int lAxis,double dVel);	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

LS_GetVelFac			
Description:	Function for inquiring the velocity reduction (not for LSTEPexpress).		
Delphi:	function LS_GetVelFac(var X, Y, Z, A: Double): Integer; 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:	LS.GetVelFac(X, Y, Z, A);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?velfac	-

LS_SetVelFac			
Description:	Function for setting the velocity reduction (not for LSTEPexpress).		
Delphi:	function LS_SetVelFac(X, Y, Z, A: Double): Integer; 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:	LS.SetVelFac(1, 1, 0.1, 0.1);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!velfac	-

LS_SetVelFacSingleAxis			
Description:	Function for setting the velocity reduction of a single axis (not for LSTEPexpress).		
Delphi:	function LS_SetVelFacSingleAxis(Axis: Integer; Value: Double): Integer; function LSX_SetVelFacSingleAxis(LSID: Integer; Axis: Integer; Value: Double): Integer;		
C++:	int SetVelFacSingleAxis(int lAxis, double dValue);		
LabView:			
Parameter:	Axis	Axis whose velocity reduction is to be set X = 1 Y = 2 …	
	VelFac	Factor to be set for velocity reduction. The resulting velocity is v=Vel*VelFac.	
Example:	LS.SetVelFacSingleAxis(1, 0.1)		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!velfac	-

LS_GetVLevel			
Description:	Delivers the speed limits of the indicated speed range (not for LSTEPexpress).		
Delphi:	function LS_GetVLevel(lVRegion: Integer; var dDownLevel, dUpLevel: Double): Integer; 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:	LS.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 (LSTEPexpress series)
	1	?vlevel	-

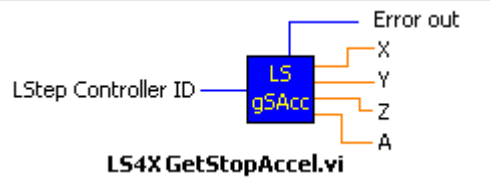
LS_SetVLevel		
Description:	Exclude speed ranges in which the system shows resonances (not for LSTEPexpress).	
Delphi:	function LS_SetVLevel(lVRegion: Integer; dDownLevel, dUppLevel: Double): Integer; function LSX_SetVLevel(LSID: Integer; lVRegion: Integer; dDownLevel, dUppLevel: Double): Integer;	
C++:	int SetVLevel(int lVRegion, double dDownLevel, double dUppLevel);	
LabView:		

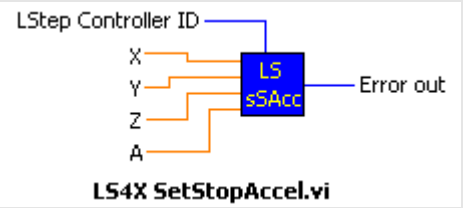
Parameter:	lVRegion	Value range 1-4. 1 – First/lowest speed range 2 – Second/middle speed range 3 – Third/highest speed range 4 – Up to this speed limit, the correction table is used.	
	dDownLevel	Lower limit of range (for lVRegion = 4 speed limit) in rps	
	dUppLevel	Upper limit of range (for lVRegion = 4 has no meaning) in rps	
Example:	LS.SetVLevel(4, 10.0, 0.0); //The correction table is active up to a speed of 10 rps.		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!vlevel	-

LS_GetSpeedPoti			
Description:	Inquires whether the potentiometer is activated or deactivated.		
Delphi:	function LS_GetSpeedPoti(var SpePoti: LongBool): Integer; 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:	LS.GetSpeedPoti(&flag);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?pot	-

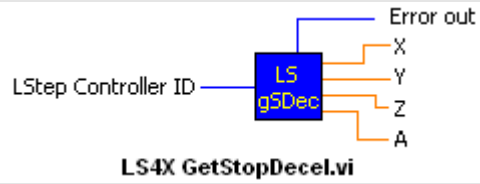
LS_SetSpeedPoti			
Description:	Activation or deactivation of potentiometer. If it is activated, the moving speed is used on a percentage basis dependent on the position of the potentiometer. If it is deactivated, the speed specified before is used. In manual joystick operation, potentiometer evaluation is always active.		
Delphi:	function LS_SetSpeedPoti(SpeedPoti: LongBool): Integer; function LSX_SetSpeedPoti(LSID: Integer; SpeedPoti: LongBool): Integer;		
C++:	int SetSpeedPoti(BOOL SpeedPoti);		
LabView:			

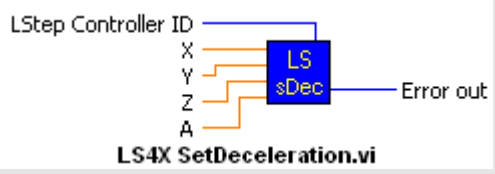
Parameter:	SpePoti	Potentiometer status True = All power amplifiers are activated False = All power amplifiers are deactivated	
Example:	LS.SetSpeedPoti(true);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!pot	-

LS_GetStopAccel			
Description:	Shows the brake acceleration, if the stop input is active (not for LSTEPexpress).		
Delphi:	function LS_GetStopAccel(var XD, YD, ZD, AD: Double): Integer; 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:	LS.GetStopAccel(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?stopaccel	-

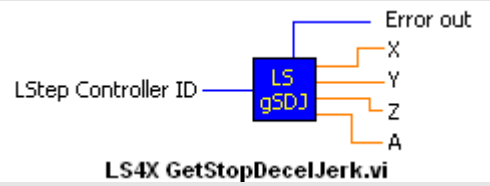
LS_SetStopAccel			
Description:	Function for setting the brake acceleration, if the stop input is active This value only applies to vector operation, not for: joystick, calibration and stroke measurement. Apart from this, it is not stored with a Save in the controllers of the LSTEP 2000 series (not for LSTEPexpress).		
Delphi:	function LS_SetStopAccel(XD, YD, ZD, AD: Double): Integer; 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	

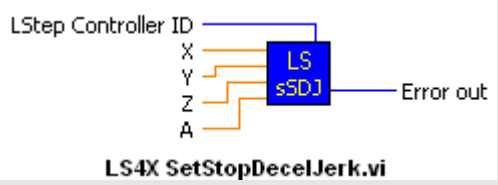
Example:	LS.SetStopAccel(1.0, 1.5, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!stopaccel	-

LS_GetStopDecel			
Description:	Sets the brake deceleration for an active stop input (only for LSTEPexpress).		
Delphi:	function LS_GetStopDecel(var XD, YD, ZD, AD: Double): Integer; function LSX_GetStopDecel(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetStopDecel(double *pdXD, double *pdYD, double *pdZD, double *pdAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Deceleration values in the set dimension/s^3	
Example:	LS.GetStopDecel(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	?stopdecel	-

LS_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 LS_SetDecel. If it is lower, a stop with the brake acceleration set by LS_SetDecel is effected (only for LSTEPexpress).		
Delphi:	function LS_SetStopDecel(XD, YD, ZD, AD: Double): Integer; function LSX_SetStopDecel(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetStopDecel(double dXD, double dYD, double dZD, double dAD);		
LabView:			

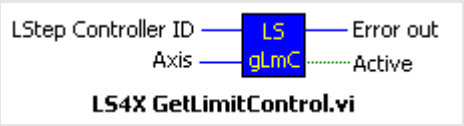
Parameter:	XD, YD, ZD, AD	Deceleration values in the set dimension/s³	
Example:	LS.SetStopDecel(1.0, 1.5, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	!stopdecel	-

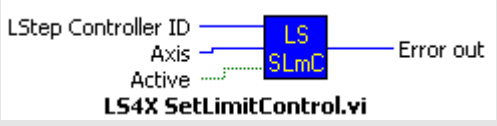
LS_GetStopDecelJerk			
Description:	Function to stop the jerk used in order to build up or relieve the deceleration in case of an emergency stop signal (only for LSTEPexpress).		
Delphi:	function LS_GetStopDecelJerk(var XD, YD, ZD, AD: Double): Integer; function LSX_GetStopDecelJerk(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetStopDecelJerk(double *pdXD, double *pdYD, double *pdZD, double *pdAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Jerk values in the set dimension/s³	
Example:	LS.GetStopDecelJerk(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	?stopdeceljerk	-

LS_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 LS_SetDecelJerk. If it is lower, a stop with the brake jerk set by LS_SetDecelJerk is effected (only for LSTEPexpress).		
Delphi:	function LS_SetStopDecelJerk (XD, YD, ZD, AD: Double): Integer; function LSX_SetStopDecelJerk(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetStopDecelJerk(double dXD, double dYD, double dZD, double dAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Jerk values in the set dimension/s³	

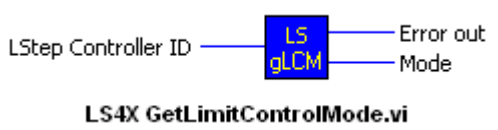
Example:	LS.SetStopDecelJerk(1000, 1500, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	!stopdeceljerk	-

4.2.7 Limit switches and software limits

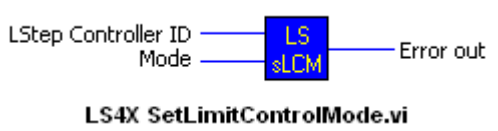
LS_GetLimitControl			
Description:	This function reads whether the range monitoring is activated or deactivated from the controller.		
Delphi:	function LS_GetLimitControl(Axis: Integer; var Active: LongBool): Integer; function LSX_GetLimitControl(LSID: Integer; Axis: Integer; var Active: LongBool): Integer;		
C++:	int GetLimitControl(int lAxis, BOOL *pbActive);		
LabView:			
Parameter:	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:	LS.GetLimitControl(2, &Active); // Active = False means: Range monitoring of axis y is deactivated.		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?limctr	-

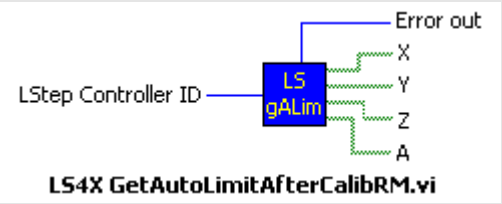
LS_SetLimitControl	
Description:	Activates or deactivates the range monitoring of the controller.
Delphi:	function LS_SetLimitControl(Axis: Integer; Active: LongBool): Integer; function LSX_SetLimitControl(LSID: Integer; Axis: Integer; Active: LongBool): Integer;
C++:	int SetLimitControl(int lAxis,BOOL Active);
LabView:	

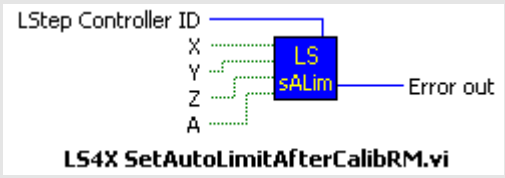
Parameter:	Axis	Axis where the range monitoring is to be written 1 = X-axis 2 = Y-axis …	
	Active	Value of range monitoring to be set True = Range monitoring is active False = Range monitoring is not active	
Example:	LS.SetLimitControl(2, true); // range monitoring of Y-axis is active		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!limctr	-

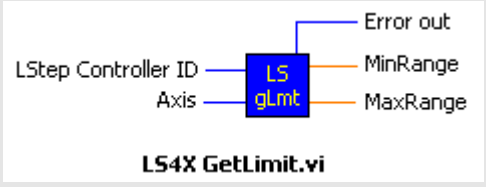
LS_GetLimitControlMode			
Description:	Shows the mode for controlling the software limits (not for LSTEPexpress).		
Delphi:	function LS_GetLimitControlMode (var Mode: Integer): Integer; function LSX_GetLimitControlMode (LSID: Integer; var Mode: Integer): Integer;		
C++:	int GetLimitControlMode(int *plMode);		
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:	LS.GetLimitControlMode(&lMode);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?limmode	-

LS_SetLimitControlMode	
Description:	Sets the mode for monitoring the software limits (not for LSTEPexpress).
Delphi:	function LS_SetLimitControlMode (Mode: Integer): Integer; function LSX_SetLimitControlMode (LSID: Integer; Mode: Integer): Integer;
C++:	int SetLimitControlMode (int lMode);

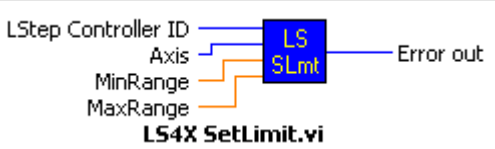
LabView:			
Parameter:	Mode	Mode to be set 0 = Moves outside of the positioning range will only be executed up to the limits of the positioning range. 1 = Moves outside the positioning range will not be executed.	
Example:	LS.SetLimitControlMode(1);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!limmode	-

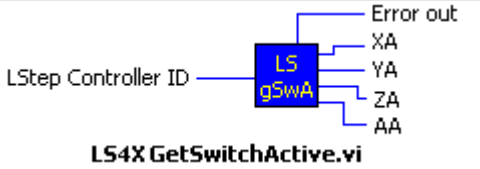
LS_GetAutoLimitAfterCalibRM			
Description:	Indicates whether the internal software limits are set during calibration and table stroke measuring.		
Delphi:	function LS_GetAutoLimitAfterCalibRM(var lFlags: Integer): Integer; function LSX_GetAutoLimitAfterCalibRM(LSID: Integer; var lFlags: Integer): Integer;		
C++:	int GetAutoLimitAfterCalibRM(int *plFlags);		
LabView:			
Parameter:	lFlags	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:	LS.SetAutoLimitAfterCalibRM(&lFlags);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?nosetlimit	-

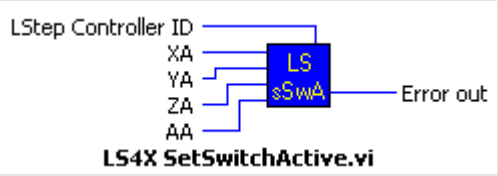
LS_SetAutoLimitAfterCalibRM			
Description:	Prevents that the internal software limits are set during calibration and table stroke measuring.		
Delphi:	function LS_SetAutoLimitAfterCalibRM(lFlags: Integer): Integer; function LSX_SetAutoLimitAfterCalibRM(LSID: Integer; lFlags: Integer): Integer;		
C++:	int SetAutoLimitAfterCalibRM(int lFlags);		
LabView:			
Parameter:	lFlags	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:	LS.SetAutoLimitAfterCalibRM(lFlags);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!nosetlimit	-

LS_GetLimit	
Description:	Reads the travel range limits of the axes. As regards controllers of the LSTEPexpress series, the error code 4032 is returned for invalid range limits.
Delphi:	function LS_GetLimit(Axis: Integer; var MinRange, MaxRange: Double): Integer; function LSX_GetLimit(LSID: Integer; Axis: Integer; var MinRange, MaxRange: Double): Integer;
C++:	int GetLimit(int lAxis, double *pdMinRange, double *pdMaxRange);
LabView:	

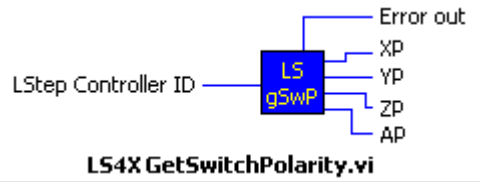
Parameter:	Axis	Axis from where the range limits are to be read. 1 = X-axis 2 = Y-axis …	
	MinRange	Bottom range limit in the set axis dimension	
	MaxRange	Top range limit in the set axis dimension	
Example:	LS.GetLimit(1, &MinRange, &MaxRange);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?lim	-

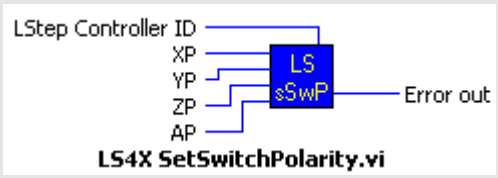
LS_SetLimit			
Description:	Sets the travel range limits of an axis.		
Delphi:	function LS_SetLimit(Axis: Integer; MinRange, MaxRange: Double): Integer; function LSX_SetLimit(LSID: Integer; Axis: Integer; MinRange, MaxRange: Double): Integer;		
C++:	int SetLimit(int lAxis,double dMinRange,double dMaxRange);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

