

# TANGO-DLL

## Documentation



In der Murch 15  
35579 Wetzlar  
Germany  
Tel.: +49/6441/9116-0  
[www.marzhauser.com](http://www.marzhauser.com)

## Table of Contents

1.	Introduction	3
1.1.	Functional Range	3
1.2.	System Requirements	3
1.3.	Supported Development Environments	3
2.	DLL-Interface	4
2.1.	General Information	4
2.2.	Integration in Visual C++	5
2.3.	Integration in Visual Basic	5
2.4.	Integration in LabView	6
3.	General Information of DLL Usage	9
3.1.	Initialization of Controller	10
3.2.	Own Program Section	12
4.	Functions	13
4.1.	Quick Reference	13
4.2.	DLL Configuration / Interface	19
4.3.	Controller Information	23
4.4.	Status Requests	24
4.5.	Settings	27
4.6.	Move Commands and Positioning Management	38
4.7.	Joystick and Handwheel	45
4.8.	Control Console with Trackball and Joyspeed Keys	51
4.9.	Limit Switches (Hardware and Software)	54
4.10.	Digital and Analog Inputs and Outputs	59
4.11.	Encoder Settings	62
4.12.	Closed Loop Settings	67
4.13.	Trigger Output	72
4.14.	Snapshot Input	74
5.	SlideExpress Interface	77
5.1.	Eject	77
5.2.	Insert	77
5.3.	SlideSeated	77
5.4.	MagazinSeated	77
5.5.	GetGripper	78
5.6.	SetGripper	78
5.7.	GetSlide	78
5.8.	PutSlide	78
5.9.	GetPrioHandlerPos	79
5.10.	SetPrioHandlerPos	79
6.	Error Codes	80
6.1.	Tango Error Messages	80
6.2.	DLL Error Messages	80
7.	Document Revision History	82

# 1. Introduction

The Tango-DLL (programming interface for Tango controllers) is designed to help software developers writing applications for 2/4-phase stepper motors fast and effectively without the need of hardware-oriented programming. The Tango-DLL supports all commands of the Tango controller.

## 1.1. Functional Range

- Windows DLL 32-bit and 64-bit
- Supports Tango stepper motor controllers
- Control via RS232, or Virtual COM Port (PCI, USB and PCIe)
- Supports all controller commands
- Up to 4 Axes per Tango
- Up to 8 Tango controllers

## 1.2. System Requirements

The Tango-DLL can be used on all Windows PC's from Windows 98 to Windows 8.

## 1.3. Supported Development Environments

The Tango-DLL is available as 32 Bit and 64 Bit version.

It has been tested on operating systems Windows XP, Windows 7 and Windows 8 using following development tools:

Microsoft Visual Studio 2010 languages Visual Basic, C# and C++  
National Instruments LabVIEW  
Embarcadero Delphi 2007 and Delphi XE

Compatibility is assumed for all other programming environments which are able to use DLL.

(DLL = Dynamic Link Library, generally means a dynamic library. In programming, a software library is a collection of program functions for tasks belonging together. Other than programs, libraries are not independently operating units, but auxiliary modules, which are made available to programs.)

## 2. DLL-Interface

Main part of the Tango DLL is the data file Tango\_DLL.dll. Use this file for developing own programs to configure Tango, send commands, retrieve the status of inputs or outputs etc.

### 2.1. General Information

All functions are declared with a 32-bit Integer return value. A return value of 0 (zero) indicates the error free execution of the function. In case of errors (e.g. Timeouts), the corresponding error code (**see error codes**) is returned.

The examples provided in this documentation exclusively use „LSX\_“ commands in which the first value stands for the Tango ID (LSID). This ID is needed to address a variety of controllers simultaneously. As the "LSX\_" commands currently only support one controller, we recommend using the "LS\_" commands. With this, the first value of the Tango-ID is not needed in function calls, neither is a CreateLSID required.

#### Example

##### „LS\_“-Command:

```
pTango->MoveAbs(50.0, 50.0, 50.0, 10.0, TRUE);
```

##### „LSX\_“-Command:

```
pTango->MoveAbs(1, 50.0, 50.0, 50.0, 10.0, TRUE);
```

*// the first value is the LSID, which is not needed with „LS\_“ commands*

With functions such as LSX\_MoveAbs, values of 4 axes have to be passed to the function. If the controller has only 1-3 axes, values of the not available axes are ignored; they can be set to 0.

## 2.2. Integration in Visual C++

An enclosure of Tango\_DLL.dll has been created for Visual C++. The class CTango loads the DLL and all pointers on function calls dynamically. There is no „LS\_“ or „LSX\_“ prefix in the function names of the Tango object.

(Example pTango->Calibrate() instead of LS\_Calibrate).

Only one instance should be created of the class CTango, as with Tango-DLL, momentarily, it is not possible to operate several controllers at the same time.

The required files for your C/C++ Application Tango.h and Tango.cpp can be found on the CD in the directory Software\API\Examples\Visual\_C\SourceCode.

Required files: Tango\_DLL.dll, Tango.h and Tango.cpp

Visual C++ example for controlling a Tango:

```
...  
pTango = new CTango();  
...  
  
pTango->ConnectSimple(1, „COM3“, 57600, TRUE);  
pTango->MoveAbs(30, 50, 70, 0, TRUE);  
pTango->Disconnect();  
delete pTango;
```

## 2.3. Integration in Visual Basic

In order to use the functions of Tango-DLL, the file Tango.vb must be added to the project.

The file Tango.vb can be found on the CD in directory Software\API\Examples\Visual\_Basic\SourceCode.

Required files: Tango\_DLL.dll and Tango.vb

Visual Basic example for controlling a Tango:

```
Dim return value As Integer  
Dim return value2 As Integer  
Dim return value3 As Integer  
  
...  
Return value = LS_ConnectSimple(1, „COM3“, 57600, 1)  
Return value2 = LS_MoveAbs(30, 50, 70, 0, 1)  
Return value3 = LS_Disconnect
```

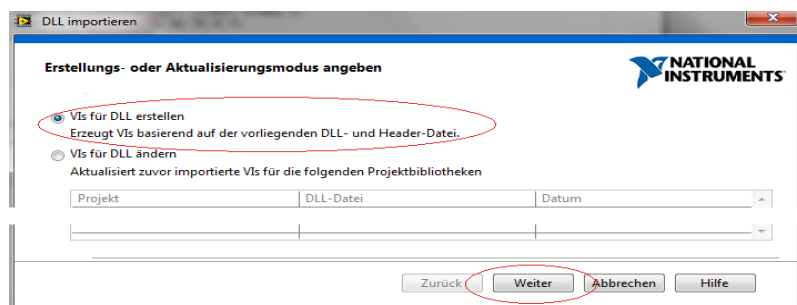
## 2.4. Integration in LabView

This DLL-import description can be used with every LabView Version, which supports DLL-import functionality.

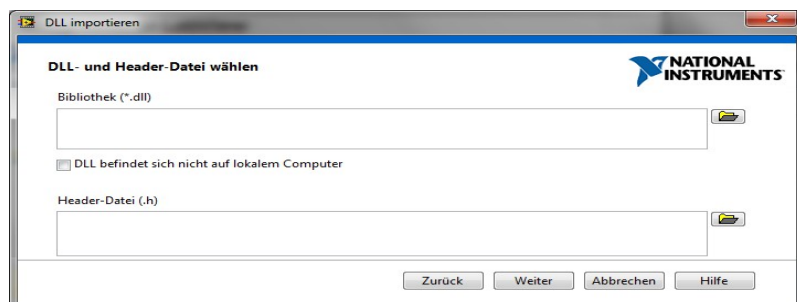
In order to use the functions of TANGO-DLL with LabView, the TANGO-DLL has to be imported to LabView.

Therefore follow the steps listed below:

- 1) Start LabView
- 2) In LabView window: Tools → Import → DLL
- 3) Select the first radio button and press next

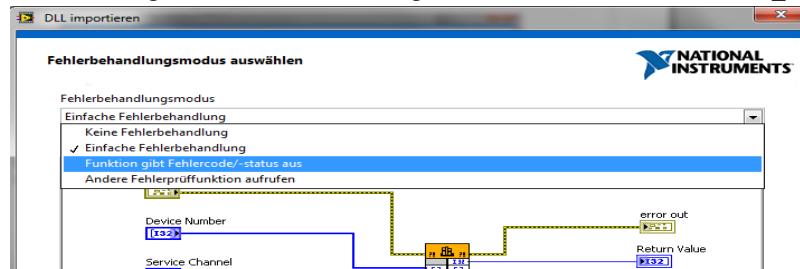


- 4) Select the TANGO\_DLL.dll and the TANGOLSX\_API.h, which can be found on the CD in directory Software/API&DLL/LabView, in the corresponding fields.



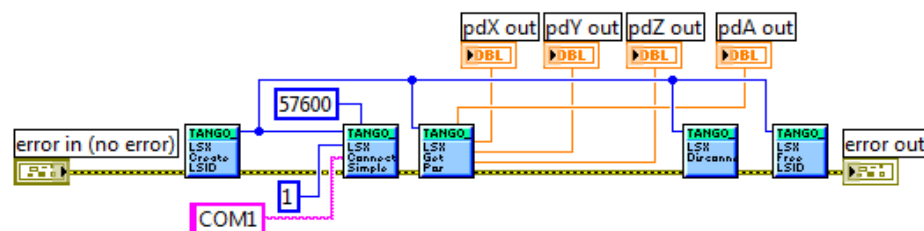
- 5) "Including Paths" in the next window need not to be configured
- 6) In the next window the included functions of the TANGO\_DLL.dll are listed and selectable. It is recommended to select all functions. You may notice, that only half of the functions included in TANGO\_DLL.dll are found in the TANGOLSX\_API.h which is correct, because all functions in TANGO\_DLL.dll exist in "LS\_function" and "LSX\_function" notation. The TANGOLSX\_API.h defines just the "LSX" functions, which should be preferred to use anyway.

- 7) After selecting the path and name for the project library the error handling mode should at least contain a simple error handling or even a error handling with return function of TANGO\_DLL.dll included.



- 8) The configuration of the VIs should not be changed and the import process can start

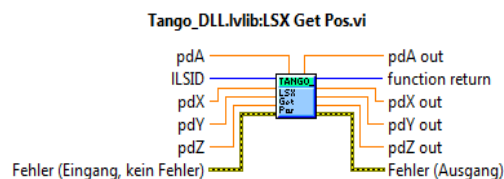
LabView starting example for controlling a TANGO:



This example creates a TANGO-ID number to select the TANGO, which is addressed for the command. A connection to the TANGO is established with virtual COM-Port 1 and Baud-Rate 57600. The actual position of all axes is read out and the TANGO is disconnected. Last step is to free the created TANGO-ID number.

#### Remarks:

“Get” functions defined in TANGO\_DLL.dll often have pointer as parameters. This pointer are displayed as inputs and outputs in LabView VIs because LabView is not able to detect whether this pointer is needed as input or output.



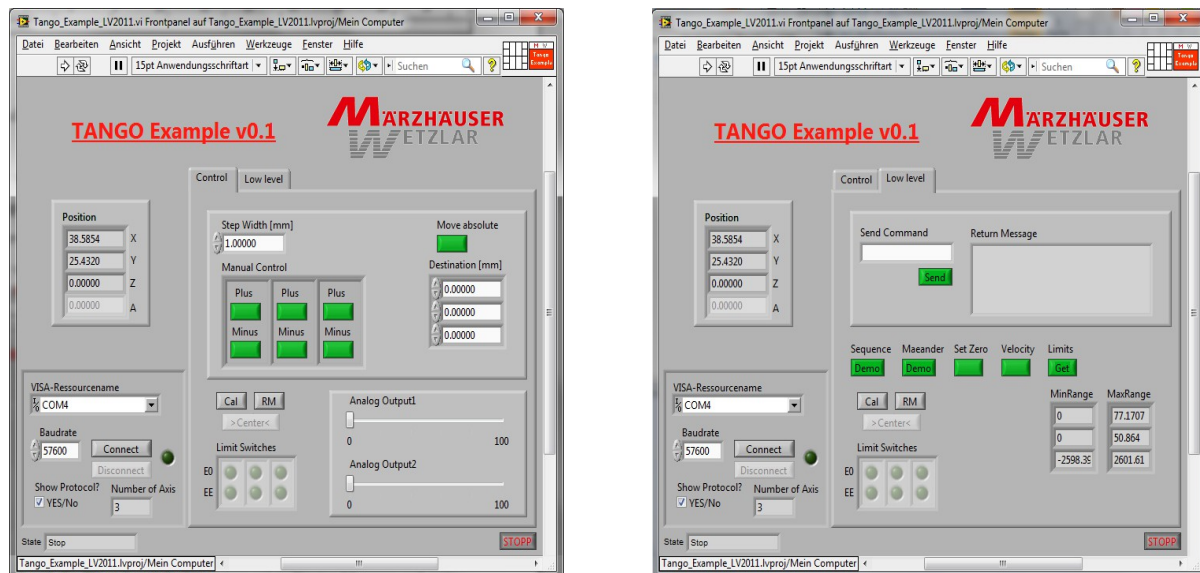
```
TANGOAPI int TANGOCALL LSX_GetPos (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);
```

It is needless to connect the input parameters in such a VI because it will be ignored anyway and won't have any effect.

## Program Example:

Required LabView-Version: LabView 2011 and newer

An example program of controlling a TANGO via LabView can be found on CD in directory Software/API&DLL/LabView/TANGO\_Example\_LV2011. This example is implemented in LV2011 and is not compatible with elder versions. It gives an overview of how the TANGO\_DLL.dll can be used with LabView and how the TANGO can be controlled with a LabView environment.



This example VI looks for a TANGO (connected with the PC and switched to power on) in Device Manager and writes the corresponding COM-Port in VISA-Ressourcenname as a pre-selection. The default baud-rate is 57600. After selecting the correct COM-Port the user is able to connect to TANGO. The program gives you an overview over the actual position of all active axes, the values for analog outputs and if a limit switch is active or not (limit switches can only be active, as long as no calibration and range measure drive has been performed).