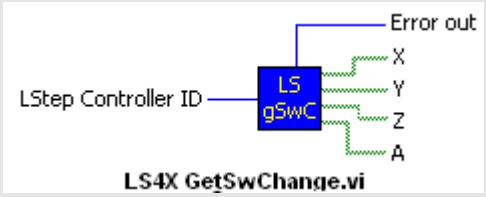
LS_GetSwitchActive			
Description:	This function reads which limit switches were configured for monitoring.		
Delphi:	function LS_GetSwitchActive(var XA, YA, ZA, AA: Integer): Integer; function LSX_GetSwitchActive(LSID: Integer; var XA, YA, ZA, AA: Integer): Integer;		
C++:	int GetSwitchActive(int *plXA, int *plYA, int *plZA, int *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 LSTEPexpress) Bit 2 = End limit switch configuration	
Example:	LS.GetSwitchActive(&XA, &YA, &ZA, &AA);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?swact	-

LS_SetSwitchActive	
Description:	Activates limit switch monitoring
Delphi:	function LS_SetSwitchActive(XA, YA, ZA, AA: Integer): Integer; function LSX_SetSwitchActive(LSID: Integer; XA, YA, ZA, AA: Integer): Integer;
C++:	int SetSwitchActive(int lXA,int lYA,int lZA,int lAA);
LabView:	

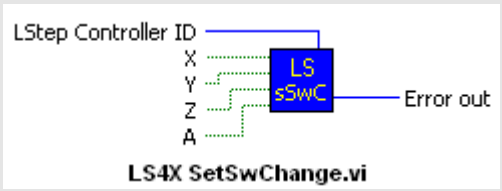
Parameter:	XA, YA, ZA, AA	Bit mask over limit switch configuration Bit 0 = Zero limit switch configuration Bit 1 = Reference limit switch configuration (always 0 for LSTEPexpress) Bit 2 = End limit switch configuration	
Example:	LS.SetSwitchActive(7, 1, 5, 0); // All X-axis limit switches On; Y-axis zero limit switch On; Z-axis E0 and EE On; A-axis: all limit switches Off		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!swact	-

LS_GetSwitchPolarity			
Description:	Function for reading the set limit switch polarity.		
Delphi:	function LS_GetSwitchPolarity(var XP, YP, ZP, AP: Integer): Integer; function LSX_GetSwitchPolarity(LSID: Integer; var XP, YP, ZP, AP: Integer): Integer;		
C++:	int GetSwitchPolarity(int *plXP, int *plYP, int *plZP, int *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 LSTEPexpress) Bit 2 = End limit switch polarity Value 0 = Reacts on negative limit switch edge Value 1 = Reacts on positive limit switch edge	
Example:	LS.GetSwitchPolarity(&XP, &YP, &ZP, &AP);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?swpol	-

LS_SetSwitchPolarity			
Description:	Function for setting the set limit switch polarity.		
Delphi:	function LS_SetSwitchPolarity(XP, YP, ZP, AP: Integer): Integer; function LSX_SetSwitchPolarity(LSID: Integer; XP, YP, ZP, AP: Integer): Integer;		
C++:	int SetSwitchPolarity(int lXP,int lYP,int lZP,int lAA);		
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 (no effect with LSTEPexpress) Bit 2 = End limit switch polarity Value 0 = Reacts on negative limit switch edge Value 1 = Reacts on positive limit switch edge	
Example:	LS.SetSwitchPolarity(7, 0, 0, 0); // All X-axis limit switches high-active, all Y-axis limit switches low-active...)		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!swpol	validconfig

LS_GetSwChange		
Description:	Function for reading the setting Limit switch change (only for LSTEPexpress).	
Delphi:	function LS_GetSwChange(var Flags: Integer): Integer; function LSX_GetSwChange(LSID: Integer; var Flags: Integer): Integer;	
C++:	int GetSwChange(int *plFlags);	
LabView:		
Parameter:	Flags	32-bit integer containing a bit mask after calling the function in the bits 0-4. Bit 0 = X-axis Bit 1 = Y-axis … Value 0 = No change of limit switch Value 1 = Change limit switch

Example:	LS.GetSwChange(&Flags);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	?swchange	-

LS_SetSwChange			
Description:	Function for setting Limit switch change (only for LSTEPexpress).		
Delphi:	function LS_SetSwChange(Flags: Integer): Integer; 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:	LS.SetSwChange(3); /* X and Y-axis – Change limit switch (bits 0 and 1 set), Z and A-axis – No change of limit switch (bit 2 = 0) */		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	!swchange	validconfig

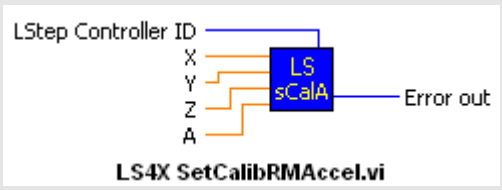
LS_GetSwitches		
Description:	This function reads the status of all limit switches.	
Delphi:	function LS_GetSwitches(var Flags: Integer): Integer; function LSX_GetSwitches(LSID: Integer; var Flags: Integer): Integer;	
C++:	int GetSwitches(int *plFlags);	
LabView:		

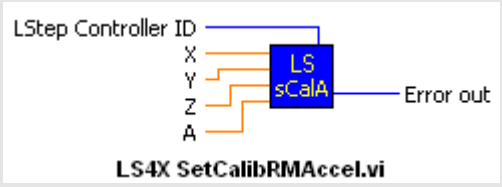
<table>
<tr><td>Parameter:</td><td>Value</td><td colspan="3">Pointer to an integer value containing the status of all limit switches as a bit mask after calling the function.
The limit switch statuses are encoded in the bit mask as follows:

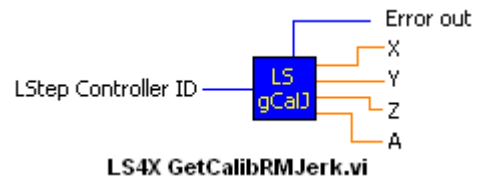
Limit switch EE Ref. E0
Axis AZYX AZYX AZYX
Bit 0000 0000 0000

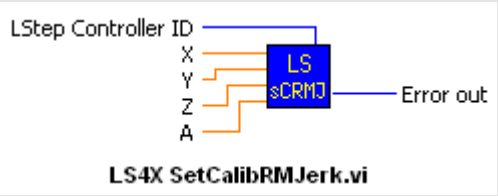
e.g.
Flags = 0x003 → E0 of X and Y-Axis are reached
Flags = 0x200 → EE of Y-Axis is reached</td></tr>
<tr><td>Example:</td><td colspan="4">LS.GetSwitches(&Flags);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td colspan="2">Activation (LSTEPexpress series)</td></tr>
<tr><td>3</td><td>?readsw</td><td colspan="2">-</td></tr>
</table>

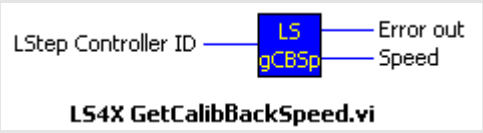
4.2.8 Reference travel

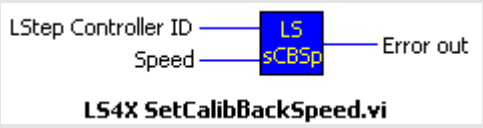
LS_GetCalibRMAccel			
Description:	Inquires the acceleration to be used for calibration (only for LSTEPexpress).		
Delphi:	function LS_GetCalibRMAccel(var XD, YD, ZD, AD: Double): Integer; function LSX_GetCalibRMAccel(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetCalibRMAccel(double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

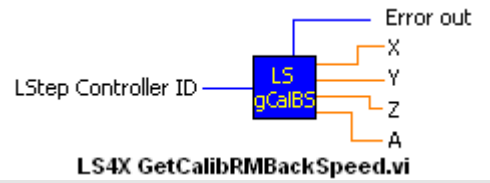
LS_SetCalibRMAccel			
Description:	Sets acceleration for calibration process (only for LSTEPexpress).		
Delphi:	function LS_SetCalibRMAccel(XD, YD, ZD, AD: Double): Integer; function LSX_SetCalibRMAccel(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetCalibRMAccel(double dXD, double dYD,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

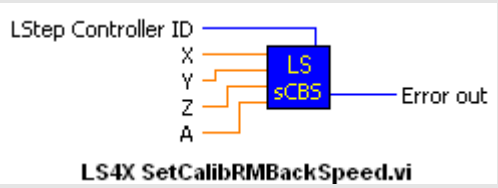
LS_GetCalibRMJerk			
Description:	Inquiry of the jerk to be used during the calibration process (only for LSTEPexpress).		
Delphi:	function LS_GetCalibRMJerk(var XD, YD, ZD, AD: Double): Integer; function LSX_GetCalibRMJerk(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;		
C++:	int GetCalibRMJerk(double *pdXD, double *pdYD, double *pdZD, double *pdAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Jerk values in the set dimension/s³	
Example:	LS.GetCalibRMJerk(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	?calibrmjerk	-

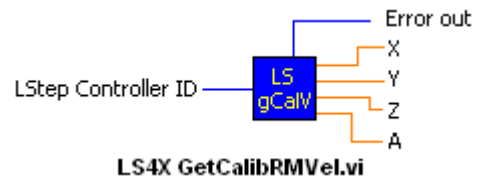
LS_SetCalibRMJerk			
Description:	Setting of the jerk to be used during the calibration process (only for LSTEPexpress).		
Delphi:	function LS_SetCalibRMJerk(XD, YD, ZD, AD: Double): Integer; function LSX_SetCalibRMJerk(LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetCalibRMJerk(double dXD, double dYD, double dZD, double dAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Jerk values in the set dimension/s³	
Example:	LS.SetCalibRMJerk(1.0, 1.5, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	!calibrmjerk	-

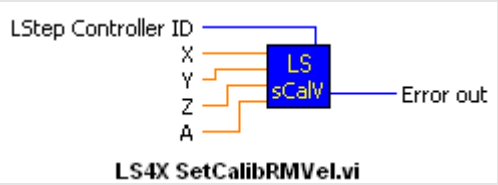
LS_GetCalibBackSpeed			
Description:	Reads positioning speeds for leaving the limit switch during the calibration process. (Only for LSTEP-2000 series. As regards LSTEPexpress, LS_GetCalibRMBackSpeed is to be used.)		
Delphi:	function LS_GetCalibBackSpeed(var Speed: Integer): Integer; function LSX_GetCalibBackSpeed(LSID: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

LS_SetCalibBackSpeed			
Description:	Sets the positioning speeds for leaving the limit switches during the calibration process. (Only for LSTEP-2000 series. As regards LSTEPexpress, LS_SetCalibRMBackSpeed is to be used.)		
Delphi:	function LS_SetCalibBackSpeed(Speed: Integer): Integer; 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:	LS.SetCalibBackSpeed (10);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!calbspeed	-

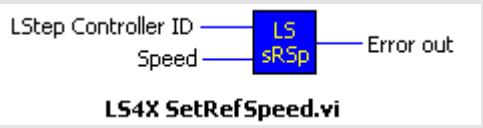
LS_GetCalibRMBackSpeed			
Description:	Inquiry of positioning speed for leaving the limit switches to be used during the calibration process or the table stroke measuring (only for LSTEPexpress).		
Delphi:	function LS_GetCalibRMBackSpeed(var XD, YD, ZD, AD: Double): Integer; function LSX_GetCalibRMBackSpeed(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;		
C++:	int GetCalibRMBackSpeed(double *pdXD, double *pdYD, double *pdZD, double *pdAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	Speed values in the set dimension/s	
Example:	LS.GetCalibRMBackSpeed(&XD, &YD, &ZD, &AD);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	?calibrmbspeed	-

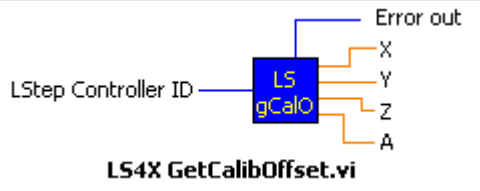
LS_SetCalibRMBackSpeed			
Description:	Setting of positioning speeds for leaving the limit switches to be used during the calibration process or the table stroke measuring (only for LSTEPexpress).		
Delphi:	function LS_SetCalibRMBackSpeed(XD, YD, ZD, AD: Double): Integer; function LSX_SetCalibRMBackSpeed(LSID: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

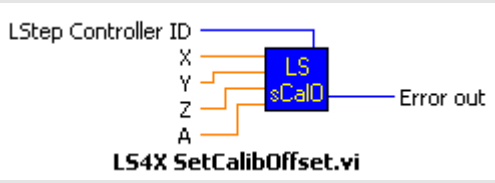
LS_GetCalibRMVel			
Description:	Inquiry of the positioning speed to be used during the calibration process (only for LSTEPexpress).		
Delphi:	function LS_GetCalibRMVel(var XD, YD, ZD, AD: Double): Integer; function LSX_GetCalibRMVel(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;		
C++:	int GetCalibRMVel(double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

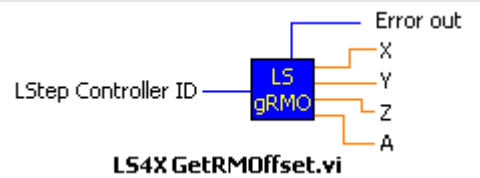
LS_SetCalibRMVel			
Description:	Setting of the positioning speeds to be used during the calibration process (only for LSTEPexpress).		
Delphi:	function LS_SetCalibRMVel (XD, YD, ZD, AD: Double): Integer; function LSX_SetCalibRMVel (LSID: Integer; XD, YD, ZD, AD: Double): Integer;		
C++:	int SetCalibRMVel(double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

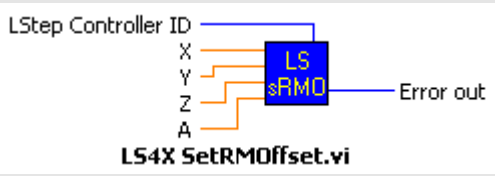
LS_GetRefSpeed			
Description:	Reads the rotation speed at which the axes move while searching the reference mark. The speed is equivalent to the output value * 0.01 rps (not for LSTEPexpress).		
Delphi:	function LS_GetRefSpeed(var lSpeed: Integer): Integer; function LSX_GetRefSpeed(LSID: Integer; var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

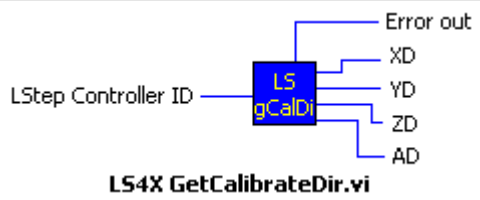
LS_SetRefSpeed			
Description:	Sets the rotation speed at which the axes move while searching the reference mark. The speed is equivalent to the output value * 0.01 rps (not for LSTEPexpress).		
Delphi:	function LS_SetRefSpeed(lSpeed: Integer): Integer; function LSX_SetRefSpeed(LSID: Integer; lSpeed: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

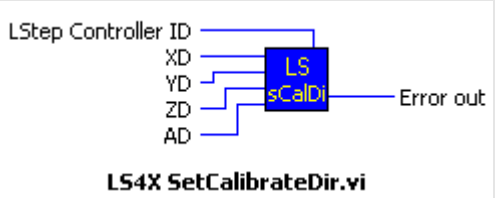
LS_GetCalibOffset			
Description:	Function for inquiring a calibration offset controlled during calibration after moving away from the limit switch.		
Delphi:	function LS_GetCalibOffset(var X, Y, Z, A: Double): Integer; function LSX_GetCalibOffset(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++: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

LS_SetCalibOffset			
Description:	Function for setting a calibration offset controlled during calibration after moving away from the limit switch.		
Delphi:	function LS_SetCalibOffset(X, Y, Z, A: Double): Integer; 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:	LS.SetCalibOffset(1, 1, 1, 1);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!caliboffset	-