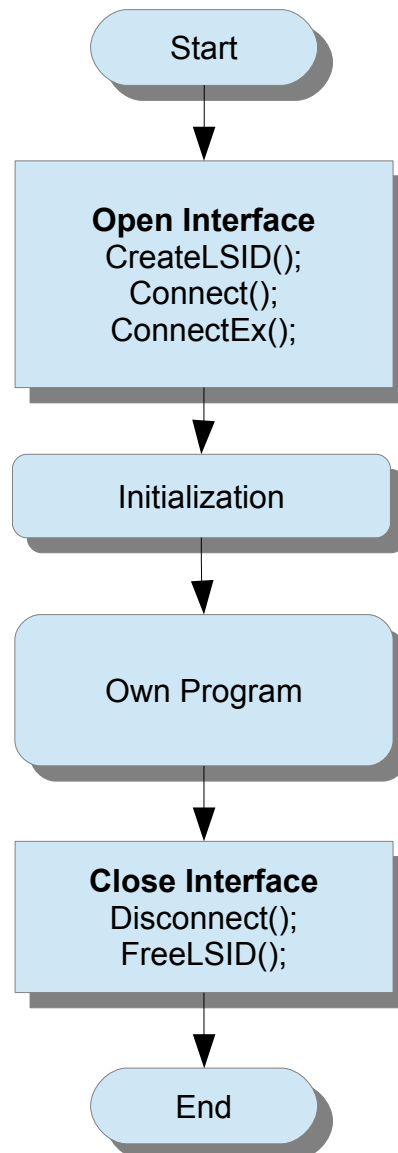
Functions included in TANGO example VI:

- Calibrate (looks for the backward limit switches)
- Range Measure (looks for the forward limit switches)
- Center Drive (Drives all axes with a limit switch into its middle position → range measure is required as precondition)
- Manual Control (Move a single axis with configured step width)
- Move Absolute (Moves all active axes to an absolute position entered in destination)
- Change value of analog output 1 & 2
- Directly send commands like “?pos” or “?version” (Please be careful, here you have full access to all parameters of the controller)
- Movement demos like “Sequence” or “Maeander”
- Set the actual position of all axes to zero
- Check and change “velocity” and “acceleration” of every axis
- Display the range values for limit switches (calibration and range measure is required before)



### 3. General Information of DLL Usage

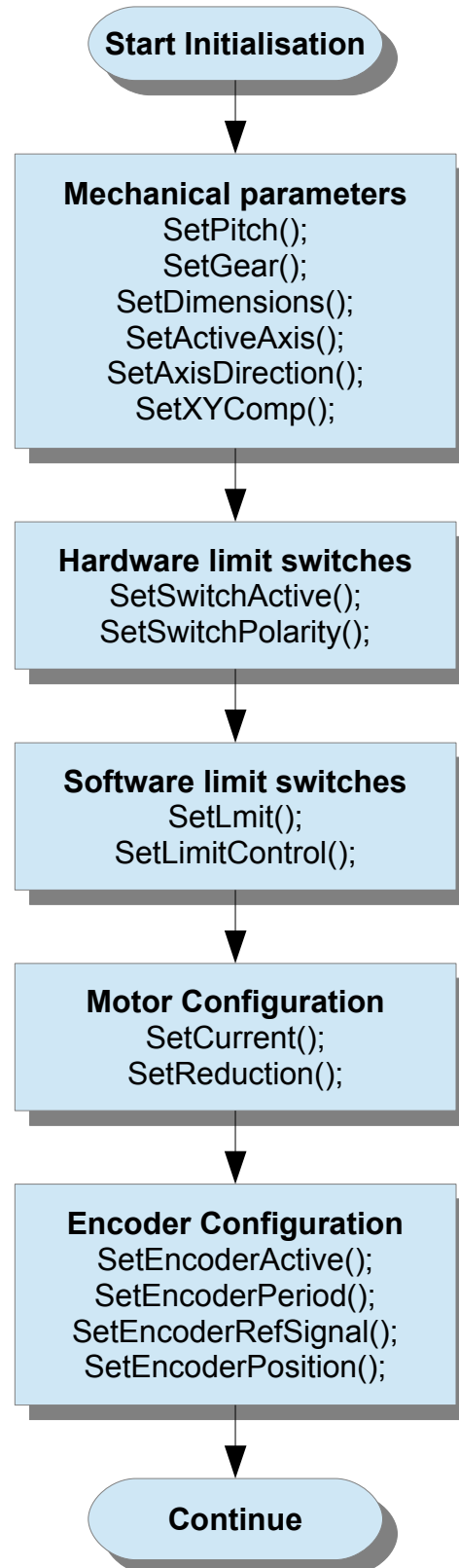
The following flow chart shows how to establish and end Tango communication and is valid for all different physical layer like RS232, USB, PCI and PCIe. All Tango application programs, independent of chosen and involved programming language, should follow this guide line. DLL functions are listed and described in detail in next chapters.

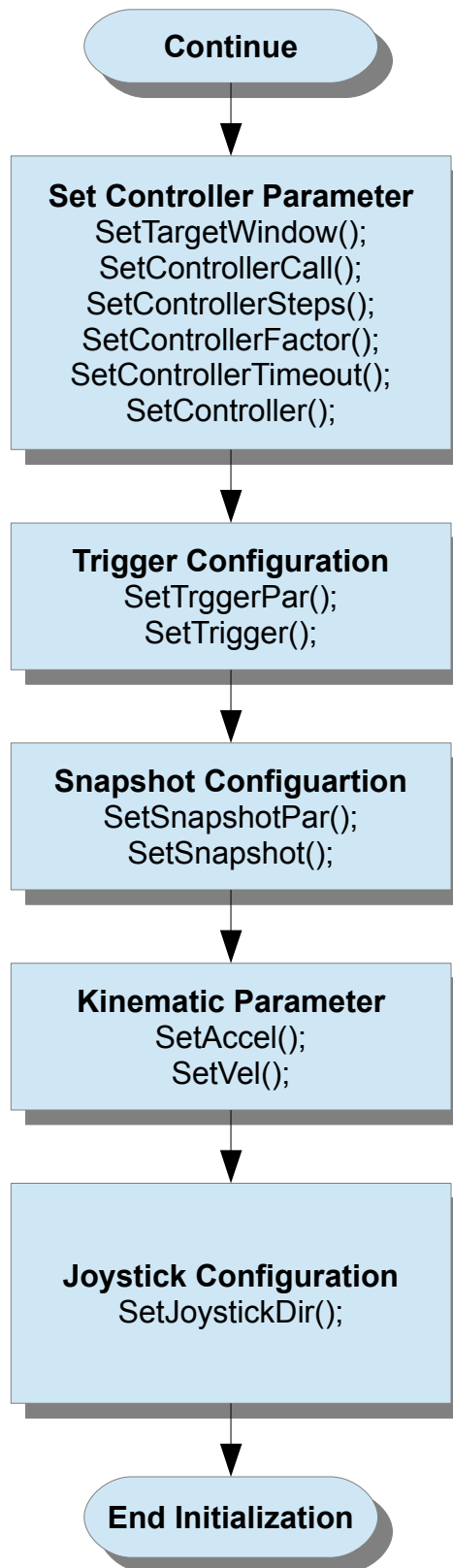


### 3.1. Initialization of Controller

Märzhäuser stages are often ETS coded. Tango uses all ETS available data for correct stage initialization. Please ask our service department if ETS is present and initialization data may be skipped.

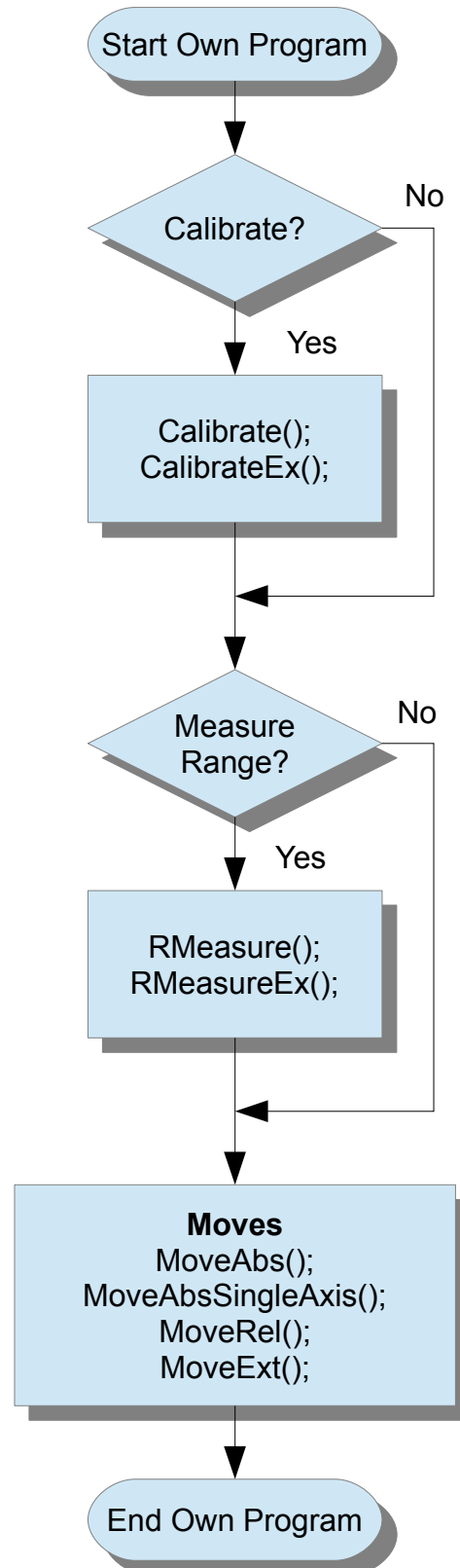
Note: Any mechanics may be damaged if wrong parameters are used. Please be careful to use correct stage data only to prevent any damage. Follow below flow chart to transmit individual settings.





## 3.2. Own Program Section

In the own program section, the user can program desired functionality of the controller. This includes movements, if desired depending on status of digital I/Os as well as setting trigger signals depending on the position, etc.



## 4. Functions

### 4.1. Quick Reference

#### DLL Configuration / Interface:

Command	Brief Description	Page
ConnectSimple	Connect to Tango	19
CreateLSID	Creates a Tango-ID number	19
Disconnect	Disconnects Tango Controller from DLL	19
EnableCommandRetry	This command enables switching on / off of repeated command sending in case of communication errors	20
FlushBuffer	Clears the receive buffer from possibly remaining data fragments	20
FreeLSID	Releases the previously created Tango ID-Number	20
SendString	Sends strings to Tango (enables using all commands as ASCII text)	21
SendStringPosCmd	Send an ASCII move command and wait for completion reply	21
SetAbortFlag	Set internal DLL flag to abort a (hanging) communication	22
SetShowProt	Switches communication monitoring on/off	22

#### Controller information:

Command	Brief Description	Page
GetSerialNr	Read out the Controller serial number	23
GetVersionStr	Provides current firmware version number	23
GetVersionStrDet	Reads detailed firmware version information	23
GetVersionStrInfo	Retrieves additional information to current version number	23

#### Status Requests:

Command	Brief Description	Page
GetError	Provides current error number	24
GetPos	Retrieves current position of all axes	24
GetPosEx	Retrieves values of current encoder- or motor-positions of all axes	24
GetPosSingleAxis	Retrieves current position of one axis	25
GetStatus	Provides current Controller status	25
GetStatusAxis	Provides current status of one axis	25
GetStatusLimit	Provides current status of software limits of all axes	26
SetAutoStatus	Switches Auto-Status reply on/off	26

## Controller Settings:

Command	Brief Description	Page
GetAccel	Read actual acceleration	27
SetAccel	Set required acceleration	27
GetActiveAxes	Retrieve axes state	27
GetAccelFunction	Retrieve actual acceleration function	27
SetAccelFunction	Set acceleration function trapezoidal or sinusoidal	27
SetActiveAxes	Set axes state	28
GetAxisDirection	Retrieve axis direction	28
SetAxisDirection	Set axis direction	28
GetCalibBackSpeed	Retrieve calibration backward speed	28
SetCalibBackSpeed	Set calibration backward speed	29
GetCalibOffset	Retrieve calibration offset	29
SetCalibOffset	Set calibration offset	29
GetCalibrateDir	Retrieve calibration direction	29
SetCalibrateDir	Set calibration direction	30
GetPitch	Read actual spindle pitch	32
SetPitch	Set required spindle pitch	33
GetPowerAmplifier	Retrieves actual state of power amplifier	33
SetPowerAmplifier	Set required state of power amplifier	33
GetReduction	Read actual current reduction	33
SetReduction	Set current reduction	34
GetRMOOffset	Retrieve range measure offset	34
SetRMOOffset	Set range measure offset	34
GetSpeedPoti	Retrieve speed potentiometer	35
SetSpeedPoti	Set speed potentiometer	35
GetStopAccel	Retrieve stop acceleration	35
SetStopAccel	Set stop acceleration	35
GetStopPolarity	Retrieve stop polarity	35
SetStopPolarity	Set stop polarity	36
GetVel	Retrieves actual max velocity	36
SetVel	Set required velocity	36
GetVelFac	Retrieves velocity factor	36
SetVelFac	Set velocity factor	37
LStepSave	save all actual parameter in controller	37
SetAccelSingleAxis	Set acceleration for a single axis	37
SetVelSingleAxis	Set velocity for a single axis	37
SoftwareReset	Reset and reboot the controller	37

## Move Commands and Position Management:

Command	Brief Description	Page
Calibrate	Calibrate enabled axes to the CAL limit switches	38
CalibrateEX	Calibrates single axes	38
ClearPos	Sets position values to zero	38
GetDelay	Provides delay of vector start	39
GetDistance	Provides distance started with MoveRelShort	39
MoveAbs	Moves to absolute position of all axes	40
MoveAbsSingleAxis	Moves to absolute position of single axis	40
MoveEx	Extended move/move relative command with axis bit mask	41
MoveRel	Move by relative vector for all axes	41
MoveRelShort	Relative positioning (short command)	42
MoveRelSingleAxis	Move single axis relatively	42
RMeasure	Measure maximum travel range of all axes	42
RMeasureEx	Measure max. travel range of axes selected by the axis bit mask	43
SetDelay	Causes delay of vector start	39
SetDistance	Sets distance for MoveRelShort command	39
SetPos	Set current position to the desired value	43
StopAxes	Stops all moving axes	43
WaitForAxisStop	Function returns as soon as all axes chosen in bit mask have reached their end position	44

## Joystick and Handwheel:

Command	Brief Description	Page
GetDigJoySpeed	Retrieves current digital joystick speed	45
GetHandWheel	Retrieves handwheel status	45
GetJoystick	Retrieves analog joystick status	46
GetJoystickDir	Retrieves revolve direction of motor for joystick	46
GetJoystickWindow	Retrieves joystick window	47
GetHwFactor	Retrieves handwheel factor	48
GetHwFactorB	Retrieves second handwheel factor	49
GetZwTravel	Retrieves z-wheel travel distances	49
GetKey	Retrieves key state	50
GetKeyLatch	Retrieves and clears latched key states	50
SetDigJoySpeed	Start a move at constant speed (commanded digital joystick)	45
SetHandWheelOff	Switches handwheel off	47
SetHandWheelOn	Switches handwheel on	48
SetJoystickDir	Sets analog joystick direction	47
SetJoystickOff	Switches analog joystick off	48
SetJoystickOn	Switches analog joystick on	48
SetJoystickWindow	Set analog joystick idle window	47
SetHwFactor	Set handwheel factor	49
SetHwFactorB	Set second handwheel factor	49
SetZwTravel	Set z-wheel travel distances	49
ClearKeyLatch	Clears latched key states	50

### **Control Console with Trackball and Joyspeed Keys (Customized Application):**

Command	Brief Description	Page
GetBPZ	Retrieves status of control console	51
GetBPZJoyspeed	Retrieves control console joystick speed	51
GetBPZTrackballBackLash	Retrieves control console trackball backlash	52
GetBPZTrackballFactor	Retrieves control console trackball factor	52
SetBPZ	Switches control console on / off	51
SetBPZJoyspeed	Set control console joystick speed	52
SetBPZTrackballBackLash	Set control console trackball backlash	52
SetBPZTrackballFactor	Set control console trackball factor	53

### **Limit Switches (Hardware and Software):**

Command	Brief Description	Page
GetAutoLimitAfterCalibRM	Provides, whether internal software limits are set when calibrating or measuring stage travel range	54
GetLimit	Provides travel range limits of single axes	54
GetLimitControl	Retrieves whether area control is switched on or off	55
GetSwitchActive	Provides, whether limit switches are active	56
GetSwitches	Retrieves status of all limit switches	56
GetSwitchPolarity	Retrieves polarity of limit switches	57
SetSwitchType	Retrieves status of pull up or pull down resistor array (NPN or PNP)	57
SetAutoLimitAfterCalibRM	Prevents setting internal software limits by calibration or range measure	54
SetLimit	Sets travel range limits of single axes	55
SetLimitControl	Switches area control on / off	55
SetSwitchActive	Enable/disable limit switches	56
SetSwitchPolarity	Sets polarity of limit switches	57
SetSwitchType	Set resistor pull-up or pull down to match NPN or PNP switches	58

### **Digital and Analog Inputs and Outputs:**

Command	Brief Description	Page
GetAnalogInput	Retrieves current level of analogue input signals	59
GetDigitalInputs	Retrieve all digital input pin levels	59
GetDigitalInputsE	Retrieve additional digital inputs 16-31	59
SetAnalogOutput	Set analogue output voltage	59
SetDigIO_Distance	Activate an output, depending on set distance before or after reaching determined position	60
SetDigIO_EmergencyStop	Assign Emergency-Stop pin	60
SetDigIO_Off	Switch off digital I/O functionality	60
SetDigIO_Polarity	Set polarity	61
SetDigitalOutput	Set digital output	61
SetDigitalOutputs	Set digital outputs 0-15	61
SetDigitalOutputsE	Set additional digital outputs 16-31	61



## Encoder Settings:

Command	Brief Description	Page
ClearEncoder	Set encoder position to zero	62
GetEncoder	Retrieves all encoder positions	62
GetEncoderActive	Retrieves which encoder is activated after calibration ( <i>encmask</i> )	62
GetEncoderMask	Retrieve status of encoders (" <i>enc</i> " command!)	63
GetEncoderPeriod	Retrieves length of encoder signal period	64
GetEncoderPosition	Provides, whether encoder- or motor- position is displayed	65
GetEncoderRefSignal	Provides if reference signal from encoder shall be evaluated when calibrating	65
SetEncoderActive	Select encoder to be activated after calibration	63
SetEncoderMask	Activates / deactivates encoders	63
SetEncoderPeriod	Set length of encoder period	65
SetEncoderPosition	Switches encoder value display on / off	65
SetEncoderRefSignal	Evaluate encoder reference signal when calibrating.	66

## Closed Loop Settings:

Command	Brief Description	Page
ClearCtrFastMoveCounter	Resets number of executed FastMove functions to 0	67
GetController	Retrieve controller mode	67
GetControllerCall	Provides controller call interval	68
GetControllerFactor	Retrieve setting of controller factor	68
GetControllerSteps	Retrieve controller steps	69
GetControllerTimeout	Retrieves setting of controller monitoring timeout	69
GetControllerTWDelay	Retrieve controller delay for target window	70
GetCtrFastMove	Retrieves whether FastMove function is switched on or off	70
GetCtrFastMoveCounter	Retrieves number of executed FastMove functions	70
GetTargetWindow	Retrieves target windows of all axes	71
SetController	Set controller mode	67
SetControllerCall	Set controller call time	68
SetControllerFactor	Set controller factor	68
SetControllerSteps	Set controller steps	69
SetControllerTimeout	Set controller monitoring timeout	69
SetControllerTWDelay	Set controller delay	70
SetCtrFastMoveOff	Switch off FastMove function	71
SetCtrFastMoveOn	Switch on FastMove function	71
SetTargetWindow	Set controller target windows	71

## Trigger Output:

Command	Brief Description	Page
GetTrigCount	Retrieve trigger counter value	72
GetTrigger	Retrieve trigger setting	72
GetTriggerPar	Retrieve trigger parameters	73
SetTrigCount	Set trigger counter value	72
SetTrigger	Switch trigger on / off	72
SetTriggerPar	Set trigger parameters	73

**Snapshot-Input:**

Command	Brief Description	Page
GetSnapshot	Provides current status of Snapshot	74
GetSnapshotCount	Read Snapshot counter	74
GetSnapshotFilter	Retrieve input filter	74
GetSnapshotPar	Retrieve Snapshot parameters	75
GetSnapshotPos	Retrieve Snapshot position	75
GetSnapshotPosArray	Retrieve Snapshot position from array	76
SetSnapshot	Switch Snapshot on / off	74
SetSnapshotFilter	Set input filter	74
SetSnapshotPar	Set Snapshot parameters	75

## 4.2. DLL Configuration / Interface

### 4.2.1 ConnectSimple

<b>Description</b>	Connect with Tango. Without connection setup, connection is not possible.
<b>C++</b>	int LSX_ConnectSimple(int lLSID, int lAnInterfaceType, char *pcAComName, int lABaudRate, BOOL bAShowProt);
<b>Parameters</b>	<i>AnInterfaceType</i> : Interface type = 1 (always 1 for RS232, PCI and USB) <i>AComName</i> : Name of COM-Interface, e.g. "COM2" <i>ABaudRate</i> : e.g. 57600 Baud (only important for RS232) <i>AShowProt</i> : Determines, if interface protocol shall be shown
<b>Example</b>	pTango->ConnectSimple(1, 1, "COM2", 57600, TRUE);

### 4.2.2 CreateLSID

<b>Description</b>	Creates a Tango ID-Number. This is used as additional parameter for Tango DLL commands to select the Tango, which is addressed for the command from a variety of connected Tangos.
<b>C++</b>	int LSX_CreateLSID(int *plLSID);
<b>Parameters</b>	<i>LSID</i> : Contains a new Tango ID-Number after calling CreateLSID, which can then be used for commands such as connect, move and others
<b>Example</b>	int Tango1, Tango2; pTango->CreateLSID(&Tango1); // create ID for first Tango pTango->CreateLSID(&Tango2); // create ID for second Tango

### 4.2.3 Disconnect

<b>Description</b>	Disconnect from Tango. After calling this function, commands can no longer be sent to the Tango Controller. This function should be called just before closing the program.
<b>C++</b>	int LSX_Disconnect(int lLSID);
<b>Parameters</b>	-
<b>Example</b>	pTango->Disconnect(1);

#### 4.2.4 EnableCommandRetry

<b>Description</b>	This function enables/disables repeated sending of commands in case of errors (Default enabled).
<b>C++</b>	<code>int LSX_EnableCommandRetry (int lLSID, BOOL bAValue);</code>
<b>Parameters</b>	<i>AValue</i> : TRUE <input type="checkbox"/> in case of errors Tango DLL repeats sending certain command (especially in case of WaitForAxisStop) FALSE <input type="checkbox"/> disable repeated sending
<b>Example</b>	<code>pTango-&gt;EnableCommandRetry(1, FALSE);</code>

#### 4.2.5 FlushBuffer

<b>Description</b>	Clear communication input buffer. Can be used in error situations to remove no longer needed feedback messages from the input buffer.
<b>C++</b>	<code>int LSX_FlushBuffer (int lLSID, int lAValue);</code>
<b>Parameters</b>	<i>AValue</i> : not used momentarily, can be set = 0
<b>Example</b>	<code>pTango-&gt;FlushBuffer(1, 0);</code>

#### 4.2.6 FreeLSID

<b>Description</b>	Sets a created Tango ID-Number free again. This is used as an additional parameter in Tango-DLL commands to select the Tango to which command is aimed at from a range of connected Tangos. FreeLSID should not be called before Disconnect.
<b>C++</b>	<code>int LSX_FreeLSID(int lLSID);</code>
<b>Parameters</b>	<i>LSID</i> : The given Tango ID-Number, which is to be set free. Do not try to use the ID after FreeLSID has been executed.
<b>Example</b>	<code>int Tango1;</code>  <code>pTango-&gt;CreateLSID(&amp;Tango1);</code> <code>pTango-&gt;ConnectSimple(Tango1, ...);</code>  <code>pTango-&gt;Disconnect(Tango1);</code> <code>pTango-&gt;FreeLSID(Tango1);</code>

4.2.7 SendString	
<b>Description</b>	Sends an ASCII string to the Tango.
<b>C++</b>	<pre>int LSX_SendString (int ILSID, char *pcStr, char *pcRet, int lMaxLen, BOOL bReadLine, int lTimeout);</pre>
<b>Parameters</b>	<p><b>Str</b> □ Zero-terminated string, which is to be sent to controller. String must end with a carriage return (\r).</p> <p><b>Ret</b> □ Buffer, containing return message from Tango, in case ReadLine = TRUE or also ZERO (NULL), in case ReadLine = FALSE;</p> <p><b>MaxLen</b> □ Max. amount of characters allowed to be copied into buffer</p> <p><b>ReadLine</b> □ TRUE = read return message from Tango FALSE = don't wait for return message</p> <p><b>Timeout</b> □ Max. waiting period for return message [ms]</p>
<b>Example</b>	<pre>pTango-&gt;SendString(1, "?version\r", pcVer, 256, TRUE, 1000); // Read version number, 1 Second Timeout  pTango-&gt;SendString(1, "!baud 115200\r", NULL, 0, FALSE, 0); // set max. baud rate for RS232</pre>

4.2.8 SendStringPosCmd	
<b>Description</b>	Send move command to Tango as a string and wait for return message.
<b>C++</b>	<pre>int LSX_SendStringPosCmd (int ILSID, char *pcStr, char *pcRet, int lMaxLen, BOOL bReadLine, int lTimeout);</pre>
<b>Parameters</b>	<p><b>Str</b> □ Zero-terminated ASCII string, which is to be sent to the controller</p> <p><b>Ret</b> □ Buffer, containing return message from Tango, in case ReadLine = TRUE Or also ZERO (NULL), in case ReadLine = FALSE;</p> <p><b>MaxLen</b> □ Max. amount of characters allowed copied into buffer</p> <p><b>ReadLine</b> □ TRUE = read return message from Tango FALSE = don't wait for return message</p> <p><b>Timeout</b> □ Max. waiting period for return message [ms]</p>
<b>Example</b>	<pre>pTango-&gt;SendStringPosCmd(1, "!moa 1 2\r", pcRet, 256, TRUE, 10000);</pre>

#### 4.2.9 SetAbortFlag

<b>Description</b>	<p>Set flag so that communication with Tango is cut off.</p> <p>A function, which, when calling LSX_SetAbortFlag is still waiting for return message from controller (e.g. drive commands), then returns with an error message. The use of this function especially makes sense for programs with message processing routines or with multiple threads, in case, for example, a drive movement shall be stopped quickly.</p>
<b>C++</b>	int LSX_SetAbortFlag (int ILSID);
<b>Parameters</b>	-
<b>Example</b>	<pre>pTango-&gt;SetAbortFlag(1); pTango-&gt;StopAxes(1); // closes communication with Tango and sends stop command for all axes</pre>

#### 4.2.10 SetShowProt

<b>Description</b>	Switches the interface protocol window on / off.
<b>C++</b>	int LSX_SetShowProt (int ILSID, BOOL bShowProt);
<b>Parameters</b>	<b>ShowProt:</b> TRUE = show Interface Protocol window FALSE = hide Interface Protocol window
<b>Example</b>	<pre>pTango-&gt;SetShowProt(1, TRUE); // Show interface protocol for Tango1, in case not already visible</pre>

## 4.3. Controller Information

4.3.1 GetSerialNr	
<b>Description</b>	Reads out the Tango serial number.
<b>C++</b>	<code>int LSX_GetSerialNr (int ILSID, char *pcSerialNr, int lMaxLen);</code>
<b>Parameters</b>	<p><b>SerialNr:</b> Pointer to a buffer, in which the serial number will be returned</p> <p><b>MaxLen:</b> Max. amount of digits allowed to be copied into buffer</p> <p>Example value 090103001 = 09 = YY, 01 = WW, 03 = 3Axes max., 001 Index</p>
<b>Example</b>	<code>pTango-&gt;GetSerialNr(1, pcSerialNr, 256);</code>

4.3.2 GetVersionStr	
<b>Description</b>	Returns current firmware version number (?ver).
<b>C++</b>	<code>int LSX_GetVersionStr (int ILSID, char *pcVers, int lMaxLen);</code>
<b>Parameters</b>	<p><b>Vers:</b> Pointer to a character buffer, in which the version number will be returned</p> <p><b>MaxLen:</b> Max. amount of characters allowed to be copied into buffer</p>
<b>Example</b>	<code>pTango-&gt;GetVersionStr(1, pcVers, 64);</code> <i>// retrieve version number</i>

4.3.3 GetVersionStrDet	
<b>Description</b>	Retrieves detailed configuration of Tango (?det) as ASCII digits.
<b>C++</b>	<code>int LSX_GetVersionStrDet (int ILSID, char *pcVersDet, int lMaxLen);</code>
<b>Parameters</b>	<p><b>VersDet:</b> Pointer to a buffer, in which the string will be returned</p> <p><b>MaxLen:</b> Max. amount of characters allowed to be copied into buffer</p>
<b>Example</b>	<code>pTango-&gt;GetVersionStrDet(1, pcVersDet, 16);</code> <i>// retrieve detailed configuration</i>

4.3.4 GetVersionStrInfo	
<b>Description</b>	Provides optional internal information on the controllerversion (?iver).
<b>C++</b>	<code>int LSX_GetVersionStrInfo (int ILSID, char *pcVersInfo, int lMaxLen);</code>
<b>Parameters</b>	<p><b>VersInfo:</b> Pointer to a buffer</p> <p><b>MaxLen:</b> Max. amount of characters to be copied into buffer</p>
<b>Example</b>	<code>pTango-&gt;GetVersionStrInfo(1, pcVersInfo, 16);</code>

## 4.4. Status Requests

### 4.4.1 GetError

<b>Description</b>	Provides current error number.
<b>C++</b>	int LSX_GetError (int ILSID, int *plErrorCode);
<b>Parameters</b>	<i>ErrorCode</i> : Error number
<b>Example</b>	pTango->GetError(1, &ErrorCode);

### 4.4.2 GetPos

<b>Description</b>	Retrieves current position of all axes.
<b>C++</b>	int LSX_GetPos (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);
<b>Parameters</b>	<i>X, Y, Z, A</i> : Positions
<b>Example</b>	pTango->GetPos(1, &X, &Y, &Z, &A);

### 4.4.3 GetPosEx

<b>Description</b>	Retrieves encoder or motor positions of all axes.  If any axis is not available, 0.0 is returned as a value.
<b>C++</b>	int LSX_GetPosEx (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA, BOOL bEncoder);
<b>Parameters</b>	<i>X, Y, Z, A</i> : Position parameter <i>Encoder</i> = TRUE □ Provide encoder parameters if encoder connected = FALSE □ Provide motor position values
<b>Example</b>	pTango->GetPosEx(1, &X, &Y, &Z, &A, TRUE);



4.4.4 GetPosSingleAxis	
<b>Description</b>	Retrieves current position of a single axis. If axis is not available, 0.0 is returned as a value.
<b>C++</b>	<code>int LSX_GetPosSingleAxis (int ILSID, int IAxis, double *pdPos);</code>
<b>Parameters</b>	<i>Axis</i> : Axis of which the position parameters shall be retrieved from, X, Y, Z and A, numbered from 1 to 4 <i>Pos</i> : Positions
<b>Example</b>	<code>pTango-&gt;GetPosSingleAxis(1, 2, &amp;Pos);</code> <i>// retrieves position of Y-Axis</i>

4.4.5 GetStatus	
<b>Description</b>	Provides current status of the controller.
<b>C++</b>	<code>int LSX_GetStatus (int ILSID, char *pcStat, int IMaxLen);</code>
<b>Parameters</b>	<i>Stat</i> : Pointer to a buffer, in which the status string will be returned <i>MaxLen</i> : Max. amount of characters allowed to be copied into buffer
<b>Example</b>	<code>pTango-&gt;GetStatus(1, &amp;Stat, 16);</code>

4.4.6 GetStatusAxis	
<b>Description</b>	Provides current status of the axes.
<b>C++</b>	<code>int LSX_GetStatusAxis (int ILSID, char *pcStatusAxisStr, int IMaxLen);</code>
<b>Parameters</b>	<i>StatusAxisStr</i> : Pointer to a buffer in which status string will be returned <i>MaxLen</i> : Max. amount of characters allowed to be copied into buffer e.g.: @ M -- J -- C -- S -- A -- D -- U T @ = Axis stands still M = Axis is in motion = Axis is not enabled J = Joystick switched on C = Axis is in closed loop A = Return message after calibration (cal) E = Error when calibrating (limit switch not cleared correctly) D = Return message after measuring stage travel range (rm) U = Setup mode T = Timeout
<b>Example</b>	<code>pTango-&gt;GetStatusAxis(1, &amp;StatusAxisStr, 16);</code>