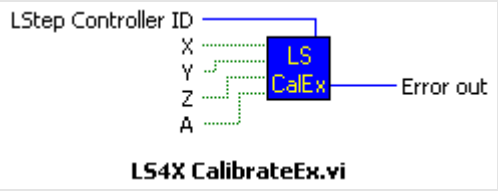
LS_GetRMOffset			
Description:	Inquiry of the set offset for the next table stroke measurement.		
Delphi:	function LS_GetRMOffset(var X, Y, Z, A: Double): Integer; function LSX_GetRMOffset(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetRMOffset(double *pdX, double *pdY, double *pdZ, double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

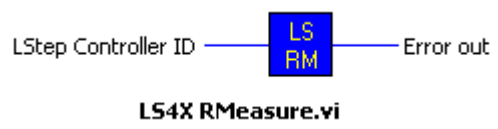
LS_SetRMOffset			
Description:	Setting of the offset for the next table stroke measurement.		
Delphi:	function LS_SetRMOffset(X, Y, Z, A: Double): Integer; function LSX_SetRMOffset(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetRMOffset (double dX, double dY,double dZ,double dA);		
LabView: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

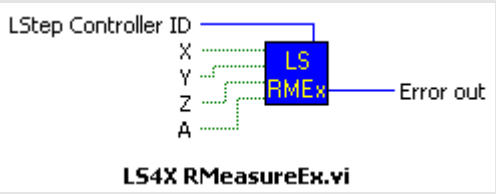
LS_GetCalibrateDir			
Description:	Inquiry whether the sign reversal is active during calibration.		
Delphi:	function LS_GetCalibrateDir(var XD, YD, ZD, AD: Integer): Integer; function LSX_GetCalibrateDir(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;		
C++:	int GetCalibrateDir (int *plXD, int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

LS_SetCalibrateDir			
Description:	Activation of sign reversal during calibration		
Delphi:	function LS_SetCalibrateDir(XD, YD, ZD, AD: Integer): Integer; function LSX_SetCalibrateDir(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;		
C++:	int SetCalibrateDir(int lXD, int lYD, int lZD, int lAD);		
LabView:			
Parameter:	XD, YD, ZD, AD	32-bit integer with indication of sign reversal. 0 = No sign reversal 1 = Sign reversal	
Example:	LS.SetCalibrateDir(1, 1, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!caldir	-

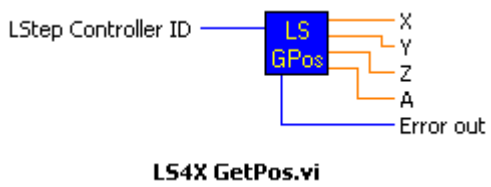
LS_Calibrate			
Description:	This function starts the calibration routine. All enabled axes are moved towards lower position values. The movements are interrupted as soon as the limit switches are reached. The position value is set to 0.		
Delphi:	function LS_Calibrate: Integer; function LSX_Calibrate(LSID: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

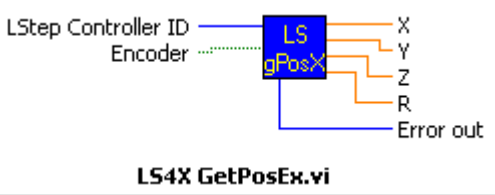
LS_CalibrateEx			
Description:	Function for starting the calibration routine of a single axis		
Delphi:	function LS_CalibrateEx(Flags: Integer): Integer; function LSX_CalibrateEx(LSID: Integer; Flags: Integer): Integer;		
C++:	int CalibrateEx(int lFlags);		
LabView:			
Parameter:	Flags	Bit mask Bit 0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

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

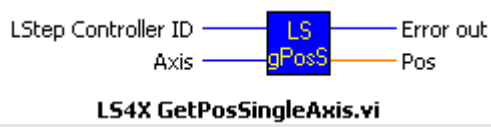
LS_RMeasureEx			
Description:	Starts the table stroke measuring process. Only the specified axes are moved towards higher position values. The movements are interrupted as soon as the limit switches are reached.		
Delphi:	function LS_RMeasureEx(Flags: Integer): Integer; function LSX_RMeasureEx(LSID: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

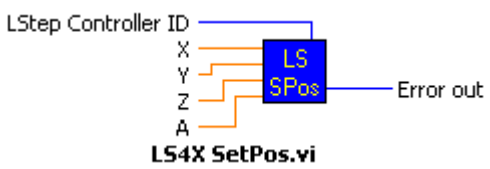
4.2.9 Travel commands and position administration

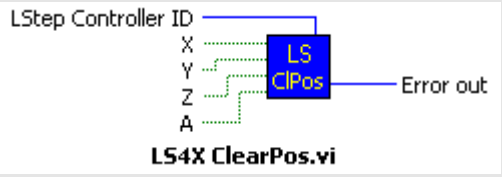
LS_GetPos			
Description:	Inquiry of current encoder or position values of all axes. For non-existing axes, a value of 0.0 is returned		
Delphi:	function LS_GetPos(var X, Y, Z, A: Double): Integer; function LSX_GetPos(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetPos(double *pdX,double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

LS_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 LS_GetPosEx(var X, Y, Z, A: Double; Encoder: LongBool): Integer; function LSX_GetPosEx(LSID: Integer; var X, Y, Z, R: Double; Encoder: LongBool): Integer;
C++:	int GetPosEx(double *pdX,double *pdY,double *pdZ,double *pdA,BOOL Encoder);
LabView:	

Parameter:	X, Y, Z, A	Position values in the set unit	
	Encoder	Read encoder value True = Show encoder values if encoder is connected False = Show position values	
Example:	LS.GetPosEx(&X, &Y, &Z, &A, true);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!enc ?pos	-

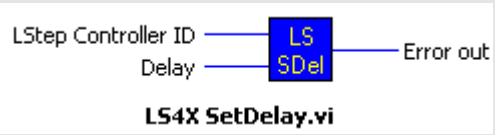
LS_GetPosSingleAxis			
Description:	Inquiry of current position of a single axis. For non-existing axes, a value of 0.0 is returned		
Delphi:	Function LS_GetPosSingleAxis(Axis: Integer; var Pos: Double): Integer; function LSX_GetPosSingleAxis(LSID: Integer; Axis: Integer; var Pos: Double): Integer;	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

LS_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 LS_SetPos(X, Y, Z, A: Double): Integer; function LSX_SetPos(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetPos(double dX, double dY, double dZ, double dA);		
LabView:			
Parameter:	X, Y, Z, A	Position values in the set unit of the axis	
Example:	LS.SetPos(10, 10, 0, 0);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!pos	-

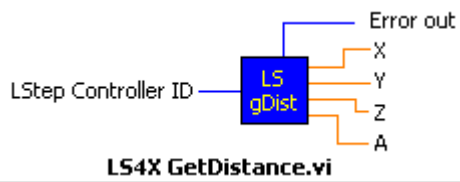
LS_ClearPos	
Description:	Sets the position to 0, including the internal counter. This function is needed for endless axes, because the controller can only process a range of +1000 motor revolutions. The function for an identified encoder is not carried out for the relevant axis. (not for LSTEPexpress, see Modulo operation)
Delphi:	function LS_ClearPos(lFlags: Integer): Integer; function LSX_ClearPos(LSID: Integer; lFlags: Integer): Integer;
C++:	int ClearPos (int lFlags);
LabView:	

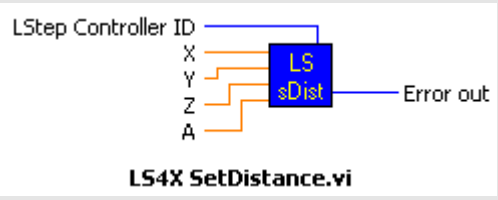
Parameter:	lFlags	Bit mask Bit 0 = X-axis Bit 1 = Y-axis … Value 0 = Position is not reset to zero Value 1 = Position is reset to zero	
Example:	LS.ClearPos(5); //Positions of x and z-axes are reset to zero.		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!clearpos	-

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

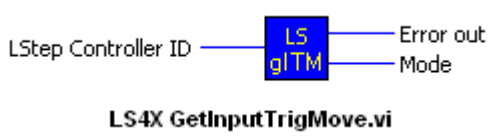
LS_SetDelay			
Description:	The delay command is used to generate e a vector start delay. (not for LSTEPexpress).		
Delphi:	function LS_SetDelay(Delay: Integer): Integer; function LSX_SetDelay(LSID: Integer; Delay: Integer): Integer;		
C++:	int SetDelay(int lDelay);		
LabView:			

Parameter:	Delay	Delay in ms	
Example:	LS.SetDelay(1000); // 1s delay		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!delay	-

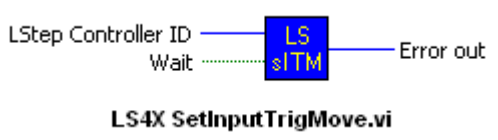
LS_GetDistance			
Description:	Delivers the distance for LS_MoveRelShort		
Delphi:	function LS_GetDistance(var X, Y, Z, A: Double): Integer; function LSX_GetDistance(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetDistance(double *pdX, double *pdY, double *pdZ, double *pdR);		
LabView: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

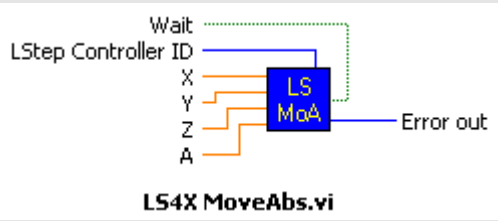
LS_SetDistance			
Description:	Set distance for LS_MoveRelShort		
Delphi:	function LS_SetDistance(X, Y, Z, A: Double): Integer; function LSX_SetDistance(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetDistance(double dX,double dY,double dZ,double dA);		
LabView:			

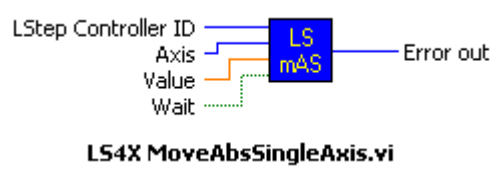
Parameter:	X, Y, Z, A	Distance to be travelled, dependent on the set axis dimension.	
Example:	LS.SetDistance(1, 2, 0, 0); /* Distances are set for the X and Y-axis, Z and A are not moved when the function LS_MoveRelShort is activated. */		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!distance	-

LS_GetInputTrigMove			
Description:	Shows the configuration of Pin1 on the MFP (not for LSTEPexpress).		
Delphi:	function LS_GetInputTrigMove (var Mode: Integer): Integer; function LSX_GetInputTrigMove (LSID: Integer; var Mode: Integer): Integer;		
C++:	int GetInputTrigMove (int *plMode);		
LabView:			
Parameter:	lMode	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:	LS.GetInputTrigMove(&lMode);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?itm	-

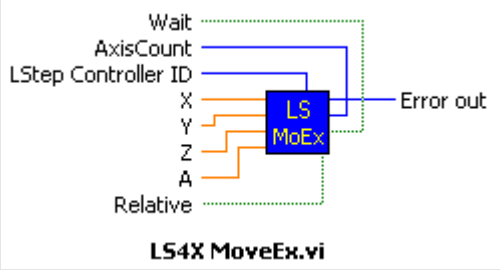
LS_SetInputTrigMove			
Description:	Configures the Pin 1 on the MFP so that a move may be started with an external signal. The movement is made in accordance with the value set by function LS_SetDistance (not for LSTEPexpress).		
Delphi:	function LS_SetInputTrigMove(Mode: Integer; Wait: LongBool): Integer; function LSX_SetInputTrigMove(LSID: Integer; Mode: Integer; Wait: LongBool): Integer;		
C++:	int SetInputTrigMove(int lMode, BOOL bWait);		

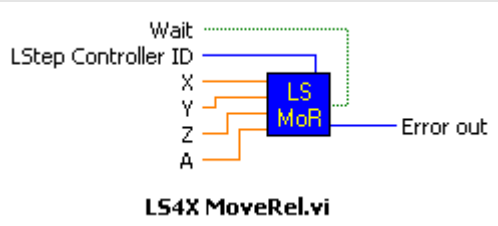
LabView:			
Parameter:	lMode	Mode to be set 0 = Function not active 1 = Absolute positioning at positive edge 2 = Absolute positioning at negative edge 3 = Relative positioning at positive edge 4 = Relative positioning at negative edge	
Parameter:	bWait	Wait for a move 1 = Waiting until a move is made after receiving an external signal; subsequently, the mode is set to 0. 0 = No move is awaited. bWait is not evaluated if IMode = 0.	
Example:	LS.SetInputTrigMove(3, False);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!itm	-

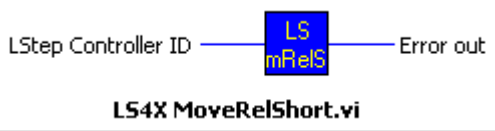
LS_MoveAbs			
Description:	Approaching an absolute position from the current position. The move is interpolated linearly.		
Delphi:	function LS_MoveAbs(X, Y, Z, A: Double; Wait: LongBool): Integer; function LSX_MoveAbs(LSID: Integer; X, Y, Z, A: Double; Wait: LongBool): Integer;		
C++:	int MoveAbs (double dX, double dY, double dZ, double dA, BOOL Wait);		
LabView:			
Parameter:	X, Y, Z, A	Absolute position in the set axis dimension.	
	Wait	Wait for the end of the movement True = Wait False = Do not wait	
Example:	LS.MoveAbs(10.0, 10.0, 10.0, 10.0, true);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!moa	-

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

LS_MoveEx	
Description:	The function LS_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 LS_MoveEx(X, Y, Z, A: Double; Relative, Wait: LongBool; AxisCount: Integer): Integer; 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);

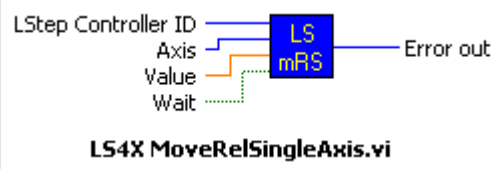
<table>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td rowspan="4">Parameter:</td><td>X, Y, Z, A</td><td colspan="2">Position vector in the set unit of the axis</td></tr>
<tr><td>Relative</td><td colspan="2">Indicates whether the position vector is to be moved relatively
True = Vector is moved relatively to the current position
False = Vector is moved absolutely to the current position</td></tr>
<tr><td>Wait</td><td colspan="2">Wait for end of movement
True = The function only returns after reaching the target position.
False = The function returns immediately after sending the command.</td></tr>
<tr><td>AxisCount</td><td colspan="2">Number of axes to be moved
1 = X-axis on
2 = X and Y-axis
3 = X, Y and Z-axis
…</td></tr>
<tr><td>Example:</td><td colspan="3">LS_MoveEx(2.0, 3.0, 0, 0, true, true, 2) ; // X and Y are moved relative by 2 or 3</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEPexpress series)</td></tr>
<tr><td>3</td><td>!moa / !mor</td><td>-</td></tr>
</table>

LS_MoveRel			
Description:	Function for moving a relative vector from the current position		
Delphi:	function LS_MoveRel(X, Y, Z, A: Double; Wait: LongBool): Integer; function LSX_MoveRel(LSID: Integer; X, Y, Z, A: Double; Wait: LongBool): Integer;		
C++:	int MoveRel(double dX, double dY, double dZ, double dA, BOOL Wait);		
LabView:			
Parameter:	X, Y, Z, A	Relative position indication in the set axis dimension	
	Wait	Wait for the end of the movement True = Wait False = Do not wait	
Example:	LS.MoveRel(10.0, 10.0, 10.0, 10.0, true);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!mor	-

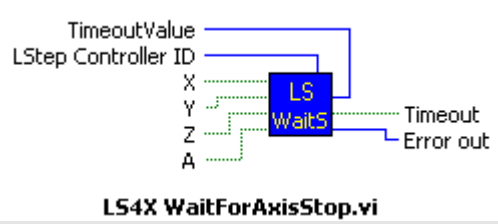
LS_MoveRelShort			
Description:	This command should be used, so that a series of consecutive relative travel commands (of the same distance) are controlled more quickly. The distance must previously have been set with LS_SetDistance once.		
Delphi:	function LS_MoveRelShort: Integer; function LSX_MoveRelShort(LSID: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