4.4.7 GetStatusLimit	
<b>Description</b>	Provides current status of software limits of each axis.
<b>C++</b>	int LSX_GetStatusLimit (int ILSID, char *pcLimit, int IMaxLen);
<b>Parameters</b>	<p><b>Limit:</b> Pointer to a buffer, in which the status of the axes will be returned</p> <p>e.g.: AA A DD LL L L</p> <p>A = Axis has been calibrated</p> <p>D = Stage travel range has been measured (rm)</p> <p>L = Software limit has been set</p> <p>= Software limit remains unchanged</p> <p><b>MaxLen:</b> Max. amount of characters allowed to be copied into the buffer</p>
<b>Example</b>	pTango->GetStatusLimit(1, &Limit, 32);

4.4.8 SetAutoStatus	
<b>Description</b>	<p>Switches Auto-Status on/off.</p> <p>Please note: As a rule, AutoStatus mode should not be changed as Tango DLL sets correct mode for travel commands etc., changing Autostatus manually to a value of 0, 2 or 3 could cause errors.</p>
<b>C++</b>	int LSX_SetAutoStatus (int ILSID, int IValue);
<b>Parameters</b>	<p><b>Value:</b> AutoStatus mode:</p> <p>0 □ Controller sends no status</p> <p>1 □ Controller automatically sends "Position reached" messages</p> <p>2 □ Controller automatically sends "Position reached" and status messages</p> <p>3 □ There is only one carriage return sent for "Position reached"</p>
<b>Example</b>	pTango->SetAutoStatus(1, 1);

## 4.5. Settings

### 4.5.1 GetAccel

<b>Description</b>	Retrieves acceleration.
<b>C++</b>	<code>int LSX_GetAccelFunc (double *pdX, double *pdY, double *pdZ, double *pdA);</code>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Acceleration values [m/s <sup>2</sup> ]
<b>Example</b>	<code>pTango-&gt;GetAccel(1, &amp;X, &amp;Y, &amp;Z, &amp;A);</code>

### 4.5.2 SetAccel

<b>Description</b>	Set acceleration.
<b>C++</b>	<code>int LSX_SetAccel (int ILSID, double dX, double dY, double dZ, double dA);</code>
<b>Parameters</b>	<i>X, Y, Z, A</i> : 0.01 - 20.00 [m/s <sup>2</sup> ]
<b>Example</b>	<code>pTango-&gt;SetAccel(1, 1.0, 1.5, 0, 0);</code>

### 4.5.3 GetActiveAxes

<b>Description</b>	Provides the axis enable states.
<b>C++</b>	<code>int LSX_GetActiveAxes (int ILSID, int *plFlags);</code>
<b>Parameters</b>	<i>Flags</i> : 32-Bit Integer. After calling this function the axis bitmask is returned in Bits 0-4 Bit 0 = 1 □ X-Axis cleared Bit 2 = 0 □ Z-Axis not cleared
<b>Example</b>	<code>pTango-&gt;GetActiveAxes(1, &amp;Flags);</code>

### 4.5.4 GetAccelFunc

<b>Description</b>	Retrieves acceleration function.
<b>C++</b>	<code>int LSX_GetAccelFunc (int ILSID, int *IX, int *IY, int *IZ, int *IR);</code>
<b>Parameters</b>	<i>IX, IY, IZ, IR</i> : Acceleration function 0 indicate trapezoidal 1 indicate sinusoidal
<b>Example</b>	<code>pTango-&gt;GetAccel(1, &amp;IX, &amp;IY, &amp;IZ, &amp;IR);</code>

### 4.5.5 SetAccelFunc

<b>Description</b>	Sets acceleration function (0 for trapezoidal, 1 for sinusoidal).
<b>C++</b>	<code>int LSX_SetAccelFunc (int ILSID, int IX, int IY, int IZ, int IR);</code>
<b>Parameters</b>	<i>IX, IY, IZ, IR</i> : Acceleration function 0 indicate trapezoidal 1 indicate sinusoidal
<b>Example</b>	<code>pTango-&gt;SetAccel(1, IX, IY, IZ, IR);</code>

4.5.6 SetActiveAxes	
<b>Description</b>	Enable or disable axes.
<b>C++</b>	<code>int LSX_SetActiveAxes (int ILSID, int IFlags);</code>
<b>Parameters</b>	<b>Flags:</b> Bit mask, bits 0 to 4 represent axes X to A Bit 0 = 1 □ X-Axis disabled Bit 2 = 0 □ Z-Axis enabled
<b>Example</b>	<pre>pTango-&gt;SetActiveAxes(1, 3); // X- and Y-Axis cleared (Bits 0 and 1 set), // Z-Axis not cleared (Bit 2 = 0)</pre>

4.5.7 GetAxisDirection	
<b>Description</b>	Retrieves axis directions.
<b>C++</b>	<code>int LSX_GetAxisDirection (int ILSID, int *plXD, int *plYD, int *plZD, int *plAD);</code>
<b>Parameters</b>	<b>XD, YD, ZD, AD:</b> 4 32-Bit Integers 0 □ normal rotating direction 1 □ reversed rotating direction
<b>Example</b>	<code>pTango-&gt;GetAxisDirection(1, &amp;XD, &amp;YD,&amp;ZD,&amp;AD);</code>

4.5.8 SetAxisDirection	
<b>Description</b>	Set axis directions.
<b>C++</b>	<code>int LSX_SetAxisDirection (int ILSID, int IXD, int IYD, int IZD, int IAD);</code>
<b>Parameters</b>	<b>XD, YD, ZD, AD:</b> 4 32-Bit Integers 0 □ normal motor turning direction 1 □ reverse reversed motor turning direction
<b>Example</b>	<pre>pTango-&gt;SetAxisDirection(1, 1, 0, 0, 0); // reverse direction of X-Axis</pre>

4.5.9 GetCalibBackSpeed	
<b>Description</b>	Retrieves revolving speed at which axes are driven from limit switches when calibrating. Speed is equivalent to issued value * 0.01 rev/sec.
<b>C++</b>	<code>int LSX_GetCalibBackSpeed (int ILSID, int *plSpeed);</code>
<b>Parameters</b>	<b>Speed:</b> Speed value in 1/100 revolutions/second
<b>Example</b>	<code>pTango-&gt;GetCalibBackSpeed(1, &amp;lSpeed);</code>

#### 4.5.10 SetCalibBackSpeed

<b>Description</b>	Sets revolving speed at which axes are driven from limit switches when calibrating. Speed is equivalent to issued value * 0.01 rev/sec
<b>C++</b>	int LSX_SetCalibBackSpeed (int ILSID, int lSpeed);
<b>Parameters</b>	<i>Speed</i> : Speed value in 1/100 revolutions/second (within parameters of 1 to 100)
<b>Example</b>	pTango->SetCalibBackSpeed(1, 10); <i>// when calibrating, limit switches are left at 0.1 rev/sec</i>

#### 4.5.11 GetCalibOffset

<b>Description</b>	Retrieves zero position offset of axes.
<b>C++</b>	int LSX_GetCalibOffset (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA)
<b>Parameters</b>	<i>X, Y, Z, A</i> : zero position offset from cal switch, depending on dimensions
<b>Example</b>	pTango->GetCalibOffset(1, &X, &Y, &Z, &A);

#### 4.5.12 SetCalibOffset

<b>Description</b>	Sets zero position offset of axes. The axis zero position is moved from the hardware cal limit switch by this amount.
<b>C++</b>	int LSX_SetCalibOffset (int ILSID, double dX, double dY, double dZ, double dA);
<b>Parameters</b>	<i>X, Y, Z, A</i> : typically 0-5 [mm]
<b>Example</b>	pTango->SetCalibOffset(1, 1, 1, 1, 1); <i>// when calibrating, axes X, Y, Z and A are each moved for 1mm (at dimension 2 2 2 2) from zero limit switch towards stage center and then zero position is set (software limit)</i>

#### 4.5.13 GetCalibrateDir

<b>Description</b>	Retrieves calibrating direction.
<b>C++</b>	int LSX_GetCalibrateDir (int ILSID, int *plXD, int *plYD, int *plZD, int *plAD);
<b>Parameters</b>	<i>XD, YD, ZD, AD</i> : 32-Bit Integer 0 □ normal calibration direction 1 □ reversed calibration direction
<b>Example</b>	pTango->GetCalibrateDir(1, &XD, &YD,&ZD,&AD);

#### 4.5.14 SetCalibrateDir

<b>Description</b>	Set calibrating direction.
<b>C++</b>	int LSX_SetCalibrateDir (int ILSID, int IXd, int IYd, int IZd, int IAd);
<b>Parameters</b>	<i>XD, YD, ZD, AD</i> : 32-Bit Integer 0 □ normal calibration direction 1 □ reverse calibration direction
<b>Example</b>	pTango->(1, 1, 1, 0, 0);

#### 4.5.15 GetCurrentDelay

<b>Description</b>	Provides time delay for motorcurrent reduction.
<b>C++</b>	int LSX_GetCurrentDelay (int ILSID, int *pIX, int *pIY, int *pIZ, int *pIA);
<b>Parameters</b>	<i>X, Y, Z, A</i> : Time delay [ms]
<b>Example</b>	pTango->GetCurrentDelay(1, &X, &Y,&Z,&A);

#### 4.5.16 SetCurrentDelay

<b>Description</b>	Sets the time delay, after which the motor current is reduced.
<b>C++</b>	int LSX_SetCurrentDelay (int ILSID, int IX, int IY, int IZ, int IA);
<b>Parameters</b>	<i>X, Y, Z, A</i> : 010000 [ms] (A delay of 0 disables the current reduction)
<b>Example</b>	pTango->SetCurrentDelay(1, 100, 300, 1000, 0);

#### 4.5.17 GetDimensions

<b>Description</b>	Provides the applied measuring units of axes
<b>C++</b>	int LSX_GetDimensions (int ILSID, int *pIXD, int *pIYD, int *pIZD, int *pIAD);
<b>Parameters</b>	<i>XD, YD, ZD, AD</i> : Dimension units 0 □ Microsteps 1 □ μm 2 □ mm (Pre-set) 3 □ Degree 4 □ Revolutions 5 □ cm 6 □ m 7 □ Inch 8 □ mil (1/1000 Inch)
<b>Example</b>	pTango->GetDimensions(1, &XD, &YD,&ZD,&AD);

#### 4.5.18 SetDimensions

<b>Description</b>	Set measuring units of axes.
<b>C++</b>	int LSX_SetDimensions (int ILSID, int IXd, int lYD, int lZD, int lAD);
<b>Parameters</b>	<p><i>XD, YD, ZD, AD</i>: Dimension units</p> <p>0 □ Microsteps  1 □ μm  2 □ mm (Pre-set)  3 □ Degree  4 □ Revolutions  5 □ cm  6 □ m  7 □ Inch  8 □ mil (1/1000 Inch)</p>
<b>Example</b>	<p>pTango-&gt;SetDimensions(1, 3, 2, 2, 1);  <i>// X-Axis in degree, Y- and Z-Axis in mm and A-Axis in μm</i></p>

#### 4.5.19 GetGear

<b>Description</b>	Retrieves gear ratio.
<b>C++</b>	int LSX_GetGear (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);
<b>Parameters</b>	<i>X, Y, Z, A</i> : Gear ratio values
<b>Example</b>	pTango->GetGear(1, &X, &Y, &Z, &A);

#### 4.5.20 SetGear

<b>Description</b>	Set gear ratio.
<b>C++</b>	int LSX_SetGear (int ILSID, double dX, double dY, double dZ, double dA);
<b>Parameters</b>	<i>X, Y, Z, A</i> : 0.01 - 1000
<b>Example</b>	<p>pTango-&gt;SetGear(1, 4.0, 2.0, 1.0, 1.0);  <i>// programs gear ratios 1/4 for Z, 1/2 for Y and 1/1 for Z and A</i></p>

#### 4.5.21 GetMotorCurrent

<b>Description</b>	Retrieves electrical motor current.
<b>C++</b>	<code>int LSX_GetMotorCurrent (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);</code>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Electrical motor currents in [A]
<b>Example</b>	<code>pTango-&gt;GetMotorCurrent(1, &amp;X, &amp;Y, &amp;Z, &amp;A);</code>

#### 4.5.22 SetMotorCurrent

<b>Description</b>	Set electrical current of motor.
<b>C++</b>	<code>int LSX_SetMotorCurrent (int ILSID, double dX, double dY, double dZ, double dA);</code>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Motor current X, Y, Z and A-Axis in [A]
<b>Example</b>	<code>pTango-&gt;SetMotorCurrent(1, 1.0, 1.0, 0.8, 0.8);</code> <i>// motor current X- and Y-Axis 1 Ampere; Z- and A-Axis 0.8 Ampere</i>

#### 4.5.23 GetMotorSteps

<b>Description</b>	Retrieves number of motor steps.
<b>C++</b>	<code>int LSX_GetMotorSteps (int ILSID, int *IX, int *IY, int *IZ, int *IA);</code>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Number of motor steps
<b>Example</b>	<code>pTango-&gt;GetMotorSteps(1, &amp;X, &amp;Y, &amp;Z, &amp;A);</code>

#### 4.5.24 SetMotorSteps

<b>Description</b>	Set number of motor steps. (default 200 for 1,8° stepper motors)
<b>C++</b>	<code>int LSX_SetMotorSteps (int ILSID, int IX, int IY, int IZ, int IA);</code>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Motor steps X, Y, Z and A-Axis
<b>Example</b>	<code>pTango-&gt;SetMotorCurrent(1, 200, 200, 200, 20);</code> <i>// set X, Y, Z to default and A axis to 20 for special motor</i>

#### 4.5.25 GetPitch

<b>Description</b>	Provides spindle pitch.
<b>C++</b>	<code>int LSX_GetPitch (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);</code>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Spindle pitch [mm]
<b>Example</b>	<code>pTango-&gt;GetPitch(1, &amp;X, &amp;Y, &amp;Z, &amp;A);</code>



#### 4.5.26 SetPitch

<b>Description</b>	Set spindle pitch.
<b>C++</b>	int LSX_SetPitch (int ILSID, double dX, double dY, double dZ, double dA);
<b>Parameters</b>	<i>X, Y, Z, A</i> : 0.001 - 68 [mm]
<b>Example</b>	pTango->SetPitch(1, 4, 4, 4, 4); <i>// Set spindle pitch of all axes to 4mm</i>

#### 4.5.27 GetPowerAmplifier

<b>Description</b>	Provides, whether amplifiers are switched on or off.
<b>C++</b>	int LSX_GetPowerAmplifier (int ILSID, BOOL *pbAmplifier);
<b>Parameters</b>	<i>Amplifier</i> : TRUE <input type="checkbox"/> Amplifiers are switched on FALSE <input type="checkbox"/> Amplifiers are switched off
<b>Example</b>	pTango->GetPowerAmplifier(1, &Amplifier);

#### 4.5.28 SetPowerAmplifier

<b>Description</b>	Switch amplifier on / off.
<b>C++</b>	int LSX_SetPowerAmplifier (int ILSID, BOOL bAmplifier);
<b>Parameters</b>	<i>Amplifier</i> : TRUE <input type="checkbox"/> Switch amplifiers on FALSE <input type="checkbox"/> Switch amplifiers off
<b>Example</b>	pTango->SetPowerAmplifier(1, TRUE); <i>// switches amplifiers on</i>

#### 4.5.29 GetReduction

<b>Description</b>	Retrieves motor current reduction factor.
<b>C++</b>	int LSX_GetReduction (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA)
<b>Parameters</b>	<i>X, Y, Z, A</i> : Electrical motor current reduction (Within parameters from 0 to 1)
<b>Example</b>	pTango->GetReduction(1, &X, &Y, &Z, &A);

4.5.30 SetReduction	
<b>Description</b>	Set reduction factor of motor current.
<b>C++</b>	int LSX_SetReduction (int ILSID, double dX, double dY, double dZ, double dA);
<b>Parameters</b>	<i>X, Y, Z, A:</i> 0 - 1.0
<b>Example</b>	pTango->SetReduction(1, 0.1, 0.7, 0.5, 0.5); // standby current X-Axis = 0.1*rated current, Y-Axis = 0.7*rated current, Z- and A-Axis = 0.5*rated current

4.5.31 GetRMOffset	
<b>Description</b>	Retrieves axis position offsets to RM limit switch.
<b>C++</b>	int LSX_GetRMOffset (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);
<b>Parameters</b>	<i>X, Y, Z, A:</i> Limit switch position offset, depending on measuring unit (dimension).
<b>Example</b>	pTango->GetRMOffset(1, &X, &Y, &Z, &A);

4.5.32 SetRMOffset	
<b>Description</b>	Sets RM position offset of axes. The axis stops this amount before the hardware RM endswitch.
<b>C++</b>	int LSX_SetRMOffset (int ILSID, double dX, double dY, double dZ, double dA);
<b>Parameters</b>	<i>X, Y, Z, A:</i> typically 0-5 [mm]
<b>Example</b>	pTango->SetRMOffset(1, 1, 1, 1, 1); // limit positions of axes are each moved for 1mm (at dimension 2 2 2 2) towards stage center

4.5.33 GetSpeedPoti	
<b>Description</b>	Shows, whether the speed potentiometer functionality is switched on or off.
<b>C++</b>	int LSX_GetSpeedPoti (int ILSID, BOOL *pbSpePoti);
<b>Parameter:</b>	The SpePoti flag shows, whether potentiometer is switched on or off
<b>Example</b>	pTango->(1, &flag);

#### 4.5.34 SetSpeedPoti

<b>Description</b>	Switches Speed Potentiometer functionality on or off.
<b>C++</b>	<code>int LSX_SetSpeedPoti (int ILSID, BOOL bSpeedPoti);</code>
<b>Parameters</b>	<i>SpeedPoti</i> = FALSE □ pre-set speed (vel) is used as movement speed = TRUE □ pre-set speed (vel) can be reduced depending on the speed-potentiometer deflection
<b>Example</b>	<code>pTango-&gt;SetSpeedPoti(1, TRUE);</code> <i>// potentiometer is switched on</i>

#### 4.5.35 GetStopAccel

<b>Description</b>	Provides deceleration for error conditions.
<b>C++</b>	<code>int LSX_GetStopAccel (int ILSID, double *pdXD, double *pdYD, double *pdZD, double *pdAD);</code>
<b>Parameters</b>	<i>XD, YD, ZD, AD</i> : Deceleration values [m/s <sup>2</sup> ]
<b>Example</b>	<code>pTango-&gt;GetStopAccel(1, &amp;XD, &amp;YD, &amp;ZD, &amp;AD);</code>

#### 4.5.36 SetStopAccel

<b>Description</b>	Deceleration value used when moving into a limit switch or causing a stop condition. If the axis acceleration (set with LSX_SetAccel) is higher, then this higher value will be used.
<b>C++</b>	<code>int LSX_SetStopAccel (int ILSID, double dX, double dY, double dZ, double dA);</code>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Brake acceleration, within parameters 0.01 to 20 [m/s <sup>2</sup> ]
<b>Example</b>	<code>pTango-&gt;SetStopAccel(1, 1.5, 1.5, 1.5, 1.5);</code>

#### 4.5.37 GetStopPolarity

<b>Description</b>	Retrieves active polarity of the stop input signal.
<b>C++</b>	<code>int LSX_GetStopPolarity (int ILSID, BOOL *pbHighActiv);</code>
<b>Parameters</b>	<i>HighActiv</i> : TRUE □ stop input is high active FALSE □ stop input is low active
<b>Example</b>	<code>pTango-&gt;GetStopPolarity(1, &amp;HighActiv);</code>

#### 4.5.38 SetStopPolarity

<b>Description</b>	Set polarity for active stop input signal. As the stop input has a pull up resistor to 5V, ensure that switches contact to ground. A normally open contact will require a low active setting while a normally closed contact requires the high active setting.
<b>C++</b>	<code>int LSX_SetStopPolarity (int ILSID, BOOL bHighActiv);</code>
<b>Parameters</b>	<b>HighActiv:</b> TRUE <input type="checkbox"/> stop input high active FALSE <input type="checkbox"/> stop input low active
<b>Example</b>	<code>pTango-&gt;SetStopPolarity(1, FALSE);</code> <i>// stop input is low active (e.g. normally open switch to ground)</i>

#### 4.5.39 GetVel

<b>Description</b>	Retrieves velocity of all axes.
<b>C++</b>	<code>int LSX_GetVel (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);</code>
<b>Parameters</b>	<b>X, Y, Z, A:</b> Velocity values [r/sec]
<b>Example</b>	<code>pTango-&gt;GetVel(1, &amp;X, &amp;Y, &amp;Z, &amp;A);</code>

#### 4.5.40 SetVel

<b>Description</b>	Set velocity of all axes.
<b>C++</b>	<code>int LSX_SetVel (int ILSID, double dX, double dY, double dZ, double dA);</code>
<b>Parameters</b>	<b>X, Y, Z, A:</b> >0 – max. speed [r/sec]
<b>Example</b>	<code>pTango-&gt;SetVel(1, 20.0, 15.0, 0.5, 10);</code>

#### 4.5.41 GetVelFac

<b>Description</b>	Retrieves velocity reduction factor of all axes.
<b>C++</b>	<code>int LSX_GetVelFac (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);</code>
<b>Parameters</b>	<b>X, Y, Z, A:</b> Velocity factor
<b>Example</b>	<code>pTango-&gt;GetVelFac(1, &amp;X, &amp;Y, &amp;Z, &amp;A);</code>

#### 4.5.42 SetVelFac

<b>Description</b>	Set velocity reduction factor.
<b>C++</b>	<code>int LSX_SetVelFac (int ILSID, double dX, double dY, double dZ, double dA);</code>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Velocity reduction factor, within parameters 0.01 -- 1.00
<b>Example</b>	<code>pTango-&gt;SetVelFac(1, 1, 1, 0.1, 0.1);</code> <i>// reduces velocity of Z and A axes to 1/10 of nominal velocity</i>

#### 4.5.43 LStepSave

<b>Description</b>	Save current configuration in Tango (EEPROM).
<b>C++</b>	<code>int LSX_LStepSave (int ILSID);</code>
<b>Parameters</b>	-
<b>Example</b>	<code>pTango-&gt;LStepSave(1);</code>

#### 4.5.44 SetAccelSingleAxis

<b>Description</b>	Set acceleration of a single axis.
<b>C++</b>	<code>int LSX_SetAccelSingleAxis (int ILSID, int IAxis, double dAccel);</code>
<b>Parameters</b>	<i>Axis</i> : X, Y, Z, A numbered from 1 to 4 <i>Accel</i> : Acceleration 0.01 - 20.00 [m/s <sup>2</sup> ]
<b>Example</b>	<code>pTango-&gt;SetAccelSingleAxis(1, 3, 1.0);</code> <i>// sets acceleration of Z-Axis to 1.0 m/s<sup>2</sup></i>

#### 4.5.45 SetVelSingleAxis

<b>Description</b>	Set velocity of a single axis.
<b>C++</b>	<code>int LSX_SetVelSingleAxis (int ILSID, int IAxis, double dVel);</code>
<b>Parameters</b>	<i>Axis</i> : X, Y, Z, A numbered from 1 to 4 <i>Vel</i> : >0 – max. speed [r/sec]
<b>Example</b>	<code>pTango-&gt;SetVelSingleAxis(1, 2, 10.0);</code> <i>// sets speed of Y-Axis to 10 r/sec</i>

#### 4.5.46 SoftwareReset

<b>Description</b>	Software is reset to starting condition (reboot).
<b>C++</b>	<code>int LSX_SoftwareReset (int ILSID);</code>
<b>Parameters</b>	-
<b>Example</b>	<code>pTango-&gt;SoftwareReset(1);</code>

## 4.6. Move Commands and Positioning Management

### 4.6.1 Calibrate

<b>Description</b>	<p>All enabled axes will be calibrated.</p> <p>Axes are driven towards smaller position values until reaching the cal limit switch and then driven with reduced speed in opposite direction until limit switch is no longer active. If a position offset is configured, the axis continues traveling for that distance. Then the zero point is set.</p>
<b>C++</b>	int LSX_Calibrate (int ILSID);
<b>Parameters</b>	-
<b>Example</b>	pTango->Calibrate(1);

### 4.6.2 CalibrateEx

<b>Description</b>	<p>Calibrates single axes.</p> <p>Only calibrates axes with corresponding Bit set in transferred Integer value.</p>
<b>C++</b>	int LSX_CalibrateEx (int ILSID, int IFlags);
<b>Parameters</b>	<p><b>Flags:</b> Bit mask</p> <p>Bit 0=X, Bit 1=Y, Bit 2=Z, Bit 3=A</p> <p>If Bit 2 = 1 □ calibrate Z-Axis</p> <p>If Bit 2 = 0 □ do not calibrate Z-Axis</p>
<b>Example</b>	<p>pTango-&gt;CalibrateEx(1, 6);</p> <p><i>// only calibrate Y- and Z-Axis (Bit 1 and 2 set)</i></p>

### 4.6.3 ClearPos

<b>Description</b>	<p>Sets current position and internal position counter to 0.</p> <p>This function is needed for endless axes, as controller can only process □1,000 motor revolutions within its parameters.</p> <p>This instruction will be ignored for axes with encoders.</p>
<b>C++</b>	int LSX_ClearPos (int ILSID, int IFlags);
<b>Parameters</b>	<p><b>Flags:</b> Bit mask</p> <p>Bit 0=X, Bit 1=Y, Bit 2=Z, Bit 3=A</p> <p>Bit 0 = 1 □ position of X-Axis is set to zero.</p> <p>Bit 1 = 0 □ function is not executed for Y-Axis.</p>

#### 4.6.4 GetDelay

<b>Description</b>	Retrieves time delay (wait time) until a commanded move is executed.
<b>C++</b>	<code>int LSX_GetDelay (int ILSID, int *plDelay);</code>
<b>Parameters</b>	<i>Delay</i> : Delay [ms]
<b>Example</b>	<code>pTango-&gt;GetDelay(1, &amp;Delay);</code>

#### 4.6.5 SetDelay

<b>Description</b>	Sets the time for which move commands are delayed. Before each positioning the controller waits for this period of time delay.
<b>C++</b>	<code>int LSX_SetDelay (int ILSID, int lDelay);</code>
<b>Parameters</b>	<i>Delay</i> : 0 - 10000 [ms]
<b>Example</b>	<code>pTango-&gt;SetDelay(1, 1000);</code> <i>// 1 Second delay until a move command is executed</i>

#### 4.6.6 GetDistance

<b>Description</b>	Retrieve distance values last used for LSX_MoveRelShort.
<b>C++</b>	<code>int LSX_GetDistance (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);</code>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Current distances of all axes, depending on corresponding measuring unit.
<b>Example</b>	<code>pTango-&gt;GetDistance(1, &amp;X, &amp;Y, &amp;Z, &amp;A);</code>

#### 4.6.7 SetDistance

<b>Description</b>	Set distance. Sets distance parameters for command LSX_MoveRelShort. This enables very fast equal distance relative positioning without the need of communication overhead.
<b>C++</b>	<code>int LSX_SetDistance (int ILSID, double dX, double dY, double dZ, double dA);</code>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Min-/max- travel range, values depend on measuring unit.
<b>Example</b>	<code>pTango-&gt;SetDistance(1, 1, 2, 0, 0);</code> <i>// sets distances for axes X to 1mm and Y to 2mm (if dimension=2), Z and A are not moved when calling function LSX MoveRelShort</i>

#### 4.6.8 MoveAbs

<b>Description</b>	All axes are moved absolute positions. Axes X, Y, Z and A are positioned at transferred position values.
<b>C++</b>	int LSX_MoveAbs (int ILSID, double dX, double dY, double dZ, double dA, BOOL bWait);
<b>Parameters</b>	<b>X, Y, Z, A:</b> □ Travel range, command depends on measuring unit <b>Wait:</b> Determines, whether function shall return after reaching position (= TRUE) or directly after sending the command (= FALSE)
<b>Example</b>	pTango->MoveAbs(1, 10.0, 10.0, -10.0, 10.0, TRUE);

#### 4.6.9 MoveAbsSingleAxis

<b>Description</b>	Positions a single axis at the transferred position.
<b>C++</b>	int LSX_MoveAbsSingleAxis (int ILSID, int IAxis, double dValue, BOOL bWait);
<b>Parameters</b>	<b>Axis:</b> X, Y, Z and A, numbered from 1 to 4 <b>Value:</b> Position, command depends on measuring unit (dimension)
<b>Example</b>	pTango->MoveAbsSingleAxis(1, 2, 10.0); // position Y-Axis absolutely at 10mm (dimension=2)



#### 4.6.10 MoveEx

<b>Description</b>	<p>Extended move command.</p> <p>Function LSX_MoveEx can execute relative and absolute travel commands, synchronously as well as asynchronously. The number of axes, which are to be moved, can be determined by using AxisCount parameter. For example this function can be used to move X and Y.</p>
<b>C++</b>	<pre>int LSX_MoveEx (int ILSID, double dX, double dY, double dZ, double dA, BOOL bRelative, BOOL bWait, int lAxisCount);</pre>
<b>Parameters</b>	<p><b>X, Y, Z, A:</b> Position vectors</p> <p><b>Relative:</b> When Relative = FALSE, values of X, Y, Z and A are interpreted as absolute coordinates when Relative = TRUE, they are interpreted as relative coordinates to current position</p> <p><b>Wait:</b> If Wait = TRUE is set, function doesn't return before reaching the target position, otherwise it returns immediately after sending the command to the Tango.</p> <p><b>AxisCount:</b> Number of axes, which are to be moved e.g. if AxisCount = 1, only X is moved e.g. if AxisCount = 2, X and Y are moved ...</p>
<b>Example</b>	<pre>pTango-&gt;MoveEx(1, 2.0, 3.0, 0, 0, TRUE, TRUE, 2); // X and Y are moved relatively by 2 or 3, function call returns when positions are reached</pre>

#### 4.6.11 MoveRel

<b>Description</b>	<p>Move relative position.</p> <p>Axes X, Y, Z and A are moved by the transmitted distances. All axes reach their destinations simultaneously.</p>
<b>C++</b>	<pre>int LSX_MoveRel (int ILSID, double dX, double dY, double dZ, double dA, BOOL bWait);</pre>
<b>Parameters</b>	<p><b>X, Y, Z, A:</b> +/- Travel range, command depends on measuring unit (dimension)</p> <p><b>Wait:</b> TRUE = function waits until position is reached FALSE = function does not wait</p>
<b>Example</b>	<pre>pTango-&gt;MoveRel(1, 10.0, 10.0, -10.0, 10.0, TRUE);</pre>

#### 4.6.12 MoveRelShort

<b>Description</b>	Relative positioning (short command).  This command may be used to execute several fast equal distance relative moves.  Distances have to be pre-set once with LSX_SetDistance.
<b>C++</b>	int LSX_MoveRelShort (int ILSID);
<b>Parameters</b>	-
<b>Example</b>	pTango->SetDistance(1, 1.0, 1.0, 0, 0); for (i = 0; i < 10; i++) pTango->MoveRelShort(1); <i>// position X- and Y-Axis 10 times relatively by 1mm</i>

#### 4.6.13 MoveRelSingleAxis

<b>Description</b>	Move single axis relative.
<b>C++</b>	int LSX_MoveRelSingleAxis (int ILSID, int lAxis, double dValue, BOOL bWait);
<b>Parameters</b>	<i>Axis</i> : X, Y, Z and A numbered from 1 to 4 <i>Value</i> : Distance, command depends on set measuring unit
<b>Example</b>	pTango->MoveRelSingleAxis(1, 3, 5,0); <i>// Z-Axis is moved by 5mm in positive direction</i>