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

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

LS_MoveRelSingleAxis			
Description:	Relative positioning of a single axis from the current position.		
Delphi:	function LS_MoveRelSingleAxis(Axis: Integer; Value: Double; Wait: LongBool): Integer; function LSX_MoveRelSingleAxis(LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;		
C++:	int MoveRelSingleAxis(int lAxis,double dValue,BOOL Wait);		
LabView:			
Parameter:	Axis	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:	LS.MoveRelSingleAxis(3, 5.0); // Move Z-axis by 5mm in positive direction		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!mor	-

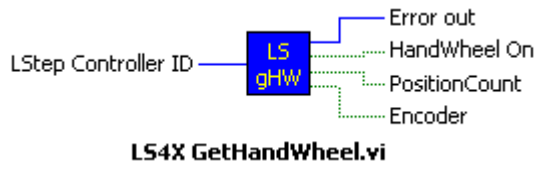
LS_StopAxes			
Description	Function for stopping all movements.		
Delphi	function LS_StopAxes: Integer; function LSX_StopAxes(LSID: Integer): Integer;		
C++	int StopAxes ();		
LabView			
Parameter	-		
Example	LS.StopAxes();		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)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

LS_WaitForAxisStop			
Description:	The function returns as soon as the axes selected in the bit mask AFlags have reached their target position. LS_WaitForAxisStop uses ‚?statusaxis' to poll the status of the axes.		
Delphi:	function LS_WaitForAxisStop(AFlags: Integer; ATimeoutValue: Integer; var ATimeout: LongBool): Integer; function LSX_WaitForAxisStop(LSID: Integer; AFlags: Integer; ATimeoutValue: Integer; var ATimeout: LongBool): Integer;		
C++:	int WaitForAxisStop(int lAFlags, int lATimeoutValue, BOOL *pbATimeout);		
LabView:			
Parameter:	AFlags	Bit mask Bit 0 = X-axis Bit 1 = Y-axis … 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	

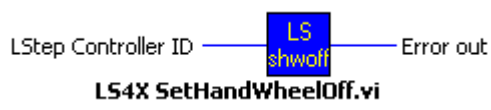
Example:	LS.WaitForAxisStop(3, 0, flag);
	// Wait until X and Y-axis have stopped, no timeout
	LS.WaitForAxisStop(7, 10000, flag);
	// Wait until X and Y-axis have stopped, 10 seconds timeout		

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

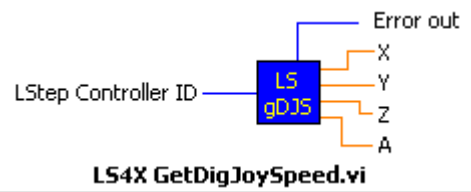
4.2.10 Joystick and handwheel

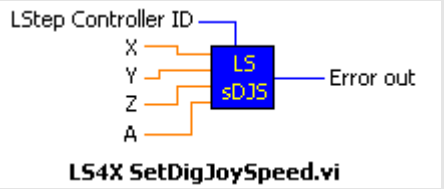
LS_GetHandWheel			
Description:	Function for reading the handwheel status (not for LSTEPexpress).		
Delphi:	function LS_GetHandWheel(var PositionCount, Encoder: Boolean): Integer; function LSX_GetHandWheel(LSID: Integer; var PositionCount, Encoder: LongBool): Integer;		
C++:	int GetHandWheel(BOOL *pbHandWheelOn, BOOL *pbPositionCount, BOOL *pbEncoder);		
LabView:			
Parameter:	HWOn	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:	LS.GetHandWheel(&HWOn, &PosCount, &Encoder);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?hw	-

LS_SetHandWheelOff	
Description:	Function deactivates the handwheel. (not for LSTEPexpress).
Delphi:	function LS_SetHandWheelOff: Integer; function LSX_SetHandWheelOff(LSID: Integer): Integer;
C++:	int SetHandWheelOff();

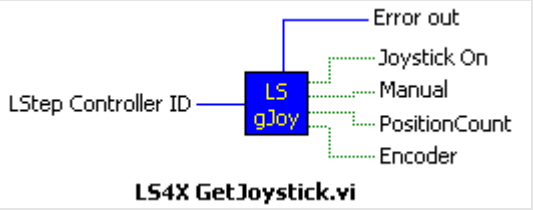
LabView:			
Parameter:	-		
Example:	LS.SetHandWheelOff();		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!hw	-

LS_SetHandWheelOn			
Description:	Function activates the handwheel (not for LSTEPexpress).		
Delphi:	function LS_SetHandWheelOn(PositionCount, Encoder: Boolean): Integer; function LSX_SetHandWheelOn(LSID: Integer; PositionCount, Encoder: LongBool): Integer;		
C++:	int SetHandWheelOn(BOOL fPositionCount,BOOL fEncoder);		
LabView:			
Parameter:	PositionCount	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:	LS.SetHandWheelOn(true, true); // Handwheel On with position counting (encoder values)		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!hw	-

LS_GetDigJoySpeed			
Description:	Reading of set speed for moving at a constant speed (not for LSTEPexpress).		
Delphi:	function LS_GetDigJoySpeed(var dX, dY, dZ, dA: Double): Integer; 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:	LS.GetDigJoySpeed(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?speed	-

LS_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 LS_SetDigJoyOff (not for LSTEPexpress).		
Delphi:	function LS_SetDigJoySpeed(dX, dY, dZ, dA: Double): Integer; 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:	LS.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 (LSTEPexpress series)
	1	!speed	-

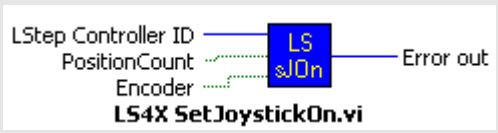
LS_SetDigJoyOff			
Description:	Deactivates the digital joystick (not for LSTEPexpress).		
Delphi:	function LS_SetDigJoyOff: Integer; function LSX_SetDigJoyOff(LSID: Integer): Integer;		
C++:	int SetDigJoyOff() ;		
LabView:			
Parameter:	-		
Example:	LS.SetDigJoyOff() ;		
Other:	Compatibility	"SendString"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

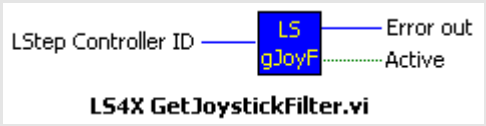
LS_GetJoystick	
Description:	Inquiry of the current condition of the analogue joystick.
Delphi:	function LS_GetJoystick(var JoystickOn, Manual, PositionCount, Encoder: LongBool): Integer; function LSX_GetJoystick(LSID: Integer; var JoystickOn, Manual, PositionCount, Encoder: LongBool): Integer;
C++:	int GetJoystick(BOOL *pbJoystickOn, BOOL *pbManual, BOOL *pbPositionCount, BOOL *pbEncoder);
LabView:	

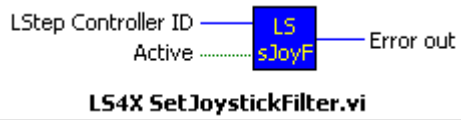
Parameter:	JoyOn	Joystick is active True = Joystick is activated False = Joystick is deactivated	
	Manual	Activation type of manual mode (not for LSTEPexpress) True = Joystick has been activated manually by switch False = Joystick is set to Automatic	
	PosCount	Position counting is active (not for LSTEPexpress) True = Position counter is activated False = Position counter is deactivated	
	Enc	Encoder values are used for position counting if available (not for LSTEPexpress). True = Encoder values are used False = Encoder values are not used	
Example:	LS.GetJoystick(&JoyOn, &Manual, &PosCount, &Enc);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?joy	-

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

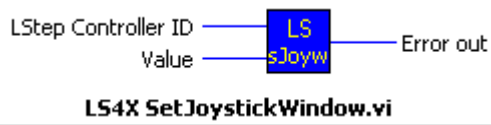
LS_SetJoystickOn	
Description:	Function activates the analogue joystick.
Delphi:	function LS_SetJoystickOn(PositionCount, Encoder: LongBool): Integer; function LSX_SetJoystickOn(LSID: Integer; PositionCount, Encoder: LongBool): Integer;
C++:	int SetJoystickOn(BOOL PositionCount,BOOL Encoder);

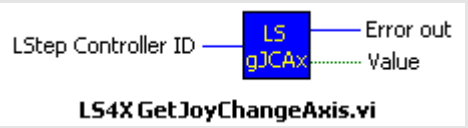
LabView:			
Parameter:	PositionCount	Activates or deactivates position counting. True = On False = Off	
	Encoder	Encoder values are used for position counting if available (not for LSTEPexpress). True = Use encoder values False = Do not use encoder values	
Example:	LS.SetJoystickOn(true, true); // Joystick On with position counting (encoder values)		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!joy	-

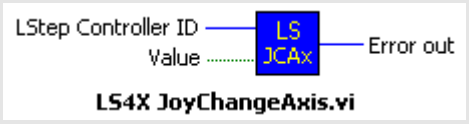
LS_GetJoystickFilter			
Description:	Indicates whether filtering and hysteresis are activated in joystick operation (not for LSTEPexpress).		
Delphi:	function LS_GetJoystickFilter(var bActive: LongBool): Integer; function LSX_GetJoystickFilter(LSID: Integer; var bActive: LongBool): Integer;		
C++:	int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

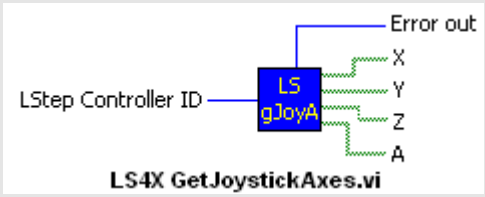
LS_SetJoystickFilter			
Description:	Activation/Deactivation of filtering and hysteresis in joystick operation (not for LSTEPexpress).		
Delphi:	function LS_SetJoystickFilter(bActive: LongBool): Integer; function LSX_SetJoystickFilter(LSID: Integer; bActive: LongBool): Integer;		
C++:	int SetJoystickFilter(BOOL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

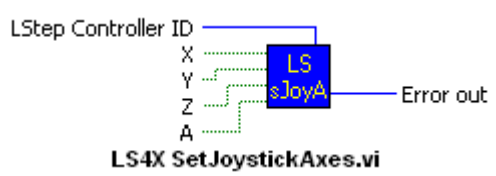
LS_GetJoystickWindow			
Description	Function for reading the joystick window. The joystick window defines and analogous range in which the axes do not move.		
Delphi	function LS_GetJoystickWindow(var AValue: Integer): Integer; function LSX_GetJoystickWindow(LSID: Integer; var AValue: Integer): Integer;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

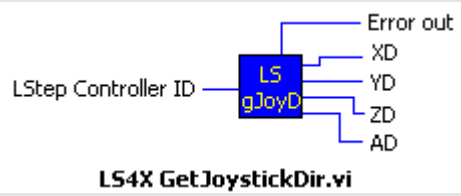
LS_SetJoystickWindow			
Description	Function for setting the joystick window. The joystick window defines and analogous range in which the axes do not move.		
Delphi	function LS_SetJoystickWindow(AValue: Integer): Integer; function LSX_SetJoystickWindow(LSID: Integer; AValue: Integer): Integer;		
C++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

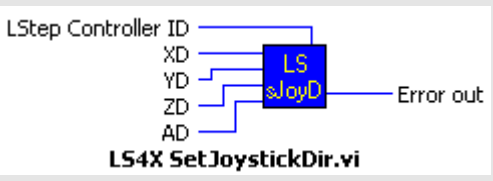
LS_GetJoyChangeAxis			
Description:	Reads joystick axis allocation (not for LSTEPexpress).		
Delphi:	LS_GetJoyChangeAxis(var Value: LongBool): Integer; LSX_GetJoyChangeAxis(LSID: Integer; var Value: LongBool): Integer;		
C++:	int GetJoyChangeAxis(BOOL *pbValue);		
LabView:			
Parameter:	Value	Axis allocation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

LS_JoyChangeAxis			
Description:	Sets joystick axis allocation (not for LSTEPexpress).		
Delphi:	LS_JoyChangeAxis(Value: LongBool): Integer; 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:	LS.JoyChangeAxis(true);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!joychangeaxis	-

LS_GetJoystickAxes			
Description:	Shows the axis for which the joystick is active if joystick operation is activated (only for LSTEPexpress).		
Delphi:	function LS_GetJoystickAxes(var Flags: Integer): Integer; function LSX_GetJoystickAxes(LSID: Integer; var Flags: Integer): Integer;		
C++:	int GetJoystickAxes(int *plFlags);		
LabView:			
Parameter:	Flags	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	
Example:	LS.GetJoystickAxes(&Flags);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	?joyenable	-

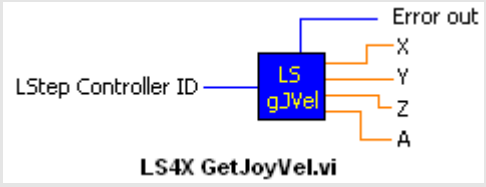
<table>
<tr><th colspan="4">LS_SetJoystickAxes</th></tr>
<tr><td>Description:</td><td colspan="3">Allows joystick operation for the indicated axes
(only for LSTEPexpress).</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LS_SetJoystickAxes(Flags: Integer): Integer;
function LSX_SetJoystickAxes(LSID: 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">LS.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 (LSTEPexpress series)</td></tr>
<tr><td>2</td><td>!joyenable</td><td>SetJoystickOn / SetJoystickOff</td></tr>
</table>

LS_GetJoystickDir			
Description:	Reads motor direction for joystick.		
Delphi:	function LS_GetJoystickDir(var XD, YD, ZD, AD: Integer): Integer; function LSX_GetJoystickDir(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;		
C++:	int GetJoystickDir(int *plXD, int *plYD, int *plZD, int *plRD);		
LabView:			
Parameter:	X, Y, Z, A	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 **LSTEPexpress series:** 0 = Normal direction of rotation 1 = Reverse direction of rotation	
Example:	LS.GetJoystickDir(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?joydir	-

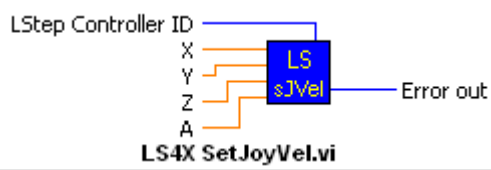
LS_SetJoystickDir			
Description:	Set joystick direction of rotation.		
Delphi:	function LS_SetJoystickDir(XD, YD, ZD, AD: Integer): Integer; function LSX_SetJoystickDir(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;		
C++:	int SetJoystickDir(int lXD,int lYD,int lZD,int lAD);		
LabView:			

Parameter:	X, Y, Z, A	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 **LSTEPexpress series:** 0 = Normal direction of rotation 1 = Reverse direction of rotation
Example:	LS.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 */	

Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!joydir	SetJoystickOn / SetJoystickOff

LS_GetJoyVel		
Description:	Inquiry of the maximum positioning speeds in joystick operation (only for LSTEPexpress).	
Delphi:	function LS_GetJoyVel(var XD, YD, ZD, AD: Double): Integer; function LSX_GetJoyVel(LSID: Integer; var XD, YD, ZD, AD: Double): Integer;	
C++:	int GetJoyVel(double *pdXD, double *pdYD, double *pdZD, double *pdAD);	
LabView:		
Parameter:	XD, YD, ZD, AD	Speed values in the set dimension/s
Example:	LS.GetJoyVel(&XD, &YD, &ZD, &AD);	

Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	2	?joyvel	-

<table>
<tr><th colspan="4">LS_SetJoyVel</th></tr>
<tr><td>Description:</td><td colspan="3">Setting of the maximum positioning speeds in joystick operation
(only for LSTEPexpress).</td></tr>
<tr><td>Delphi:</td><td colspan="3">function LS_SetJoyVel(XD, YD, ZD, AD: Double): Integer;
function LSX_SetJoyVel(LSID: Integer; XD, YD, ZD, AD: Double): Integer;</td></tr>
<tr><td>C++:</td><td colspan="3">int SetJoyVel(double dXD, double dYD, double dZD, double dAD);</td></tr>
<tr><td>LabView:</td><td colspan="3"> 
</td></tr>
<tr><td>Parameter:</td><td>XD, YD, ZD, AD</td><td colspan="2">Speed values in the set dimension/s</td></tr>
<tr><td>Example:</td><td colspan="3">LS.SetJoyVel(1.0, 15.0, 0, 0);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td>“SendString” command</td><td>Activation (LSTEPexpress series)</td></tr>
<tr><td>2</td><td>!joyvel</td><td>SetJoystickOn / SetJoystickOff</td></tr>
</table>

4.2.11 Control panel with trackball and joystick keys

LS_GetBPZ			
Description:	Reads the status of the additional control panel with track ball (not for LSTEPexpress).		
Delphi:	function LS_GetBPZ(var AValue: Integer): Integer; function LSX_GetBPZ(LSID: Integer; var AValue: Integer): Integer;		
C++:	int GetBPZ(int *plAValue);		
LabView:	-		
Parameter:	AValue	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:	LS.GetBPZ(&AValue);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?bpz	-

LS_SetBPZ			
Description	Sets the status of the additional control panel with track ball (not for LSTEPexpress).		
Delphi	function LS_SetBPZ(AValue: Integer): Integer; 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	LS.SetBPZ(1);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!bpz	-

LS_GetBPZJoyspeed			
Description:	Inquiry of control panel joystick speed (not for LSTEPexpress).		
Delphi:	function LS_GetBPZJoyspeed(APar: Integer; var AValue: Double): Integer; function LSX_GetBPZJoyspeed(LSID: Integer; APar: Integer; var AValue: Double): Integer;		
C++:	int GetBPZJoyspeed(int lAPar, double *pdAValue);		
LabView:	-		
Parameter:	APar	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 (LSTEPexpress series)
	1	?joyspeed	-

LS_SetBPZJoyspeed			
Description	Set operating panel speed of joystick (not for LSTEPexpress).		
Delphi	function LS_SetBPZJoyspeed(APar: Integer; AValue: Double): Integer; function LSX_SetBPZJoyspeed(LSID: Integer; APar: Integer; AValue: Double): Integer;		
C++	int SetBPZJoyspeed(int lAPar, double dAValue);		
LabView:	-		
Parameter:	APar	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 (LSTEPexpress series)
	1	!joyspeed	-

LS_GetBPZTrackballBacklash			
Description	Function for reading the set trackball backlash on the control panel (not for LSTEPexpress).		
Delphi	function LS_GetBPZTrackballBackLash(var X, Y, Z, A: Double): Integer; 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	LS.GetBPZTrackballBackLash(&X, &Y, &Z, &R);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?bpzbl	-

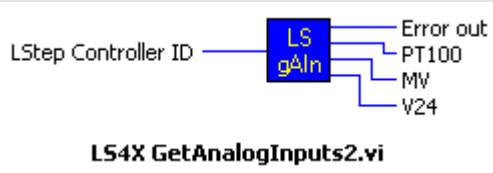
LS_SetBPZTrackballBacklash			
Description	Function for setting the trackball backlash on the control panel (not for LSTEPexpress).		
Delphi	function LS_SetBPZTrackballBackLash(X, Y, Z, A: Double): Integer; function LSX_SetBPZTrackballBackLash(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++	int SetBPZTrackballBackLash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

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

LS_SetBPZTrackballFactor			
Description	Function for setting the trackball factor on the control panel (not for LSTEPexpress).		
Delphi	function LS_SetBPZTrackballFactor(AValue: Double): Integer; 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	LS.SetBPZTrackballFactor(1.0) ;		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!bpztf	-

4.2.12 Digital and analogue inputs and outputs

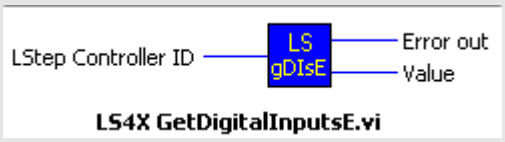
LS_GetAnalogInput			
Description:	Function for reading the current value of the analogue channel.		
Delphi:	function LS_GetAnalogInput(Index: Integer; var Value: Integer): Integer; function LSX_GetAnalogInput(LSID: Integer; Index: Integer; var Value: Integer): Integer;		
C++:	int GetAnalogInput(int lIndex,int *plValue);		
LabView:			
Parameter:	Index	Analogue channel to be read	
	Value	Pointer to integer value to where the status of the analogue channel is copied.	
Example:	LS.GetAnalogInput(0, &Eingang0);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?anain	-

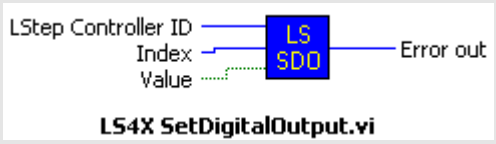
LS_GetAnalogInputs2		
Description:	Reading of the status of the analogue channels (channels 6, 7, 8). (only with the LSTEP-PCI, LSTEP-PC)	
Delphi:	function LS_GetAnalogInputs2(var PT100, MV, V24: Integer): Integer; function LSX_GetAnalogInputs2(LSID: Integer; var PT100, MV, V24: Integer): Integer;	
C++:	int GetAnalogInputs2 (int *plPT100, int *plMV, int *plV24);	
LabView:		
Parameter:	PT100, MV, V24:	Pointer to integer value where GetAnalogInputs2 is supposed to write the status of the analogue channel.
Example:	LS.GetAnalogInputs2(&PT100, &MV, &V24);	
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	-	-

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

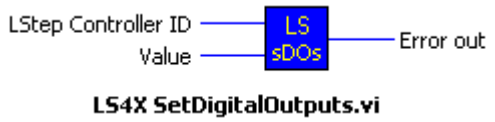
LS_GetDigitalInputs		
Description:	Function for reading the digital inputs 0 through 15	
Delphi:	function LS_GetDigitalInputs(var Value: Integer): Integer; function LSX_GetDigitalInputs(LSID: Integer; var Value: Integer): Integer;	
C++:	int GetDigitalInputs(int *plValue);	
LabView:		
Parameter:	Value	Pointer to integer value 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; LS.GetDigitalInputs(&Eingange); if (inputs & 16) ... // if Input pin 4 is set	

Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?digin	-

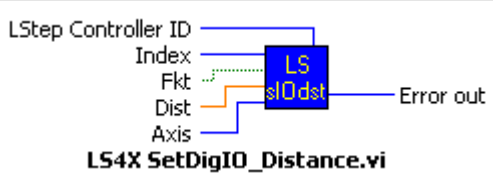
LS_GetDigitalInputsE			
Description:	Reading of additional digital inputs 16 through 31 (not for LSTEPexpress).		
Delphi:	function LS_GetDigitalInputsE(var Value: Integer): Integer; function LSX_GetDigitalInputsE(LSID: Integer; var Value: Integer): Integer;		
C++:	int GetDigitalInputsE(int *plValue);		
LabView:			
Parameter:	Value	Pointer to 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:	LS.GetDigitalInputsE(i);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?edigin	-

LS_SetDigitalOutput			
Description:	Function for setting a digital output.		
Delphi:	function LS_SetDigitalOutput(Index: Integer; Value: LongBool): Integer; function LSX_SetDigitalOutput(LSID: Integer; Index: Integer; Value: LongBool): Integer;		
C++:	int SetDigitalOutput(int lIndex,BOOL Value);		
LabView:			

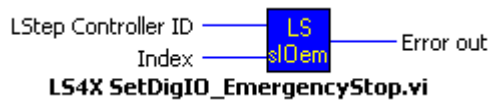
Parameter:	Index	Number of the digital output 0 = Output 0 1 = Output 1 …	
	Value	Set status to “0” or “1” True = Set output to 1 False = Set output to 0	
Example:	LS.SetDigitalOutput(0, true); // Set output pin 0 to "1"		
Other:	Compatibility	“SendString” command	Activation (LSTEPexpress series)
	3	!digout	-

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

LS_SetDigitalOutputsE			
Description:	Function for simultaneously setting the digital outputs 16 through 31 (not for LSTEPexpress).		
Delphi:	function LS_SetDigitalOutputsE(Value: Integer): Integer; function LSX_SetDigitalOutputsE(LSID: Integer; Value: Integer): Integer;		
C++:	int SetDigitalOutputsE(int lValue);		
LabView:			
Parameter:	Value	Bit mask for setting the outputs Bit 0 = Output 16 Bit 1 = Output 17 … Value 0 = Set output to 0 Value 1 = Set output to 1	
Example:	LS.SetDigitalOutputsE($03); // Set outputs 16 and 17 to 1, the remaining ones to 0		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!edigout	-

LS_SetDigIO_Distance			
Description:	Function for activating an output dependent on the set distance before/behind of the target position (not for LSTEPexpress).		
Delphi:	function LS_SetDigIO_Distance(Index: Integer; Fkt: LongBool; Dist: Double; Axis: Integer): Integer; function LSX_SetDigIO_Distance(LSID: Integer; Index: Integer; Fkt: LongBool; Dist: Double; Axis: Integer): Integer;		
C++:	int SetDigIO_Distance(int lIndex,BOOL Fkt,double dDist,int lAxis);		
LabView:			

Parameter:	Index	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:	LS.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 (LSTEPexpress series)
	1	!digfkt / !edigfkt	-

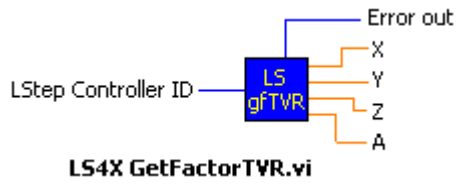
LS_SetDigIO_EmergencyStop			
Description:	Function for allocating a digital input as emergency-stop pin (not for LSTEPexpress).		
Delphi:	function LS_SetDigIO_EmergencyStop(Index: Integer): Integer; function LSX_SetDigIO_EmergencyStop(LSID: Integer; Index: Integer): Integer;		
C++:	int SetDigIO_EmergencyStop(int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

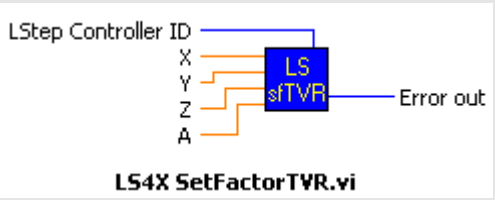
LS_SetDigIO_Off			
Description:	Deactivation of the function of the digital inputs/outputs. (No impact on inputs/outputs). (Not for LSTEPexpress).		
Delphi:	function LS_SetDigIO_Off(Index: Integer): Integer; function LSX_SetDigIO_Off(LSID: Integer; Index: Integer): Integer;		
C++:	int SetDigIO_Off(int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

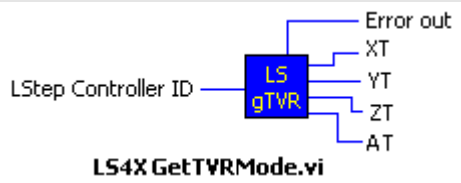
LS_SetDigIO_Polarity	
Description:	Setting of the polarity for the different functions of digital inputs/outputs (not for LSTEPexpress).
Delphi:	function LS_SetDigIO_Polarity(Index: Integer; High: LongBool): Integer; function LSX_SetDigIO_Polarity(LSID: Integer; Index: Integer; High: LongBool): Integer;
C++:	int SetDigIO_Polarity(int lIndex,BOOL High);
LabView:	

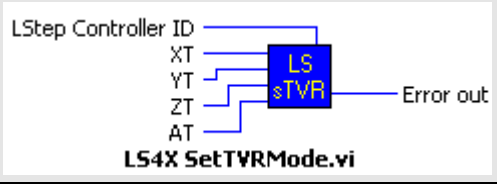
Parameter:	Index	Number of digital input whose polarity is to be changed. 0 = Input 0 1 = Input 1 …	
	High	Polarity setting True = High-active False = Low-active	
Example:	LS.SetDigIO_Polarity(3, True); // Input-/Outputpin 3 High-Aktiv		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!digfkt / !edigfkt	-

4.2.13 Cycle Forward / Back In

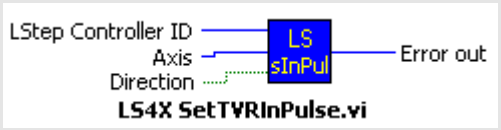
LS_GetFactorTVR			
Description:	Function for reading the cycle forward/back factor.		
Delphi:	function LS_GetFactorTVR(var X, Y, Z, A: Double): Integer; function LSX_GetFactorTVR(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetFactorTVR(double *pdX, double *pdY, double *pdZ, double *pdR);		
LabView:			
Parameter:	X, Y, Z, A	Set cycle forward/back factor in motor increments/cycle	
Example:	LS.GetFactorTVR(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?tvrf	-

LS_SetFactorTVR			
Description:	Function for setting the cycle forward/back factor.		
Delphi:	function LS_SetFactorTVR(X, Y, Z, A: Double): Integer; function LSX_SetFactorTVR(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetFactorTVR(double dX,double dY,double dZ,double dA);		
LabView:			
Parameter:	X, Y, Z, A	Cycle forward/back factor in motor increments/cycle to be set	
Example:	LS.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 (LSTEPexpress series)
	3	!tvrf	tvr

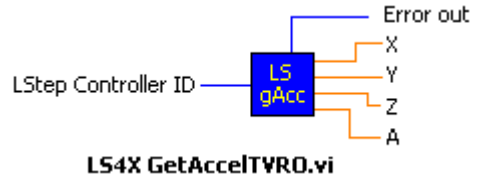
LS_GetTVRMode			
Description:	Reads the set cycle forward/back mode.		
Delphi:	function LS_GetTVRMode(var XT, YT, ZT, AT: Integer): Integer; function LSX_GetTVRMode(LSID: Integer; var XT, YT, ZT, AT: Integer): Integer;		
C++:	int GetTVRMode(int *plXT, int *plYT, int *plZT, int *plAT);		
LabView:			
Parameter:	XT, YT, ZT, AT	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.	
Example:	LS.GetTVRMode(&XT, &YT, &ZT, &AT);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?tvr	-

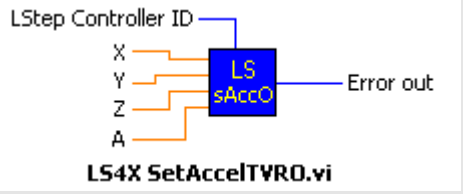
LS_SetTVRMode	
Description:	Function for setting the cycle forward/back mode.
Delphi:	function LS_SetTVRMode(XT, YT, ZT, AT: Integer): Integer; function LSX_SetTVRMode(LSID: Integer; XT, YT, ZT, AT: Integer): Integer;
C++:	int SetTVRMode(int lXT, int lYT,int lZT,int lAT);
LabView:	

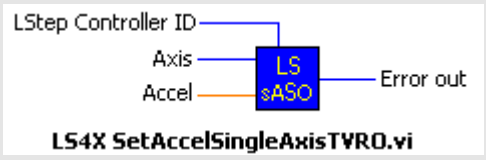
Parameter:	XT, YT, ZT, AT	Cycle forward/back mode to be set 0 = Pulse Forward/Back is "OFF" 1 = Normal cycle Forward/Back processing 2 = Cycle Forward/Back operates with a factor 3 = Cycle Forward/Back processing requires external enabling by the triggerout pin (MFP). 4 = Combination of 2 & 3.	
Example:	LS.SetTVRMode(1, 1, 0, 0); // TVR X and Y-axis ON		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!tvr	-