#### 4.6.14 RMeasure

<b>Description</b>	Travels to maximum position of all enabled axes.  Axes are driven towards larger position values until reaching rm limit switch and then driven with reduced speed in opposite direction until limit switch is no longer active. If a rm position offset is configured, the axis continues traveling for that distance. Then the max. possible travel range is set. Only to be executed when the stage features limit switches on either end. After this command the controller remembers the switch position and disables a possible security speed limitation.
<b>C++</b>	int LSX_RMeasure (int ILSID);
<b>Parameters</b>	-
<b>Example</b>	pTango->RMeasure(1);

#### 4.6.15 RMeasureEx

<b>Description</b>	Measure maximum position of axes (max. travel range). Moves the stage towards the RM limit switch only for the axes whose corresponding axis bit mask is set.
<b>C++</b>	int LSX_RMeasureEx (int ILSID, int IFlags);
<b>Parameters</b>	<i>Flags</i> : Bit mask Bit 2 = 1 □ calibrate Z-Axis Bit 2 = 0 □ Do not calibrate Z-Axis ...
<b>Example</b>	pTango->RMeasureEx(1, 2); <i>// only measure maximum position of Y-Axis</i>

#### 4.6.16 SetPos

<b>Description</b>	Set position.
<b>C++</b>	int LSX_SetPos (int ILSID, double dX, double dY, double dZ, double dA);
<b>Parameters</b>	<i>X, Y, Z, A</i> : Min- / max. range of travel, command depends on dimension
<b>Example</b>	pTango->SetPos(1, 10, 10, 0, 0); <i>// Set current position to this values</i>

#### 4.6.17 StopAxes

<b>Description</b>	Abort. Stops all moving axes.
<b>C++</b>	int LSX_StopAxes (int ILSID);
<b>Parameters</b>	-
<b>Example</b>	pTango->StopAxes(1);

#### 4.6.18 WaitForAxisStop

<b>Description</b>	<p>Function returns as soon as the axes selected by the bit mask “lAFlags” have reached their target positions or the timeout is exceeded.</p> <p>LSX_WaitForAxisStop uses '?statusaxis', to poll axis status.</p>
<b>C++</b>	<pre>int LSX_WaitForAxisStop (int lLSID, int lAFlags, int lATimeoutValue, BOOL *pbATimeout);</pre>
<b>Parameters</b>	<p><b>AFlags:</b> Bit mask</p> <p>Bit 0: X-Axis</p> <p>Bit 1: Y-Axis</p> <p>Bit 2: Z-Axis</p> <p>Bit 3: A-Axis</p> <p><b>ATimeoutValue:</b> Timeout in milliseconds</p> <p>WaitForAxisStop returns latest after this period of time</p> <p>pbATimeout is set to “TRUE”, if axes are still in motion.</p> <p>Setting lATimeoutValue = 0 disables the Timeout (wait infinite)</p> <p><b>pbATimeout</b> Flag: Shows whether a Timeout has occurred</p>
<b>Example</b>	<pre>pTango-&gt;WaitForAxisStop(1, 3, 0, flag); // wait until X- and Y-Axes have stopped, no Timeout  pTango-&gt;WaitForAxisStop(1, 7, 10000, flag); // wait until X-, Y- and Z-Axis has stopped, 10 sec. Timeout</pre>

## 4.7. Joystick and Handwheel

4.7.1 GetDigJoySpeed	
<b>Description</b>	Retrieves current travel speed (initiated by SetDigJoySpeed digital Joystick command).
<b>C++</b>	<pre>int LSX_GetDigJoySpeed (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);</pre>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Speed values [r/sec]
<b>Example</b>	pTango->GetDigJoySpeed(1, &X, &Y, &Z, &A);

4.7.2 SetDigJoySpeed	
<b>Description</b>	This command moves axes at a constant speed. To stop the axes, a speed of 0 has to be set. Else the constant velocity is maintained until approaching a limit switch.
<b>C++</b>	<pre>int LSX_SetDigJoySpeed (int ILSID, double dX, double dY, double dZ, double dA);</pre>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Speed [r/sec], within parameter range: + max. speed
<b>Example</b>	<pre>pTango-&gt;SetDigJoySpeed(1, 0, 10.0, 25.0, 0); // Axes X and A - speed 0 and Joystick operation "OFF", // Axis Y - speed 10.0 r/sec and Joystick operation "ON", // Axis Z - speed 25.0 r/sec and Joystick operation "ON"</pre>

4.7.3 GetHandWheel	
<b>Description</b>	Retrieves handwheel status.
<b>C++</b>	<pre>int LSX_GetHandWheel (int ILSID, BOOL *pbHandWheelOn, BOOL *pbPositionCount, BOOL *pbEncoder);</pre>
<b>Parameters</b>	<p><b>HandWheelOn:</b> TRUE = handwheel switched on FALSE = handwheel switched off</p> <p><b>PositionCount:</b> TRUE = position count switched on FALSE = position count switched off</p> <p><b>Encoder:</b> TRUE = encoder values, if available</p>
<b>Example</b>	pTango->GetHandWheel(1, &HandWheelOn, &PositionCount, &Encoder);

#### 4.7.4 GetJoystick

<b>Description</b>	Retrieves analog Joystick status.
<b>C++</b>	<pre>int LSX_GetJoystick (int ILSID,     BOOL *pbJoystickOn,     BOOL *pbManual,     BOOL *pbPositionCount,     BOOL *pbEncoder);</pre>
<b>Parameters</b>	<p><b>JoystickOn:</b> TRUE = Joystick switched on</p> <p><b>Manual:</b> FALSE = Joystick switch set on automatic TRUE = Joystick is switched on manually via switch</p> <p><b>PositionCount:</b> TRUE = position count switched on</p> <p><b>Encoder:</b> TRUE = encoder values, if available</p>
<b>Example</b>	pTango->GetJoystick(1, &JoystickOn, &Manual, &PositionCount, &Encoder);

#### 4.7.5 GetJoystickDir

<b>Description</b>	Retrieves axis direction for the analog Joystick and other HDI input devices.
<b>C++</b>	<pre>int LSX_GetJoystickDir (int ILSID, int *plXD, int *plYD, int *plZD, int *plAD);</pre>
<b>Parameters</b>	<p><b><i>XD, YD, ZD, AD:</i></b></p> <p>0 □ Axis disabled for Joystick (deflection ignored)</p> <p>1 □ positive axis direction, current reduction disabled</p> <p>-1 □ negative axis direction, current reduction disabled</p> <p>2 □ positive axis direction with current reduction (default)</p> <p>-2 □ negative axis direction with current reduction</p>
<b>Example</b>	pTango->GetJoystickDir(1, &XD, &YD, &ZD, &AD);

4.7.6 SetJoystickDir	
<b>Description</b>	Sets axis direction for Joystick and other HDI input devices.
<b>C++</b>	int LSX_SetJoystickDir (int ILSID, int IXD, int IYD, int IZD, int IAD);
<b>Parameters</b>	<b><i>XD, YD, ZD, AD:</i></b> 0 □ Axis disabled for Joystick (deflection ignored) 1 □ positive axis direction, current reduction disabled -1 □ negative axis direction, current reduction disabled 2 □ positive axis direction with current reduction (default) -2 □ negative axis direction with current reduction
<b>Example</b>	pTango->SetJoystickDir(1, 1, 1, -1, 0); <i>// X- and Y-Axis positive direction, Z-Axis negative direction, A-Axis blocked</i>

4.7.7 GetJoystickWindow	
<b>Description</b>	Retrieves Joystick idle window.
<b>C++</b>	int LSX_GetJoystickWindow (int ILSID, int *pIAValue);
<b>Parameters</b>	<b><i>AValue:</i></b> Analogoue signal range (as digits) in which axes do not move.
<b>Example</b>	pTango->GetJoystickWindow(1, &AValue);

4.7.8 SetJoystickWindow	
<b>Description</b>	Set Joystick idle window. A value in digits which configures an angle where a analog Joystick deflection has no effect. Used to compensate for mechanical and signal noise effects which else would cause a minor motion of the axes.
<b>C++</b>	int LSX_SetJoystickWindow (int ILSID, int IAValue);
<b>Parameters</b>	<b><i>AValue:</i></b> Analogoue signal range (as digits) in which axes do not move. 0 ... 100
<b>Example</b>	pTango->SetJoystickWindow(1, 30);

4.7.9 SetHandWheelOff	
<b>Description</b>	Switch handwheel off.
<b>C++</b>	int LSX_SetHandWheelOff (int ILSID);
<b>Parameters</b>	-
<b>Example</b>	pTango->SetHandWheelOff(1);

#### 4.7.10 SetHandWheelOn

<b>Description</b>	Switch handwheel on.
<b>C++</b>	int LSX_SetHandWheelOn (int ILSID, BOOL bPositionCount, BOOL bEncoder);
<b>Parameters</b>	<b>PositionCount</b> = TRUE □ position counter on = FALSE □ position counter off <b>Encoder</b> = TRUE □ encoder values, if encoders available
<b>Example</b>	pTango->SetHandWheelOn(1, TRUE, TRUE); <i>// switch on handwheel with position count (encoder values)</i>

#### 4.7.11 SetJoystickOff

<b>Description</b>	Switch analog Joystick off.
<b>C++</b>	int LSX_SetJoystickOff (int ILSID);
<b>Parameters</b>	-
<b>Example</b>	pTango->SetJoystickOff(1);

#### 4.7.12 SetJoystickOn

<b>Description</b>	Switch analog Joystick on.
<b>C++</b>	int LSX_SetJoystickOn (int ILSID, BOOL bPositionCount, BOOL bEncoder);
<b>Parameters</b>	<b>PositionCount</b> = TRUE □ position count on = FALSE □ position count off <b>Encoder</b> = TRUE □ encoder values, if encoders available
<b>Example</b>	pTango->SetJoystickOn(1, TRUE, TRUE); <i>// switch on joystick with position count (encoder values)</i>

#### 4.7.13 GetHwFactor

<b>Description</b>	Read handwheel factor of all axes, in [mm per knob rotation]
<b>C++</b>	int LSX_GetHwFactor (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);
<b>Parameters</b>	Pointer to double
<b>Example</b>	pTango->GetHwFactor(1, &dX, &dY, &dZ, &dA);



#### 4.7.14 SetHwFactor

<b>Description</b>	Set handwheel factor for all axes, in [mm per knob rotation]
<b>C++</b>	int LSX_SetHwFactor (int ILSID, double dX, double dY, double dZ, double dA)
<b>Parameters</b>	Double values
<b>Example</b>	pTango->SetHwFactor(1, dX, dY, dZ, dA);

#### 4.7.15 GetHwFactorB

<b>Description</b>	Read second handwheel factor of all axes, in [mm per knob rotation]
<b>C++</b>	int LSX_GetHwFactorB (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);
<b>Parameters</b>	Pointer to double
<b>Example</b>	pTango->GetHwFactorB(1, &dX, &dY, &dZ, &dA);

#### 4.7.16 SetHwFactorB

<b>Description</b>	Set second handwheel factor for all axes, in [mm per knob rotation]
<b>C++</b>	int LSX_SetHwFactorB (int ILSID, double dX, double dY, double dZ, double dA)
<b>Parameters</b>	Double values
<b>Example</b>	pTango->SetHwFactorB(1, dX, dY, dZ, dA);

#### 4.7.17 GetZwTravel

<b>Description</b>	Read z-wheel travel distances, in [mm per knob rotation]
<b>C++</b>	int LSX_GetZwTravel (int ILSID, int lIndex, double *pdDistance);
<b>Parameters</b>	lIndex: 1: Get setting for standard distance 2: Get setting for slow distance 3: Get setting for fast distance dDistance: Pointer to double
<b>Example</b>	pTango-> GetZwTravel (1, lIndex, &dDistance);

#### 4.7.18 SetZwTravel

<b>Description</b>	Set z-wheel travel distances, in [mm per knob rotation]
<b>C++</b>	int LSX_SetZwTravel (int ILSID, int lIndex, double dDistance);
<b>Parameters</b>	lIndex: 1: Set standard distance 2: Set slow distance 3: Set fast distance dDistance: Double value
<b>Example</b>	pTango-> SetZwTravel (1, lIndex, dDistance);

#### 4.7.19 GetKey

<b>Description</b>	Get HDI device key states
<b>C++</b>	int LSX_GetKey (int ILSID, BOOL *pbKey1, BOOL *pbKey2, BOOL *pbKey3, BOOL *pbKey4);
<b>Parameters</b>	Pointers to BOOL, TRUE=Key pressed
<b>Example</b>	pTango-> GetKey(1, &bKey[0], &bKey[1], &bKey[2], &bKey[3]);

#### 4.7.20 GetKeyLatch

<b>Description</b>	Get and clear HDI device key states
<b>C++</b>	int LSX_GetKeyLatch (int ILSID, BOOL *pbKey1, BOOL *pbKey2, BOOL *pbKey3, BOOL *pbKey4);
<b>Parameters</b>	Pointers to BOOL, TRUE=Key was or is pressed
<b>Example</b>	pTango-> GetKeyLatch(1, &bKey[0], &bKey[1], &bKey[2], &bKey[3]);

#### 4.7.21 ClearKeyLatch

<b>Description</b>	Clear latched key state(s)
<b>C++</b>	int LSX_ClearKeyLatch (int ILSID, int lKey);
<b>Parameters</b>	lKey: 0 = clear latched keystate of all 4 keys 1 = clear latched keystate of key 1 only 2 = clear latched keystate of key 2 only 3 = clear latched keystate of key 3 only 4 = clear latched keystate of key 4 only
<b>Example</b>	pTango-> ClearKeyLatch(1, 0); // Clear all

## 4.8. Control Console with Trackball and Joyspeed Keys

4.8.1 GetBPZ	
<b>Description</b>	Retrieves status of a custom-built control console with trackball.
<b>C++</b>	int LSX_GetBPZ (int ILSID, int *pIValue);
<b>Parameters</b>	<i>AValue</i> : 0 □ control console is "OFF" 1 □ control console active, trackball operated at 0,1µm step resolution. 2 □ control console active, trackball operated with trackball factor.
<b>Example</b>	pTango->GetBPZ(1, &AValue);

4.8.2 SetBPZ	
<b>Description</b>	Switches custom-built control console on / off.
<b>C++</b>	int LSX_SetBPZ (int ILSID, int IValue);
<b>Parameters</b>	<i>AValue</i> : 0...2 0 □ control console is "OFF" 1 □ activate control console and operate trackball at 0,1µm step resolution. 2 □ activate control console and operate trackball with trackball factor.
<b>Example</b>	pTango->SetBPZ(1, 1);

4.8.3 GetBPZJoyspeed	
<b>Description</b>	Retrieves custom-built control console Joystick speed.
<b>C++</b>	int LSX_GetBPZJoyspeed (int ILSID, int IAPar, double *pdAValue);
<b>Parameters</b>	<i>APar</i> : 1, 2 or 3 (console keys for speed selection: slow, medium, fast) <i>AValue</i> : max. speed [r/sec]
<b>Example</b>	pTango->GetBPZJoyspeed(1, &AValue); <i>// retrieve set speed of key 1 (slow)</i>

#### 4.8.4 SetBPZJoyspeed

<b>Description</b>	Set custom-built control console joystick speed.
<b>C++</b>	<code>int LSX_SetBPZJoyspeed (int ILSID, int IAPar, double dAValue);</code>
<b>Parameters</b>	<i>APar</i> : 1, 2 or 3 (console keys for speed selection: slow, medium, fast) <i>AValue</i> : $\pm$ max. speed [r/sec]
<b>Example</b>	<code>pTango-&gt;SetBPZJoyspeed(1, 1, 25);</code> <i>// Set key 1 parameter (slow) to speed 25</i>

#### 4.8.5 GetBPZTrackballBackLash

<b>Description</b>	Retrieves custom-built control console trackball backlash.
<b>C++</b>	<code>int LSX_GetBPZTrackballBackLash (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);</code>
<b>Parameters</b>	<i>X, Y, Z, A</i> : backlash [mm]
<b>Example</b>	<code>pTango-&gt;GetBPZTrackballBackLash(1, &amp;X, &amp;Y, &amp;Z, &amp;A);</code>

#### 4.8.6 SetBPZTrackballBackLash

<b>Description</b>	Set custom-built control console trackball backlash.
<b>C++</b>	<code>int LSX_SetBPZTrackballBackLash (int ILSID, double dX, double dY, double dZ, double dA);</code>
<b>Parameters</b>	<i>X, Y, Z, A</i> : 0.001 to 0.15 mm
<b>Example</b>	<code>pTango-&gt;SetBPZTrackballBackLash(1, 0.01, 0.01, 0.01, 0.01);</code> <i>// Set backlash for all axes to 10<math>\mu</math>m</i>

#### 4.8.7 GetBPZTrackballFactor

<b>Description</b>	Retrieves control console trackball factor.
<b>C++</b>	<code>int LSX_GetBPZTrackballFactor (int ILSID, double *pdAValue);</code>
<b>Parameters</b>	<i>AValue</i> : Trackball factor e.g. AValue of 3 means that one trackball pulse results in 3 motor increments.
<b>Example</b>	<code>pTango-&gt;GetBPZTrackballFactor(1, &amp;AValue);</code>

#### 4.8.8 SetBPZTrackballFactor

<b>Description</b>	Set custom-built control console trackball factor.
<b>C++</b>	<code>int LSX_SetBPZTrackballFactor (int lLSID, double dAValue);</code>
<b>Parameters</b>	<i>AValue</i> : 0.01 ... 100 AValue = 1 $\square$ Trackball factor = 1, i.e. one trackball impulse results in one motor increment
<b>Example</b>	<code>pTango-&gt;SetBPZTrackballFactor(1, 1,0);</code>

## 4.9. Limit Switches (Hardware and Software)

4.9.1 GetAutoLimitAfterCalibRM	
<b>Description</b>	Provides, whether internal software limits are set when calibrating (cal) or measuring stage travel range (rm).
<b>C++</b>	int LSX_GetAutoLimitAfterCalibRM (int ILSID, int *plFlags);
<b>Parameters</b>	<b>Flags:</b> Bit mask: Bit0=X, Bit1=Y, Bit2=Z, Bit3=A Bit 0 = 1 <input type="checkbox"/> no travel range limits are set from X-Axis calibration or range measure Bit 1 = 0 <input type="checkbox"/> software limits are set for Y-Axis (cal/rm)
<b>Example</b>	pTango->GetAutoLimitAfterCalibRM(1, &Flags);

4.9.2 SetAutoLimitAfterCalibRM	
<b>Description</b>	Prevents setting of internal software limits when calibrating or measuring travel range.
<b>C++</b>	int LSX_SetAutoLimitAfterCalibRM (int ILSID, int lFlags);
<b>Parameters</b>	<b>Flags:</b> Bit mask: Bit0=X, Bit1=Y, Bit2=Z, Bit3=A Bit 0 = 1 <input type="checkbox"/> no travel range limits are set from X-Axis calibration or range measure Bit 1 = 0 <input type="checkbox"/> software limits are set for Y-Axis (cal/rm)
<b>Example</b>	pTango->SetAutoLimitAfterCalibRM(1, Flags);

4.9.3 GetLimit	
<b>Description</b>	Provides soft travel range limits.
<b>C++</b>	int LSX_GetLimit (int ILSID, int lAxis, double *pdMinRange, double *pdMaxRange);
<b>Parameters</b>	<b>Axis:</b> Axis from which travel range limits are to be retrieved (X, Y, Z, A numbered from 1=X to 4=A) <b>MinRange:</b> lower travel range limit, unit depends on dimension <b>MaxRange:</b> upper travel range limit, unit depends on dimension
<b>Example</b>	pTango->GetLimit(1, &MinRange, &MaxRange);

#### 4.9.4 SetLimit

<b>Description</b>	Set soft travel range limits.
<b>C++</b>	<code>int LSX_SetLimit (int ILSID, int lAxis, double dMinRange, double dMaxRange);</code>
<b>Parameters</b>	<p><b>Axis:</b> Axis from which travel range limits are to be retrieved (X, Y, Z, A numbered from 1=X to 4=A)</p> <p><b>MinRange:</b> lower travel range limit, unit depends on dimension</p> <p><b>MaxRange:</b> upper travel range limit, unit depends on dimension</p>
<b>Example</b>	<pre>pTango-&gt;SetLimit(1, 1, -10.0, 20.0); // assign X-Axis -10 as lower and 20 as upper travel range limits</pre>

#### 4.9.5 GetLimitControl

<b>Description</b>	Retrieves, whether area control (limits) is switched on or off.
<b>C++</b>	<code>int LSX_GetLimitControl (int ILSID, int lAxis, BOOL *pbActive);</code>
<b>Parameters</b>	<p><b>Axis:</b> X, Y, Z and A, numbered from 1=X to 4=A</p> <p><b>Active:</b> TRUE = area control of corresponding axis is active FALSE = area control of corresponding axis is deactivated</p>
<b>Example</b>	<code>pTango-&gt;GetLimitControl(1, 2, &amp;Active);</code>

#### 4.9.6 SetLimitControl

<b>Description</b>	Switches area control on / off.
<b>C++</b>	<code>int LSX_SetLimitControl (int ILSID, int lAxis, BOOL bActive);</code>
<b>Parameters</b>	<p><b>Axis:</b> X, Y, Z and A, numbered from 1=X to 4=A</p> <p><b>Active:</b> TRUE = activate area control of corresponding axis FALSE = disable area control of corresponding axis</p>
<b>Example</b>	<code>pTango-&gt;SetLimitControl(1, 2, TRUE); // Area control of Y-Axis is active</code>

4.9.7 GetSwitchActive	
<b>Description</b>	Provides, whether hardware limit switches are enabled.
<b>C++</b>	int LSX_GetSwitchActive (int ILSID, int *plXA, int *plYA, int *plZA, int *plAA);
<b>Parameters</b>	<p>A bit mask is supplied for each axis:</p> <p>Bit 0 <input type="checkbox"/> zero limit switch (cal, "E0")</p> <p>Bit 1 <input type="checkbox"/> reference limit switch (unused)</p> <p>Bit 2 <input type="checkbox"/> end limit switch (rm, "EE")</p> <p>The limit switch is enabled if the corresponding bit is set.</p>
<b>Example</b>	pTango->GetSwitchActive(1, &XA, &YA, &ZA, &AA);

4.9.8 SetSwitchActive	
<b>Description</b>	Switches limit switches on / off.
<b>C++</b>	int LSX_SetSwitchActive (int ILSID, int lXA, int lYA, int lZA, int lAA);
<b>Parameters</b>	<p>A bit mask is supplied for each axis:</p> <p>Bit 0 <input type="checkbox"/> zero limit switch (cal, "E0")</p> <p>Bit 1 <input type="checkbox"/> reference limit switch (unused)</p> <p>Bit 2 <input type="checkbox"/> end limit switch (rm, "EE")</p> <p>The limit switch is enabled if the corresponding bit is set.</p>
<b>Example</b>	<p>pTango-&gt;SetSwitchActive(1, 7, 1, 5, 0);</p> <p><i>// X-Axis: All limit switches enabled, Y-Axis: Only Zero limit switch enabled,</i></p> <p><i>// Z-Axis: E0 and EE switches enabled (default,) A-Axis: All limit switches ignored</i></p>

4.9.9 GetSwitches															
Description	Retrieves actuation status of all limit switches.														
C++	int LSX_GetSwitches (int ILSID, int *plFlags);														
Parameters	<p><b>Flags:</b> Pointer on Integer Value, which includes status of all limit switches as bit mask</p> <p>In bit mask, status of limit switches is encoded as follows:</p> <table><tr><td>Limit switch</td><td>EE (rm)Ref.</td><td colspan="2">E0 (cal)</td></tr><tr><td>Axis</td><td>AZYX</td><td>AZYX</td><td>AZYX</td></tr><tr><td>Bit</td><td>0000</td><td>0000</td><td>0000</td></tr></table> <p>E.g.:</p> <p>Flags = 0x003 □ E0 of X- and Y-Axis are actuated</p> <p>Flags = 0x200 □ EE of Y-Axis is actuated</p>			Limit switch	EE (rm)Ref.	E0 (cal)		Axis	AZYX	AZYX	AZYX	Bit	0000	0000	0000
Limit switch	EE (rm)Ref.	E0 (cal)													
Axis	AZYX	AZYX	AZYX												
Bit	0000	0000	0000												
Example	pTango->GetSwitches(1, &Flags);														



#### 4.9.10 GetSwitchPolarity

<b>Description</b>	Retrieves polarity of limit switches.
<b>C++</b>	<code>int LSX_GetSwitchPolarity (int ILSID, int *plXP, int *plYP, int *plZP, int *plAP);</code>
<b>Parameters</b>	<p>A bit mask is supplied for each axis:</p> <p>Bit 0 <input type="checkbox"/> zero limit switch (cal, "E0")</p> <p>Bit 1 <input type="checkbox"/> reference limit switch (unused)</p> <p>Bit 2 <input type="checkbox"/> end limit switch (rm, "EE")</p> <p>If bit is set (1), the corresponding switch is interpreted active when high.</p> <p>If bit is reset (0), the corresponding switch is active low.</p>
<b>Example</b>	<code>pTango-&gt;GetSwitchPolarity(1, &amp;XP, &amp;YP, &amp;ZP, &amp;AP);</code>

#### 4.9.11 SetSwitchPolarity

<b>Description</b>	Sets polarity of limit switches.
<b>C++</b>	<code>int LSX_SetSwitchPolarity (int ILSID, int IXP, int IYP, int IZP, int IAP);</code>
<b>Parameters</b>	<p>A bit mask is supplied for each axis:</p> <p>Bit 0 <input type="checkbox"/> zero limit switch (cal, "E0")</p> <p>Bit 1 <input type="checkbox"/> reference limit switch (unused)</p> <p>Bit 2 <input type="checkbox"/> end limit switch (rm, "EE")</p> <p>If bit is set (1), the corresponding switch is interpreted active when high.</p> <p>If bit is reset (0), the corresponding switch is active low.</p>
<b>Example</b>	<code>pTango-&gt;SetSwitchPolarity(1, 7, 0, 0, 0);</code> <i>// all limit switches of X-Axis are high active,</i> <i>all limit switches of Y-, Z- and A-Axis are low active</i>

#### 4.9.12 GetSwitchType

<b>Description</b>	Retrieves type of limit switches.
<b>C++</b>	<code>int LSX_GetSwitchType (int ILSID, int *plXP, int *plYP, int *plZP, int *plAP);</code>
<b>Parameters</b>	<p>A bit mask is supplied for each axis:</p> <p>Bit 0 <input type="checkbox"/> zero limit switch (cal, "E0")</p> <p>Bit 1 <input type="checkbox"/> reference limit switch (unused)</p> <p>Bit 2 <input type="checkbox"/> end limit switch (rm, "EE")</p> <p>If bit is set (1), input is for NPN type limit switch.</p> <p>If bit is reset (0), input is for PNP type limit switch (default).</p>
<b>Example</b>	<code>pTango-&gt;GetSwitchType(1, &amp;XP, &amp;YP, &amp;ZP, &amp;RP);</code>

### 4.9.13 SetSwitchType

<b>Description</b>	Sets type of limit switches.
<b>C++</b>	<code>int LSX_SetSwitchType (int ILSID, int IXP, int IYP, int IZP, int IAP);</code>
<b>Parameters</b>	<p>A bit mask is supplied for each axis:</p> <p>Bit 0 <input type="checkbox"/> zero limit switch (cal, "E0")</p> <p>Bit 1 <input type="checkbox"/> reference limit switch (unused)</p> <p>Bit 2 <input type="checkbox"/> end limit switch (rm, "EE")</p> <p>If bit is set (1), input is configured for NPN type limit switch using pull-up resistor.</p> <p>If bit is reset (0), input is configured for for PNP type limit switch with pull down resistor (default).</p>
<b>Example</b>	<code>pTango-&gt;SetSwitchType(1, XP, YP, ZP, AP);</code>

## 4.10. Digital and Analog Inputs and Outputs

### 4.10.1 GetAnalogInput

<b>Description</b>	Retrieves current A/D conversion result of an analogue channel.
<b>C++</b>	<code>int LSX_GetAnalogInput (int ILSID, int IIndex, int *pIValue);</code>
<b>Parameters</b>	<p><b>Index:</b> 0...15 (analog channel),  0...9 = HDI connector, pins 1...10  10 = ANAIN0 of AUX-IO connector</p> <p><b>Value:</b> Pointer to Integer value, to which the channel's A/D conversion result is written.  0...5V analog = 0...1023</p>
<b>Example</b>	<code>pTango-&gt;GetAnalogInput(1, 0, &amp;Input); // Read channel 0</code>

### 4.10.2 GetDigitalInputs

<b>Description</b>	Retrieve signal level of all 16 digital input pins (I/O extension).
<b>C++</b>	<code>int LSX_GetDigitalInputs (int ILSID, int *pIValue);</code>
<b>Parameters</b>	<b>Value:</b> Pointer to Integer value, to which the status of all inputs is written (as bit mask). LSB = Digital input 0
<b>Example</b>	<pre>int inputs; pTango-&gt;GetDigitalInputs(1, &amp;inputs); if (Inputs &amp; 16) ... // if input 4 is set ...</pre>

### 4.10.3 GetDigitalInputsE

<b>Description</b>	Retrieve signal level of additional digital inputs (16...31).
<b>C++</b>	<code>int LSX_GetDigitalInputsE (int ILSID, int *pIValue);</code>
<b>Parameters</b>	<b>Value:</b> Pointer on a 32-Bit Integer, which returns the inputs 16...31 in the bits 0...15
<b>Example</b>	<pre>int ext_inputs; pTango-&gt;GetDigitalInputsE(1, &amp;ext_inputs);</pre>

### 4.10.4 SetAnalogOutput

<b>Description</b>	Set analog output signals.
<b>C++</b>	<code>int LSX_SetAnalogOutput (int ILSID, int IIndex, int IValue);</code>
<b>Parameters</b>	<p><b>Index:</b> 0,1 (analog circuits)</p> <p><b>Value:</b> 0...100 [%]</p>
<b>Example</b>	<pre>pTango-&gt;SetAnalogOutput(1, 0, 100); // set analog output 0 to max. voltage (10V)</pre>

#### 4.10.5 SetDigIO\_Distance

<b>Description</b>	Function of digital inputs / outputs. Activate an output depending on preset distance before or after reaching designated position.
<b>C++</b>	int LSX_SetDigIO_Distance (int ILSID, int IIndex, BOOL bFkt, double dDist, int IAxis);
<b>Parameters</b>	<b>Index:</b> 0 to 15 (output pin) <b>Fkt</b> = FALSE    □    activation of an output depending on set distance before reaching determined position <b>Fkt</b> = TRUE    □    activation of an output depending on set distance after start position <b>Dist:</b> Distance, depends on selected dimension (unit) <b>Axis:</b> X, Y, Z and A, numbered from 1 to 4
<b>Example</b>	pTango->SetDigIO_Distance(1, 7, FALSE, 78.9, 3); // output 7 is activated 78.9mm before reaching final position (Z-Axis)

#### 4.10.6 SetDigIO\_EmergencyStop

<b>Description</b>	Function of digital inputs / outputs. Assignment of Emergency-Stop pin functionality.
<b>C++</b>	int LSX_SetDigIO_EmergencyStop (int ILSID, int IIndex);
<b>Parameters</b>	<b>Index:</b> 0 to 15 (input/output)
<b>Example</b>	pTango->SetDigIO_EmergencyStop(1, 15); // Pin 15 is used for Emergency-Stop