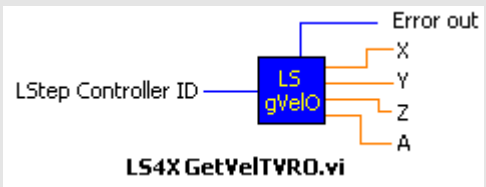
LS_SetTVRInPulse			
Description:	This function can be used to control the cycle forward/back function by software (not for LSTEPexpress).		
Delphi:	function LS_SetTVRInPulse(Axis: Integer; Direction: Boolean): Integer; function LSX_SetTVRInPulse(LSID: Integer; Axis: Integer; Direction: Boolean): Integer;		
C++:	int SetTVRInPulse(int Axis, BOOL Direction);		
LabView:			
Parameter:	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:	LS.SetTVRInPulse (2, true); // 1 cycle Forward on y-Axis.		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!px / !nx !py / !ny …	-

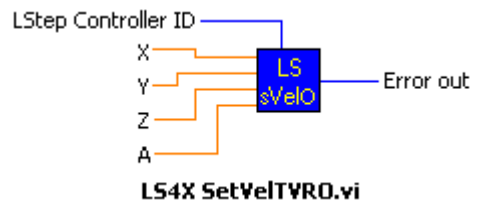
4.2.14 Cycle Forward/Back outputs for additional axes

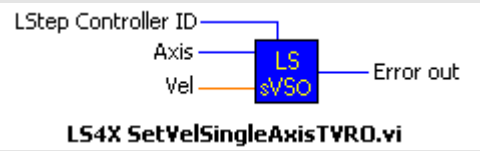
LS_GetAccelTVRO			
Description:	Reads the set accelerations for additional cycle forward/back output axes (not for LSTEPexpress).		
Delphi:	function LS_GetAccelTVRO(var X, Y, Z, A: Double): Integer; function LSX_GetAccelTVRO(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetAccelTVRO(double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

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

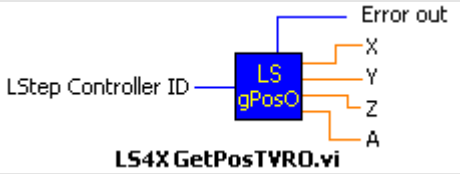
LS_SetAccelSingleAxisTVRO			
Description:	Function for setting the acceleration for a single TVRO axis (not for LSTEPexpress).		
Delphi:	function LS_SetAccelSingleAxisTVRO(Axis: Integer; Accel: Double): Integer; function LSX_SetAccelSingleAxisTVRO(LSID: Integer; Axis: Integer; Accel: Double): Integer;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

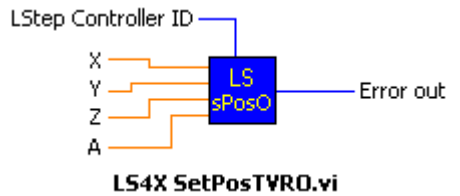
LS_GetVelTVRO			
Description:	Reads the set velocities for the TVRO axes from the controller (not for LSTEPexpress).		
Delphi:	function LS_GetVelTVRO(var X, Y, Z, A: Double): Integer; function LSX_GetVelTVRO(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetVelTVRO(double *pdX, double *pdY,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

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

LS_SetVelSingleAxisTVRO	
Description:	Function for setting the velocity for a single TVRO axis (not for LSTEPexpress).
Delphi:	function LS_SetVelSingleAxisTVRO(Axis: Integer; Vel: Double): Integer; function LSX_SetVelSingleAxisTVRO(LSID: Integer; Axis: Integer; Vel: Double): Integer;
C++:	int SetVelSingleAxisTVRO(int lAxis, double dVel);
LabView:	

Parameter:	Axis	Axis whose acceleration is to be set TVRO X = 1 TVRO Y = 2 …	
	Vel	Velocity value to be set in rps	
Example:	LS.SetVelSingleAxis(1, 10.0); // The X-axis should run with a max. velocity 10 rp/s		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!tvrov	-

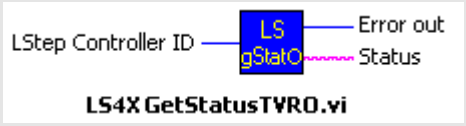
LS_GetPosTVRO			
Description:	Function for reading the position of the additional TVRO axes (not for LSTEPexpress).		
Delphi:	function LS_GetPosTVRO(var dX, dY, dZ, dA: Double): Integer; function LSX_GetPosTVRO(LSID: Integer; var dX, dY, dZ, dA: Double): Integer;		
C++:	int GetPosTVRO (double *pdX, double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

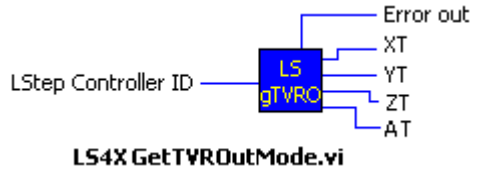
LS_SetPosTVRO			
Description:	Setting of position of the additional TVRO axes (not for LSTEPexpress).		
Delphi:	function LS_SetPosTVRO(dX, dY, dZ, dA: Double): Integer; function LSX_SetPosTVRO(LSID: Integer; dX, dY, dZ, dA: Double): Integer;		
C++:	int SetPosTVRO(double dX, double dY, double dZ, double dA);		
LabView:			

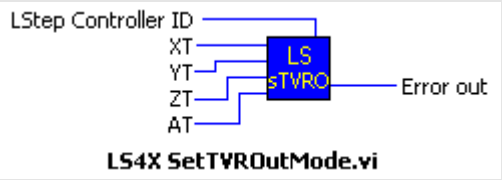
Parameter:	dX, dY, dZ, dA	Position value dependent on the set dimension	
Example:	LS.SetPosTVRO(10.0, 5.0, 0.0, 0.0);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!tvropos	-

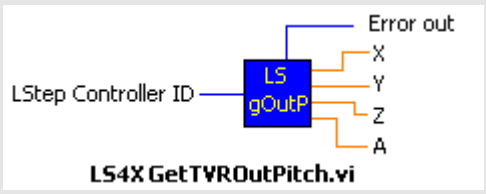
LS_GetStatusTVRO, LS_GetStatusTVROW			
Description:	Delivers the current status of the additional TVRO axes (not for LSTEPexpress).		

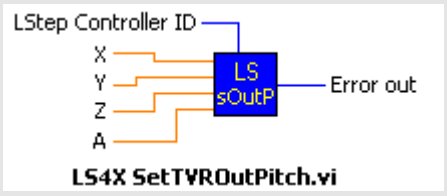
Delphi:	function LS_GetStatusTVRO(pcStat: PAnsiChar; MaxLen: Integer): Integer;
	function LS_GetStatusTVROW(pcStat: PWideChar; MaxLen: Integer): Integer;
	function LSX_GetStatusTVRO(LSID: Integer; pcStat: PAnsiChar; MaxLen: Integer): Integer;
	function LSX_GetStatusTVRO(LSID: Integer; pcStat: PWideChar; MaxLen: Integer): Integer;		

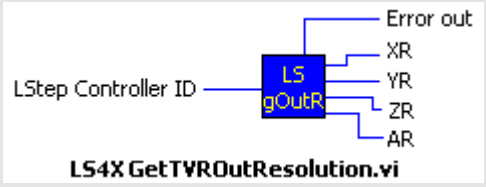
C++:	int GetStatusTVRO(char *pcStat, int lMaxLen); int GetStatusTVROW(TCHAR *pcStat, int lMaxLen);		
LabView:			
Parameter:	pcStat	Pointer to a buffer to which the status string is returned. The axis mask contains one of the following characters: @ = Axis is at standstill M = Axis is moving (Motion) - = Axis is not enabled	
	MaxLen	Maximum number of characters which may be copied into the buffer.	
Example:	LS.GetStatusTVRO(pcStat, 256); // Move additional Z-axis by 5mm in positive direction		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?tvrostatus	-

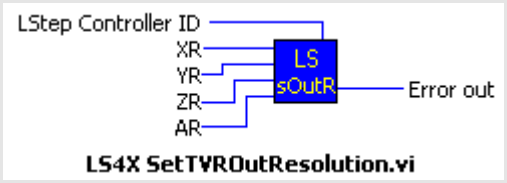
LS_GetTVROutMode			
Description:	Function for reading the set mode from cycle forward/back output (not for LSTEPexpress).		
Delphi:	function LS_GetTVROutMode(var X, Y, Z, A: Integer): Integer; function LSX_GetTVROutMode(LSID: Integer; var X, Y, Z, A: Integer): Integer;		
C++:	int GetTVROutMode(int *plXT, int *plYT, int *plZT, int *plAT);		
LabView:			
Parameter:	X, Y, Z, A	Set TVRO mode 0 = Cycle Forw/Back is “OFF” 1 = Cycle Forw/Back is “ON”	
Example:	LS.GetTVROutMode(&X, &Y, &Z, &A);		
Other:	Compatibility	“SendString” command	Activation (LSTEPexpress series)
	1	?tvrout	-

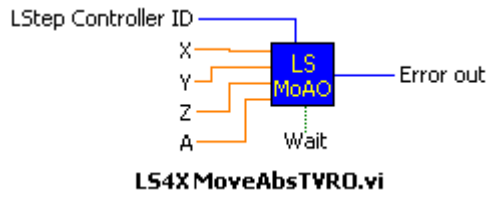
LS_SetTVROutMode			
Description:	Function for setting the set mode of the cycle forward/back output (not for LSTEPexpress).		
Delphi:	function LS_SetTVROutMode(X, Y, Z, A: Integer): Integer; function LSX_SetTVROutMode(LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetTVROutMode(int lXT, int lYT, int lZT, int lAT);		
LabView:			
Parameter:	X, Y, Z, A	TVRO mode to be set 0 = Cycle Forw/Back is “OFF” 1 = Cycle Forw/Back is “ON”	
Example:	LS.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 (LSTEPexpress series)
	1	!tvrout	-

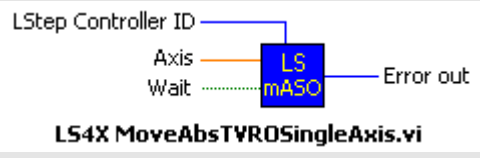
LS_GetTVROutPitch			
Description:	Reads the spindle pitch of the additional TVRO axes (not for LSTEPexpress).		
Delphi:	function LS_GetTVROutPitch(var X, Y, Z, A: Double): Integer; function LSX_GetTVROutPitch (LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetTVROutPitch (double *pdX, double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

LS_SetTVROutPitch			
Description:	Sets the spindle pitches for the additional axes (not for LSTEPexpress).		
Delphi:	function LS_SetTVROutPitch(X, Y, Z, A: Double): Integer; 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:	LS.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 1mm are used*/		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!tvropitch	-

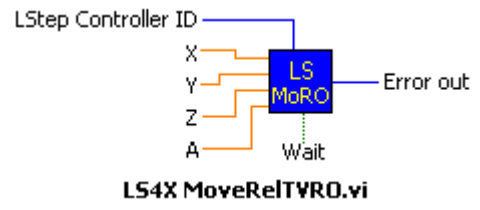
LS_GetTVROutResolution			
Description:	Reads the resolution set in the LSTEP for the power amplifier to be controlled (not for LSTEPexpress).		
Delphi:	function LS_GetTVROutResolution(var X, Y, Z, A: Integer): Integer; function LSX_GetTVROutResolution(LSID: Integer; var X, Y, Z, A: Integer): Integer;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

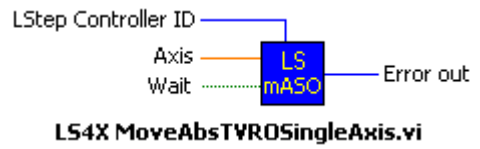
LS_SetTVROutResolution			
Description:	Writes the resolution for controlling the external power amplifier into the LSTEP controller (not for LSTEPexpress).		
Delphi:	function LS_SetTVROutResolution(X, Y, Z, A: Integer): Integer; function LSX_SetTVROutResolution (LSID: Integer; X, Y, Z, A: Integer): Integer;		
C++:	int SetTVROutResolution(int lX, int lY, int lZ, int lA);		
LabView:			
Parameter:	X, Y, Z, A	Resolution to be set in impulses/revolution	
Example:	LS.SetTVROutResolution(1000, 1000, 0, 0); /* A resolution of 1000 impulses per revolution is set for axes X and Y*/		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!tvrores	-

LS_MoveAbsTVRO			
Description:	Approaching the absolute position with TVRO axes from the current position (not for LSTEPexpress).		
Delphi:	function LS_MoveAbsTVRO(X, Y, Z, A: Double; Wait: LongBool): Integer; function LSX_MoveAbsTVRO(LSID: Integer; X, Y, Z, A: Double; Wait: LongBool): Integer;		
C++:	int MoveAbsTVRO(double dX, double dY, double dZ, double dA, BOOL bWait);		
LabView:			
Parameter:	X, Y, Z, A	Absolute position indication in the set axis dimension	
	Wait	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	
Example:	LS.MoveAbsTVRO(10.0, 10.0, 10.0, 10.0, true);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!tvromoa	-

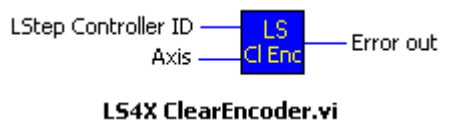
LS_MoveAbsTVROSingleAxis			
Description:	Absolute positioning of single TVRO axis (not for LSTEPexpress).		
Delphi:	function LS_MoveAbsTVROSingleAxis(Axis: Integer; Value: Double; Wait: LongBool): Integer; function LSX_MoveAbsTVROSingleAxis(LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;		
C++:	int MoveAbsTVROSingleAxis(int lAxis, double dValue, BOOL bWait);		
LabView:			

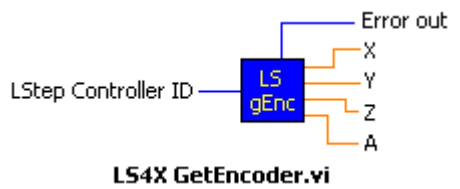
Parameter:	Axis	Axis to be moved 1 = X-axis 2 = Y-axis …	
	Value	Position (input depends on the set dimension)	
Example:	LS.MoveAbsTVROSingleAxis(2, 10.0); //Position additional Y-axis to 10mm absolute		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!tvromoa	-

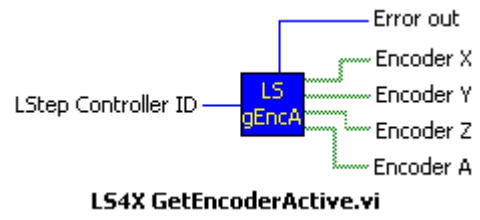
LS_MoveRelTVRO			
Description:	Function for moving a relative vector from the current position (not for LSTEPexpress).		
Delphi:	function LS_MoveRelTVRO(X, Y, Z, A: Double; Wait: LongBool): Integer; function LSX_MoveRelTVRO(LSID: Integer; X, Y, Z, A: Double; Wait: LongBool): Integer;		
C++:	int MoveRelTVRO(double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

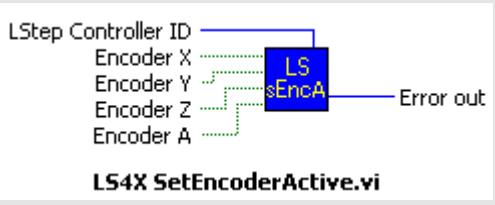
LS_MoveRelTVROSingleAxis			
Description:	Function for relative positioning of a single TVRO axis (not for LSTEPexpress).		
Delphi:	function LS_MoveRelTVROSingleAxis(Axis: Integer; Value: Double; Wait: LongBool): Integer; function LSX_MoveRelTVROSingleAxis(LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;		
C++:	int MoveRelTVROSingleAxis(int lAxis,double dValue,BOOL Wait);		
LabView:			
Parameter:	Axis	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:	LS.MoveRelTVROSingleAxis(3, 5.0); // Move additional Z-axis by 5mm in positive direction		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!tvromor	-

4.2.15 Encoder settings

LS_ClearEncoder			
Description:	Function for setting the encoder counter to zero (not for LSTEPexpress).		
Delphi:	function LS_ClearEncoder(lAxis: Integer): Integer; 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:	LS.ClearEncoder (2); //Set encoder counter of y-axis to zero		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!clearhwcount	-

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

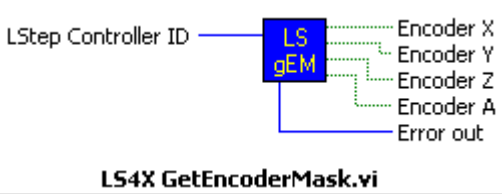
LS_GetEncoderActive			
Description:	Reads which encoders are activated after calibration (not for LSTEPexpress).		
Delphi:	function LS_GetEncoderActive(var Flags: Integer): Integer; function LSX_GetEncoderActive(LSID: Integer; var Flags: Integer): Integer;		
C++:	int GetEncoderActive(int *plFlags);		
LabView:			
Parameter:	Flags	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	
Example:	LS.GetEncoderActive(&Flags);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?encmask	-

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

Parameter:	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 calibration Value 1 = Encoder is to be activated after calibration	

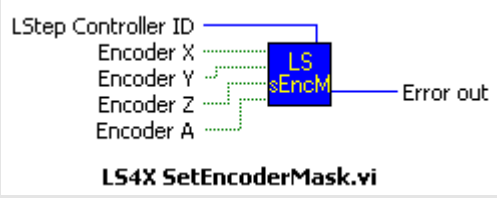
Example:	LS.SetEncoderActive(0);
	// Deactivate all encoders
	LS.SetEncoderMask(2);
	// Activate Y-axis encoder		

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

LS_GetEncoderMask			
Description:	Function for reading the encoder activation.		
Delphi:	function LS_GetEncoderMask(var Flags: Integer): Integer; function LSX_GetEncoderMask(LSID: Integer; var Flags: Integer): Integer;		
C++:	int GetEncoderMask(int *plFlags);		
LabView:			
Parameter:	Flags	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	

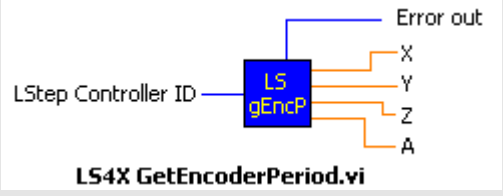
Example:	int EncMask;
	LS.GetEncoderMask(&EncMask);
	if (EncMask & 2) ...
	// If Y-axis encoder is connected + nactive		

Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?enc	-

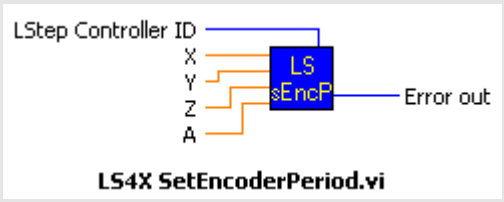
LS_SetEncoderMask			
Description:	Function for activating/deactivating the encoder (not for LSTEPexpress).		
Delphi:	function LS_SetEncoderMask(Value: Integer): Integer; function LSX_SetEncoderMask(LSID: Integer; Value: Integer): Integer;		
C++:	int SetEncoderMask(int lValue);		
LabView:			
Parameter:	Value	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	

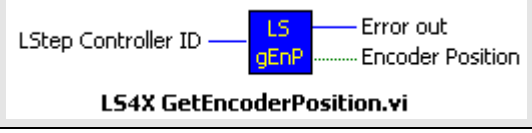
Example:	LS.SetEncoderMask(0);
	// Deactivate all encoders
	LS.SetEncoderMask(2);
	// Activate Y-axis encoder		

Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!enc	-

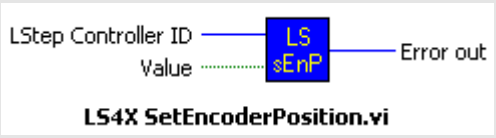
LS_GetEncoderPeriod	
Description:	Function for reading the encoder period lengths.
Delphi:	function LS_GetEncoderPeriod(var X, Y, Z, A: Double): Integer; function LSX_GetEncoderPeriod(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetEncoderPeriod(double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	

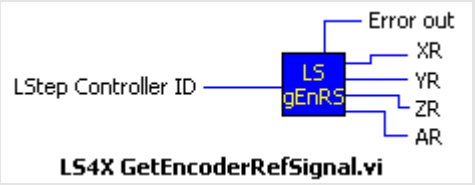
Parameter:	X, Y, Z, A	Set period lengths LSTEP 2000 series = m/s LSTEPexpress series = set dimension/s	
Example:	LS.GetEncoderPeriod(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?encperiod	-

LS_SetEncoderPeriod			
Description:	Function for setting the encoder period lengths.		
Delphi:	function LS_SetEncoderPeriod(X, Y, Z, A: Double): Integer; function LSX_SetEncoderPeriod(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetEncoderPeriod(double dX,double dY,double dZ,double dA);		
LabView:			
Parameter:	X, Y, Z, A	Period lengths to be set LSTEP 2000 series = m/s LSTEPexpress series = set dimension/s	
Example:	LS.SetEncoderPeriod(0.1, 0.1, 0.1, 0.1); // Encoder period length of all axes is 0.1mm		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!encperiod	validconfig

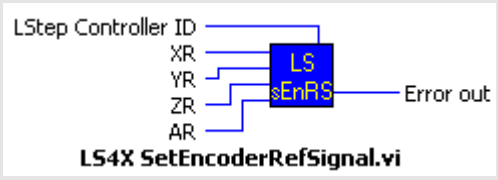
LS_GetEncoderPosition		
Description:	Reading of source data settings of position indication.	
Delphi:	function LS_GetEncoderPosition(Value: Boolean): Integer; function LSX_GetEncoderPosition(LSID: Integer; Value: LongBool): Integer;	
C++:	int GetEncoderPosition(BOOL *pbValue);	
LabView:		

Parameter:	Value	Source of position indication True = Use encoder values for position inquiry False = Do not use encoder values for position inquiry	
Example:	LS.GetEncoderPosition(&Value);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?encpos	-

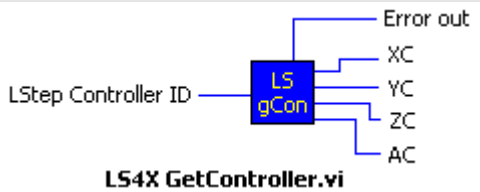
LS_SetEncoderPosition			
Description:	Setting of source data of position indication.		
Delphi:	function LS_SetEncoderPosition(Value: Boolean): Integer; function LSX_SetEncoderPosition(LSID: Integer; Value: LongBool): Integer;		
C++:	int SetEncoderPosition(BOOL fValue);		
LabView:			
Parameter:	Value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

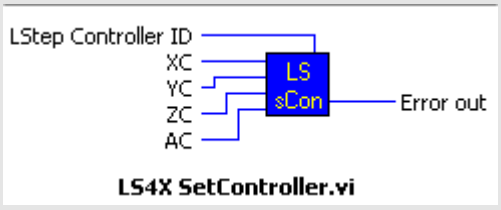
LS_GetEncoderRefSignal			
Description:	Reads whether the reference signal of the encoder is evaluated during calibration.		
Delphi:	function LS_GetEncoderRefSignal(var XR, YR, ZR, AR: Integer): Integer; function LSX_GetEncoderRefSignal(LSID: Integer; var XR, YR, ZR, AR: Integer): Integer;		
C++:	int GetEncoderRefSignal(int *plXR, int *plYR, int *plZR, int *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:	LS.GetEncoderRefSignal(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?encref	-

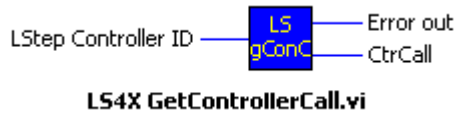
LS_SetEncoderRefSignal			
Description:	Function for activating the evaluation of the reference signal of an encoder.		
Delphi:	function LS_SetEncoderRefSignal(XR, YR, ZR, AR: Integer): Integer; function LSX_SetEncoderRefSignal(LSID: Integer; XR, YR, ZR, AR: Integer): Integer;		
C++:	int SetEncoderRefSignal(int lXR,int lYR,int lZR,int lAR);		
LabView:			
Parameter:	X, Y, Z, A	Set value 1 = The reference signal is evaluated during calibration 0 = The reference signal is not evaluated	
Example:	LS.SetEncoderRefSignal(1, 1, 0, 0); /* The reference signal of the encoders x and y is evaluated during calibration. */		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!encref	-

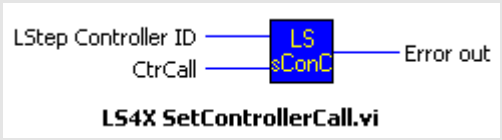
4.2.16 Controller settings

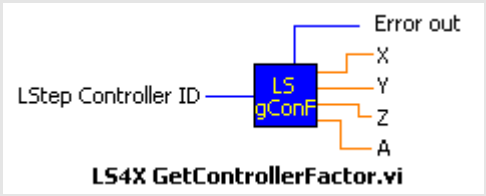
LS_GetController			
Description:	Function for reading the controller mode (not for LSTEPexpress).		
Delphi:	function LS_GetController(var XC, YC, ZC, AC: Integer): Integer; function LSX_GetController(LSID: Integer; var XC, YC, ZC, AC: Integer): Integer;		
C++:	int GetController(int *plXC, int *plYC,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

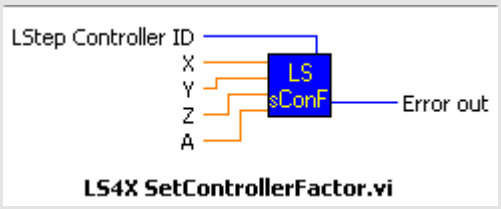
LS_SetController	
Description:	Function for setting the controller mode. (not for LSTEPexpress).
Delphi:	function LS_SetController(XC, YC, ZC, AC: Integer): Integer; function LSX_SetController(LSID: Integer; XC, YC, ZC, AC: Integer): Integer;
C++:	int SetController(int lXC,int lYC,int lZC,int lAC);
LabView:	

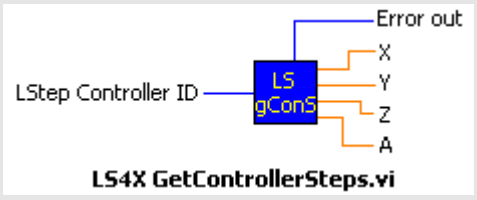
Parameter:	X, Y, Z, A	Set controller mode 0 = Controller "OFF" 1 = Controller "OFF after reaching target position" 2 = Controller "Always ON" 3 = Controller "OFF after reaching target position" with reduced current 4 = Controller "Always ON" with reduced current	
Example:	LS.SetController(1, 2, 0, 0);		
Other:	Compatibility 1	"SendString" command !ctr	Activation (LSTEPexpress series) -

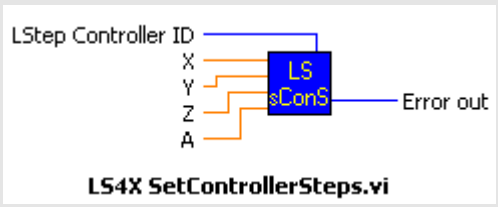
LS_GetControllerCall			
Description:	The function delivers the set controller call time (not for LSTEPexpress).		
Delphi:	function LS_GetControllerCall(var CtrCall: Integer): Integer; function LSX_GetControllerCall(LSID: Integer; var CtrCall: Integer): Integer;		
C++:	int GetControllerCall(int *plCtrCall);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

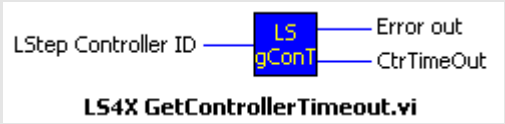
LS_SetControllerCall			
Description:	Function for setting the controller call time (not for LSTEPexpress).		
Delphi:	function LS_SetControllerCall(CtrCall: Integer): Integer; function LSX_SetControllerCall(LSID: Integer; CtrCall: Integer): Integer;		
C++:	int SetControllerCall(int lCtrCall);		
LabView: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

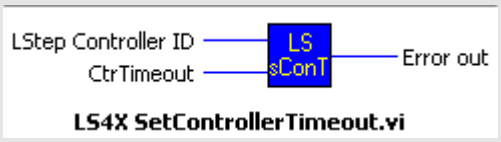
LS_GetControllerFactor			
Description:	Function for reading the controller factors. See documentation of LSTEP 2000 series. (not for LSTEPexpress).		
Delphi:	function LS_GetControllerFactor(var X, Y, Z, A: Double): Integer; function LSX_GetControllerFactor(LSID: Integer; var X, Y, Z, A: Double): Integer;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

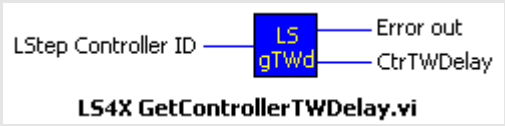
LS_SetControllerFactor			
Description:	Function for setting the controller factors. See documentation of LSTEP 2000 series (not for LSTEPexpress).		
Delphi:	function LS_SetControllerFactor(X, Y, Z, A: Double): Integer; function LSX_SetControllerFactor(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++: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

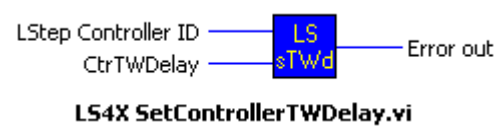
LS_GetControllerSteps			
Description:	This function delivers the set controller step length (not for LSTEPexpress).		
Delphi:	function LS_GetControllerSteps(var X, Y, Z, A: Double): Integer; function LSX_GetControllerSteps(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetControllerSteps(double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

LS_SetControllerSteps			
Description:	Function for setting the controller step length (not for LSTEPexpress).		
Delphi:	function LS_SetControllerSteps(X, Y, Z, A: Double): Integer; function LSX_SetControllerSteps(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetControllerSteps(double dX,double dY,double dZ,double dA);		
LabView:			
Parameter:	X, Y, Z, A	Controller step length in the set axis dimension.	
Example:	LS.SetControllerSteps(4, 5, 7, 9);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!ctrs	-

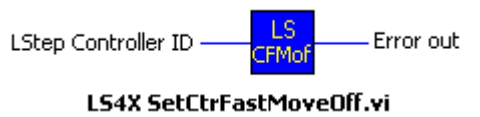
LS_GetControllerTimeout			
Description:	Function for reading the controller timeouts after which a travel command returns with an error message (error code 4013), if the controller could not definitively find a position (not for LSTEPexpress).		
Delphi:	function LS_GetControllerTimeout(var ACtrTimeout: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

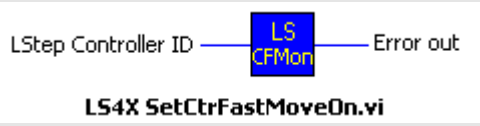
LS_SetControllerTimeout			
Description:	Function for setting the controller timeouts after which a travel command returns with an error message (error code 4013), if the controller could not definitively find a position (not for LSTEPexpress).		
Delphi:	function LS_SetControllerTimeout(ACtrTimeout: Integer):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

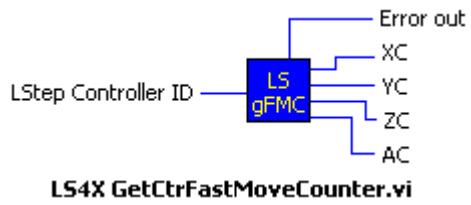
LS_GetControllerTWDelay			
Description:	Reading of controller delay (not for LSTEPexpress).		
Delphi:	function LS_GetControllerTWDelay(var CtrTWDelay: Integer): Integer; function LSX_GetControllerTWDelay(LSID: Integer; var CtrTWDelay: Integer): Integer;		
C++:	int GetControllerTWDelay(int *plCtrTWDelay);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

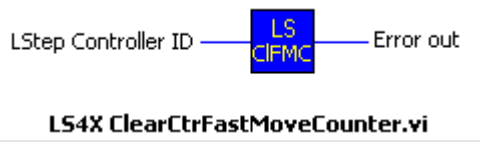
LS_SetControllerTWDelay			
Description:	Function for setting the controller delay (not for LSTEPexpress).		
Delphi:	function LS_SetControllerTWDelay(CtrTWDelay: Integer): Integer; function LSX_SetControllerTWDelay(LSID: Integer; CtrTWDelay: Integer): Integer;		
C++:	int SetControllerTWDelay(int lCtrTWDelay);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

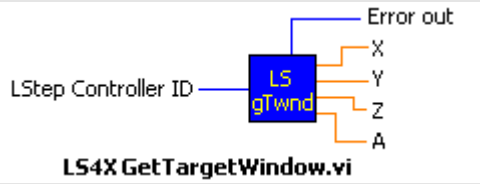
LS_GetCtrFastMove			
Description:	Reads the setting of the Fast Move function (not for LSTEPexpress).		
Delphi:	function LS_GetCtrFastMoveOff(var bActive: LongBool): Integer; function LSX_GetCtrFastMoveOff(LSID: Integer; var bActive: LongBool): Integer;		
C++:	int GetCtrFastMove(BOOL *pbActive);		
LabView: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

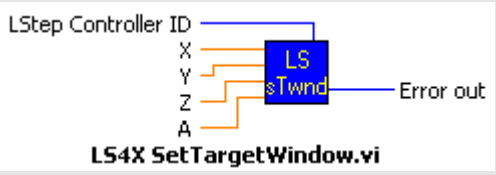
LS_SetCtrFastMoveOff			
Description:	Function for deactivating the Fast Move function (not for LSTEPexpress).		
Delphi:	function LS_SetCtrFastMoveOff: Integer; function LSX_SetCtrFastMoveOff(LSID: Integer): Integer;		
C++:	int SetCtrFastMoveOff();		
LabView:			
Parameter:	-		
Example:	LS.SetCtrFastMoveOff();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

LS_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 LSTEPexpress).		
Delphi:	function LS_SetCtrFastMoveOn: Integer; function LSX_SetCtrFastMoveOn(LSID: Integer): Integer;		
C++:	int SetCtrFastMoveOn();		
LabView:			
Parameter:	-		
Example:	LS.SetCtrFastMoveOn();		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!ctrfm	-

LS_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 LSTEPexpress).		
Delphi:	function LS_GetCtrFastMoveCounter(var XC, YC, ZC, AC: Integer): Integer; 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:	LS.SetCtrFastMoveCounter(&XC, &YC, &ZC, &AC);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	?ctrfmc	-

LS_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 LSTEPexpress).		
Delphi:	function LS_ClearCtrFastMoveCounter: Integer; function LSX_ClearCtrFastMoveCounter(LSID: Integer): Integer;		
C++:	int ClearCtrFastMoveCounter();		
LabView:			
Parameter:			
Example:	LS.ClearCtrFastMoveCounter;		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	1	!ctrfmc	-

LS_GetTargetWindow			
Description:	Reads the target windows of all axes.		
Delphi:	function LS_GetTargetWindow(var X, Y, Z, A: Double): Integer; function LSX_GetTargetWindow(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetTargetWindow(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	X, Y, Z, A	Target window depends on the set axis dimension.	
Example:	LS.GetTargetWindow(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?twi	-

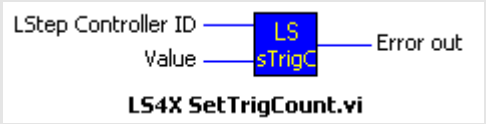
LS_SetTargetWindow			
Description:	Function for setting the target window.		
Delphi:	function LS_SetTargetWindow(X, Y, Z, A: Double): Integer; function LSX_SetTargetWindow(LSID: Integer; X, Y, Z, A: Double): Integer;		
C++:	int SetTargetWindow(double dX,double dY,double dZ,double dA);		
LabView:			
Parameter:	X, Y, Z, A	Target window to be set in the axis dimension.	
Example:	LS.SetTargetWindow(1.0, 0.002, 1.0, 1.0);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!twi	validpar / validconfig

4.2.17 Trigger output

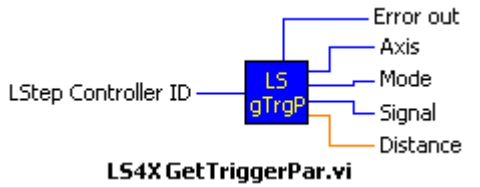
LS_GetTrigger			
Description:	This function delivers the status of the trigger function.		
Delphi:	function LS_GetTrigger(var ATrigger: LongBool): Integer; function LSX_GetTrigger(LSID: Integer; var ATrigger: LongBool): Integer;		
C++:	int GetTrigger(BOOL *pbATrigger);		
LabView:			
Parameter:	ATrigger	Status of trigger function True = Trigger "On" False = Trigger "Off"	
Example:	LS.GetTrigger(&ATrigger);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?trig	-

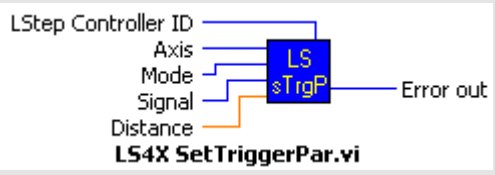
LS_SetTrigger			
Description:	Function for activating/deactivating the trigger function.		
Delphi:	function LS_SetTrigger(ATrigger: LongBool): Integer; function LSX_SetTrigger(LSID: Integer; ATrigger: LongBool): Integer;		
C++:	int SetTrigger(BOOL bATrigger);		
LabView:			
Parameter:	ATrigger	Status of trigger function True = Trigger "On" False = Trigger "Off"	
Example:	LS.SetTrigger(true);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!trig	-

LS_GetTrigCount			
Description:	Function for reading the trigger counter.		
Delphi:	function LS_GetTrigCount(var Value: Integer): Integer; function LSX_GetTrigCount(LSID: Integer; var Value: Integer): Integer;		
C++:	int GetTrigCount(int *pValue);		
LabView:			
Parameter:	Value	Number of executed trigger events	
Example:	LS.GetTrigCount(&Value);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?trigcount	-

LS_SetTrigCount			
Description:	Function for setting the trigger counter reading.		
Delphi:	Function LS_SetTrigCount(Value: Integer): Integer; function LSX_SetTrigCount(LSID: Integer; Value: Integer): Integer;		
C++:	int SetTrigCount(int Wert);		
LabView:			
Parameter:	Value	Intended value of trigger counter	
Example:	LS.SetTrigCount(0);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!trigcount	-

LS_GetTriggerPar	
Description:	Delivers the set parameters of the trigger function.
Delphi:	function LS_GetTriggerPar(var Axis, Mode, Signal: Integer; var Distance: Double): Integer; function LSX_GetTriggerPar(LSID: Integer; var Axis, Mode, Signal: Integer; var Distance: Double): Integer;
C++:	int GetTriggerPar(int *plAxis, int *plMode, int *plSignal, double *pdDistance);

LabView:			
Parameter:	Axis	Axis 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:	LS.GetTriggerPar(&Axis, & Mode, & Signal, & Distance);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?triga / ?trigm ?trigs / ?trigd	-

LS_SetTriggerPar	
Description:	Sets the trigger parameters for the trigger function
Delphi:	function LS_SetTriggerPar(Axis, Mode, Signal: Integer; Distance: Double): Integer; function LSX_SetTriggerPar(LSID: Integer; Axis, Mode, Signal: Integer; Distance: Double): Integer;
C++:	int SetTriggerPar(int lAxis, int lMode, int lSignal, double dDistance);
LabView:	

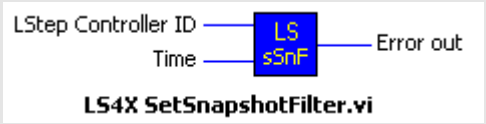
<table>
<tr><td rowspan="4">Parameter:</td><td>Axis</td><td colspan="3">Axis allocation of trigger function
X = 1
Y = 2
…</td></tr>
<tr><td>Mode</td><td colspan="3">Trigger mode (see controller manual, trigm command)</td></tr>
<tr><td>Signal</td><td colspan="3">Trigger signal length in µs</td></tr>
<tr><td>Distance</td><td colspan="3">Trigger distance in the set unit of the axis</td></tr>
<tr><td>Example:</td><td colspan="4">LS.SetTriggerPar(1, 3, 2, 5.0);</td></tr>
<tr><td rowspan="2">Other:</td><td>Compatibility</td><td colspan="2">“SendString” command</td><td>Activation (LSTEPexpress series)</td></tr>
<tr><td>3</td><td colspan="2">!triga / !trigm
!trigs / !trigd</td><td>trig</td></tr>
</table>

4.2.18 Snapshot input

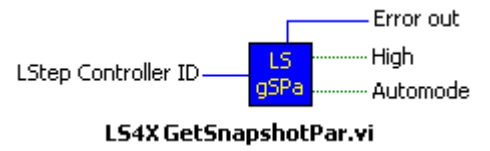
LS_GetSnapshot			
Description:	Shows the current snapshot function status.		
Delphi:	function LS_GetSnapshot(var ASnapshot: LongBool): Integer; function LSX_GetSnapshot(LSID: Integer; var ASnapshot: LongBool): Integer;		
C++:	int GetSnapshot(BOOL *pbASnapshot);		
LabView:			
Parameter:	ASnapshot	Status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

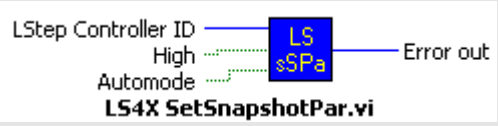
LS_SetSnapshot			
Description:	Function for activating/deactivating the snapshot function.		
Delphi:	function LS_SetSnapshot(ASnapshot: LongBool): Integer; function LSX_SetSnapshot(LSID: Integer; ASnapshot: LongBool): Integer;		
C++:	int SetSnapshot (BOOL bASnapshot);		
LabView:			
Parameter:	ASnapshot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

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

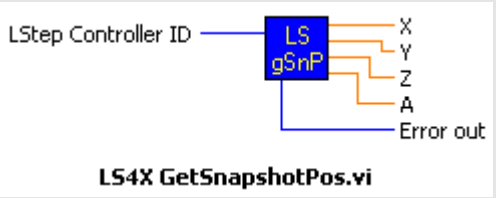
LS_SetSnapshotFilter			
Description:	Function for setting the input filter of the snapshot function for bouncing switches.		
Delphi:	function LS_SetSnapshotFilter(lTime: Integer): Integer; function LSX_SetSnapshotFilter(LSID: Integer; lTime: Integer): Integer;		
C++:	int SetSnapshotFilter(int lTime);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

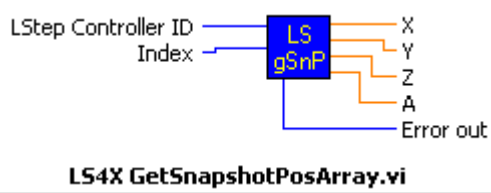
LS_GetSnapshotPar	
Description:	Function for reading the snapshot parameters.
Delphi:	function LS_GetSnapshotPar(var High, AutoMode: LongBool): Integer; function LSX_GetSnapshotPar(LSID: Integer; var High, AutoMode: LongBool): Integer;
C++:	int GetSnapshotPar(BOOL *pbHigh, BOOL *pbAutoMode);

LabView:			
Parameter:	High	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:	LS.GetSnapshotPar(&High, & AutoMode);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	?snsl / ?snsm	-

LS_SetSnapshotPar			
Description:	Function for setting the snapshot parameters.		
Delphi:	function LS_SetSnapshotPar(High, AutoMode: LongBool): Integer; function LSX_SetSnapshotPar(LSID: Integer; High, AutoMode: LongBool): Integer;		
C++:	int SetSnapshotPar(BOOL bHigh, BOOL bAutoMode);		
LabView:			
Parameter:	High	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:	LS.SetSnapshotPar(true, false);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!snsl / !snsm	sns

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

LS_GetSnapshotPos			
Description:	Function for reading the snapshot position.		
Delphi:	function LS_GetSnapshotPos(var X, Y, Z, A: Double): Integer; function LSX_GetSnapshotPos(LSID: Integer; var X, Y, Z, A: Double): Integer;		
C++:	int GetSnapshotPos(double *pdX, double *pdY, double *pdZ, double *pdA);		
LabView:			
Parameter:	X, Y, Z, A	Snapshot position in the set unit	
Example:	double X, Y, Z, A; LS.GetSnapshotPos(&X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!snsp	-

LS_GetSnapshotPosArray			
Description:	Function for reading the arrays of the snapshot positions.		
Delphi:	function LS_GetSnapshotPosArray(Index: Integer; var X, Y, Z, A: Double): Integer; 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; LS.GetSnapshotPos(2, &X, &Y, &Z, &A);		
Other:	Compatibility	"SendString" command	Activation (LSTEPexpress series)
	3	!snsa	-

5 CallBack functions of 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 of 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 LS_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 of extended CallBack function (OsziCallBackFct)		
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 LS_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 LS_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 LS_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
LSTEPexpress 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 LS_GetStatusAxis, LS_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	
LSTEPexpress series	Yes	Yes	Yes	Yes	Yes	!errorchannel 2
LSTEP 2000 series	-	-	-	-	-	-
Others	-	-	-	-	-	-

Interface type		Supported channel					
No.	Type	1	2	3	4	…	10000
1	RS232	-	-	-	-	-	-
2	ArcNet	-	-	-	-	-	-
3	DPRAM / ISA-Bus	-	-	-	-	-	-
4	DPRAM / PCI-Bus	-	-	-	-	-	-
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 LS_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 controller by means of the API function LS_GetError. It is indicated as a 32-bit integer and has a value below 4000. If the value is 0, no error has occurred since the last inquiry. Apart from the function LS_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 Manual
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
26SECURITY 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

1010=Other manual mode is active
1011=Servo and step motor cannot be coupled (joystick)
1012=Output already allocated to other function (digital output)

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

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

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 0 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

1160=Incorrect dynamic check sum of EEProm
1161=Incorrect static check sum of EEProm
1162=Incorrect EEProm version
1163=Incorrect EEProm structure
1164=Window for computation time exceeded (500/320µs)
1165=Window for computation time exceeded (62.5/40µs)

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

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 are the LSTEP4.DLL or 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 the LSTEP-PCI?
Why does my program with the LSTEP4.DLL not get any connection to the LSTEP-PCI?
A fault has occurred during the process, in the LSTEP4.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 several LSTEP-PCI cards in a PC?
When should LSTEP4.DLL, when LSTEP4X.DLL be used?
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 LStep4.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 in LSTEP?
How many entries will fit into the log window of the LSTEP-API?

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

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

LSTEP4X.DLL is embedded analogously to this procedure.

How do I initialize the connection to LSTEPwith 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 LoadConfig
ConnectEx	When loading the interface parameters out of a data structure

How do I initialise the driver for the LSTEPPCI?

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 not able to establish a connection to the LSTEPPCI?

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 LSTEP4.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 SetProcessMessagesProc 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
LS.MoveAbsSingleAxis(Yaxis, 20, false); // Move the Y-axis asynchronously
LS.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):

```
LS.MoveAbsSingleAxis(Xaxis, 10, false); // Move the X-axis asynchronously
LS.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 that LS_ConnectSimple(4, nil, 0, true) will establish a connection to the LSTEP-PCI in the first PCI slot of the mainboard, LS_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.

When should LSTEP4.DLL, when LSTEP4X.DLL be used?

If several LSTEP controllers/LSTEP-PCI cards are controlled from one PC, this is only possible with LSTEP4X.DLL. Otherwise, LSTEP4.DLL is suitable, with LSTEP4X.DLL being recommended.

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 LStep4.cpp?

This is not a fault in the file LStep4.cpp. The message usually appears, if the compiler is looking for a pre-compiled header file and cannot find it. Should the message "fatal error C1010 precompiled header files" appear in MS Visual C++, the option "pre compiled Header-file" for LStep4.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 LStep4.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.