#### 4.10.7 SetDigIO\_Off

<b>Description</b>	Switch off digital inputs / outputs function. (Does not affect inputs / outputs states).
<b>C++</b>	int LSX_SetDigIO_Off (int ILSID, int IIndex);
<b>Parameters</b>	Index: 0 to 15 (individual Input/Output pins), 16 (all 16 port pins)
<b>Example</b>	pTango->SetDigIO_Off(1, 0); // Function of I/O pin 0 is switched 'Off'

#### 4.10.8 SetDigIO\_Polarity

<b>Description</b>	Set polarity of digital inputs / outputs.
<b>C++</b>	int LSX_SetDigIO_Polarity (int ILSID, int IIndex, BOOL bHigh);
<b>Parameters</b>	<b>Index:</b> 0 to 15 (individual I/O pin), 16 (all 16 port pins) <b>High</b> = TRUE □ high active <b>High</b> = FALSE □ low active
<b>Example</b>	pTango->SetDigIO_Polarity(1, 3, TRUE); <i>// input pin / output pin 3 high active</i>

#### 4.10.9 SetDigitalOutput

<b>Description</b>	Set individual digital output pin.
<b>C++</b>	int LSX_SetDigitalOutput (int ILSID, int IIndex, BOOL bValue);
<b>Parameters</b>	<b>Index:</b> 0 to 15 <b>Value:</b> Set pin level to FALSE = low TRUE = high
<b>Example</b>	pTango->SetDigitalOutput(1, 0, TRUE); <i>// set output pin 0 to '1'</i>

#### 4.10.10 SetDigitalOutputs

<b>Description</b>	Set all digital output pins (0-15).
<b>C++</b>	int LSX_SetDigitalOutputs (int ILSID, int IValue);
<b>Parameters</b>	<b>Value:</b> Bit mask, bits 0-15 determine value that is set for outputs 0-15
<b>Example</b>	pTango->SetDigitalOutputs(1, 3); <i>// set outputs 0 and 1 to 1, remaining pins to 0</i>

#### 4.10.11 SetDigitalOutputsE

<b>Description</b>	Set additional digital outputs (16-31).
<b>C++</b>	int LSX_SetDigitalOutputsE (int ILSID, int IValue);
<b>Parameters</b>	<b>Value:</b> Bit mask, bits 0-15 determine value that is set for outputs 16-31
<b>Example</b>	pTango->SetDigitalOutputsE(1, 3); <i>// set outputs 16 and 17 to 1, remaining pins to 0</i>

## 4.11. Encoder Settings

### 4.11.1 ClearEncoder

<b>Description</b>	Reset encoder positions to zero.
<b>C++</b>	int LSX_ClearEncoder (int ILSID, int lAxis);
<b>Parameters</b>	<i>Axis</i> : X, Y, Z and A, numbered from 1 to 4
<b>Example</b>	pTango->ClearEncoder(1, 2); <i>// reset encoder counter of Y-Axis to zero</i>

### 4.11.2 GetEncoder

<b>Description</b>	Retrieves all encoder positions.
<b>C++</b>	int LSX_GetEncoder (int ILSID, double *pdXP, double *pdYP, double *pdZP, double *pdAP);
<b>Parameters</b>	<i>XP, YP, ZP, AP</i> : Counter values, 4x interpolated
<b>Example</b>	pTango->GetEncoder(1, &XP, &YP, &ZP, &AP);

### 4.11.3 GetEncoderActive

<b>Description</b>	Retrieves which encoder will be activated after calibration.  Please note: This function is corresponding to the „?encmask“ command!
<b>C++</b>	int LSX_GetEncoderActive (int ILSID, int *plFlags);
<b>Parameters</b>	<i>Flags</i> : Encoder mask (flags) Bit 0 = X encoder will be activated Bit 1 = Y encoder will be activated Bit 2 = Z encoder will be activated
<b>Example</b>	pTango->GetEncoderActive(1, &Flags);

#### 4.11.4 SetEncoderActive

<b>Description</b>	Retrieves which encoder is activated after calibration  Please note: This function is corresponding to „!encmask“ command.
<b>C++</b>	int LSX_SetEncoderActive (int ILSID, int IFlags);
<b>Parameters</b>	<b>Value:</b> Encoder mask (flags) Bit 0 = X encoder will be activated Bit 1 = Y encoder will be activated Bit 2 = Z encoder will be activated
<b>Example</b>	pTango->SetEncoderActive(1, 0); <i>// No encoder will be used</i> pTango->SetEncoderActive(1, 2); <i>// encoder of Y-Axis will be activated after calibration</i>

#### 4.11.5 GetEncoderMask

<b>Description</b>	Retrieve status of encoders.  Please note: This function is corresponding to „?enc“ command.
<b>C++</b>	LSX_GetEncoderMask (int ILSID, int *plFlags);
<b>Parameters</b>	<b>Flags:</b> Active encoder mask (flags) Bit 0 = X encoder is active / inactive Bit 1 = Y encoder is active / inactive Bit 2 = Z encoder is active / inactive
<b>Example</b>	int EncMask; pTango->GetEncoderMask(1, &EncMask); if (EncMask & 2) ... <i>// if encoder of Y-Axis connected + active ...</i>

#### 4.11.6 SetEncoderMask

<b>Description</b>	Activates / deactivates encoders manually.  Please note: This function is corresponding to „!enc“ command. Do not use in closed loop. Encoders should always be activated with Calibrate command.
<b>C++</b>	int LSX_SetEncoderMask (int ILSID, int IValue);
<b>Parameters</b>	<b>Value:</b> Active encoder mask (flags) Bit 0 = (activate)/deactivate X encoder Bit 1 = (activate)/deactivate Y encoder Bit 2 = (activate)/deactivate Z encoder
<b>Example</b>	pTango->SetEncoderMask(1, 0); <i>// deactivate all encoders</i> pTango->SetEncoderMask (1, 2); <i>// deactivate X and Z encoders, activate Y-Axis encoder</i>

#### 4.11.7 GetEncoderPeriod

<b>Description</b>	Retrieves encoder signal period length.
<b>C++</b>	<pre>int LSX_GetEncoderPeriod (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);</pre>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Period length [mm]
<b>Example</b>	<pre>pTango-&gt;GetEncoderPeriod(1, &amp;X, &amp;Y, &amp;Z, &amp;A);</pre>



#### 4.11.8 SetEncoderPeriod

<b>Description</b>	Set encoder signal period length.
<b>C++</b>	<code>int LSX_SetEncoderPeriod (int ILSID, double dX, double dY, double dZ, double dA);</code>
<b>Parameters</b>	<i>X, Y, Z, A:</i> 0.0001 - 4 mm
<b>Example</b>	<code>pTango-&gt;SetEncoderPeriod(1, 0.5, 0.5, 0.5, 0.5);</code> <i>// encoder signal period of all axes is set to 0.5mm</i>

#### 4.11.9 GetEncoderPosition

<b>Description</b>	Retrieves position response type.
<b>C++</b>	<code>int LSX_GetEncoderPosition (int ILSID, BOOL *pbValue);</code>
<b>Parameters</b>	<i>Value:</i> TRUE <input type="checkbox"/> axis position values will be read from the encoder, if activated. Else the position will be taken from the motor position. FALSE <input type="checkbox"/> Position will be taken from the motor position.
<b>Example</b>	<code>pTango-&gt;GetEncoderPosition(1, &amp;Value);</code>

#### 4.11.10 SetEncoderPosition

<b>Description</b>	Switches encoder value display on / off.
<b>C++</b>	<code>int LSX_SetEncoderPosition (int ILSID, BOOL bValue);</code>
<b>Parameters</b>	<i>Value:</i> TRUE <input type="checkbox"/> axis position values will be read from the encoder, if activated. Else the position will be taken from the motor position. FALSE <input type="checkbox"/> Position will be taken from the motor position.
<b>Example</b>	<code>pTango-&gt;SetEncoderPosition(1, TRUE);</code>

#### 4.11.11 GetEncoderRefSignal

<b>Description</b>	Retrieves whether the encoder reference signal is evaluated when calibrating.
<b>C++</b>	<code>int LSX_GetEncoderRefSignal (int ILSID, int *plXR, int *plYR, int *plZR, int *plAR);</code>
<b>Parameters</b>	1 <input type="checkbox"/> encoder reference signal is evaluated while calibrating 0 <input type="checkbox"/> reference signal is not evaluated, zero position is set at the CAL end switch
<b>Example</b>	<code>pTango-&gt;GetEncoderRefSignal(1, &amp;X, &amp;Y, &amp;Z, &amp;A);</code>

#### 4.11.12 SetEncoderRefSignal

<b>Description</b>	Evaluate reference signal from encoder when calibrating.
<b>C++</b>	<code>int LSX_SetEncoderRefSignal (int ILSID, int IXR, int IYR, int IZR, int IAR);</code>
<b>Parameters</b>	<b><i>XR, YR, ZR, AR</i></b> : 0 (encoder reference signal is evaluated while calibrating) or 1 (reference signal is not evaluated, zero position is set at the CAL end switch)
<b>Example</b>	<code>pTango-&gt;SetEncoderRefSignal(1, 1, 1, 0, 0);</code> <i>// when calibrating, reference signals of encoders X and Y are evaluated</i>

## 4.12. Closed Loop Settings

### 4.12.1 ClearCtrFastMoveCounter

<b>Description</b>	If position difference is larger than lock-in range, a new vector will be started and corresponding counter will be increased by one.
<b>C++</b>	int LSX_ClearCtrFastMoveCounter (int ILSID);
<b>Parameters</b>	-
<b>Example</b>	pTango->ClearCtrFastMoveCounter(1);

### 4.12.2 GetController

<b>Description</b>	Retrieve Closed Loop mode.
<b>C++</b>	int LSX_GetController (int ILSID, int *plXC, int *plYC, int *plZC, int *plRC);
<b>Parameters</b>	<b>Controller mode XC, YC, ZC, AC:</b> 0 □ controller "OFF" 1 □ controller "OFF after reaching target position" 2 □ controller "Always ON" 3 □ controller "OFF after reaching designated end position" with current reduction 4 □ controller "Always ON" with current reduction
<b>Example</b>	pTango->GetController(1, &X, &Y, &Z, &A);

### 4.12.3 SetController

<b>Description</b>	Set Closed Loop mode.
<b>C++</b>	int LSX_SetController (int ILSID, int IXC, int IYC, int IZC, int IAC);
<b>Parameters</b>	<b>Controller mode XC, YC, ZC, AC:</b> 0 □ controller "OFF" 1 □ controller "OFF after reaching target position" 2 □ controller "Always ON" 3 □ controller "OFF after reaching designated end position" with current reduction 4 □ controller "Always ON" with current reduction
<b>Example</b>	pTango->SetController(1, 2, 2, 0, 0); // Enable permanent closed loop for X and Y axes

#### 4.12.4 GetControllerCall

<b>Description</b>	Provides Closed Loop interval time.
<b>C++</b>	int LSX_GetControllerCall (int ILSID, int *plCtrCall);
<b>Parameter:</b>	<i>CtrCall</i> : Controller call time [ms]
<b>Example</b>	pTango->GetControllerCall(1, &CtrCall);

#### 4.12.5 SetControllerCall

<b>Description</b>	Set Closed Loop interval time.
<b>C++</b>	int LSX_SetControllerCall (int ILSID, int lCtrCall);
<b>Parameters</b>	<i>CtrCall</i> : Controller call time [ms]
<b>Example</b>	pTango->SetControllerCall(1, 5); <i>// CtrCall = 5 means: Closed Loop controller is called every 5 milliseconds</i>

#### 4.12.6 GetControllerFactor

<b>Description</b>	Retrieve Closed Loop controller factors.
<b>C++</b>	int LSX_GetControllerFactor (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);
<b>Parameters</b>	<i>X, Y, Z, A</i> : Closed Loop factors
<b>Example</b>	pTango->GetControllerFactor(1, &X, &Y, &Z, &A);

#### 4.12.7 SetControllerFactor

<b>Description</b>	Set Closed Loop controller factor.
<b>C++</b>	int LSX_SetControllerFactor (int ILSID, double dX, double dY, double dZ, double dA);
<b>Parameters</b>	<i>X, Y, Z, A</i> : Position difference amplification factor 1 - 64
<b>Example</b>	pTango->SetControllerFactor(1, 2, 2, 2, 0); <i>//Closed Loop amplification is set to 2 for X, Y and Z axes</i>

#### 4.12.8 GetControllerSteps

<b>Description</b>	Retrieves length of controller steps.
<b>C++</b>	<pre>int LSX_GetControllerSteps (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);</pre>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Length of controller steps [mm]
<b>Example</b>	<code>pTango-&gt;GetControllerSteps(1, &amp;X, &amp;Y, &amp;Z, &amp;A);</code>

#### 4.12.9 SetControllerSteps

<b>Description</b>	Set controller steps.
<b>C++</b>	<pre>int LSX_SetControllerSteps (int ILSID, double dX, double dY, double dZ, double dA);</pre>
<b>Parameters</b>	<i>X, Y, Z, A</i> : 1 - spindle pitch (values depend on dimension)
<b>Example</b>	<code>pTango-&gt;SetControllerSteps(1, 4, 5, 7, 9);</code>

#### 4.12.10 GetControllerTimeout

<b>Description</b>	Retrieves controller timeout.
<b>C++</b>	<pre>Int LSX_GetControllerTimeout (int ILSID, int *plACtrTimeout);</pre>
<b>Parameters</b>	<p><i>ACtrTimeout</i>: Timeout [ms],</p> <p>If the Closed Loop controller is unable to settle in the target window for this time, the move is aborted (move function calls return with error code 4013).</p>
<b>Example</b>	<code>pTango-&gt;GetControllerTimeout(1, &amp;ACtrTimeout);</code>

#### 4.12.11 SetControllerTimeout

<b>Description</b>	Set controller timeout.
<b>C++</b>	<pre>int LSX_SetControllerTimeout (int ILSID, int lACtrTimeout);</pre>
<b>Parameters</b>	<p><i>ACtrTimeout</i>: Timeout 0 – 10000 ms,</p> <p>If the Closed Loop controller is unable to settle in the target window for this time, the move is aborted (move function calls return with error code 4013). This time should be set longer than the target window delay (TWDelay).</p>
<b>Example</b>	<pre>pTango-&gt;SetControllerTimeout(1, 500); // Abort after trying to settle in the target window for 500ms</pre>

#### 4.12.12 GetControllerTWDelay

<b>Description</b>	Retrieve controller delay.
<b>C++</b>	<code>int LSX_GetControllerTWDelay (int ILSID, int *plCtrTWDelay);</code>
<b>Parameters</b>	<b><i>CtrTWDelay</i></b> : Controller delay [ms]
<b>Example</b>	<code>pTango-&gt;GetControllerTWDelay(1, &amp;CtrTWDelay);</code>

#### 4.12.13 SetControllerTWDelay

<b>Description</b>	Set controller delay.
<b>C++</b>	<code>int LSX_SetControllerTWDelay (int ILSID, int lCtrTWDelay);</code>
<b>Parameters</b>	<b><i>CtrTWDelay</i></b> : Controller delay 0 - 250 ms Time for which the axis has to remain in the target window. Moves are delayed by at least this time.
<b>Example</b>	<code>pTango-&gt;SetControllerTWDelay(1, 0);</code> <i>// controller delay switched off, closed loop end position will be inaccurate</i>

#### 4.12.14 GetCtrFastMove

<b>Description</b>	Retrieves setting of FastMove function.
<b>C++</b>	<code>int LSX_GetCtrFastMove (int ILSID, BOOL *pbActive);</code>
<b>Parameters</b>	<b><i>Active</i></b> : TRUE <input type="checkbox"/> FastMove function active
<b>Example</b>	<code>pTango-&gt;GetCtrFastMove(1, &amp;Active);</code>

#### 4.12.15 GetCtrFastMoveCounter

<b>Description</b>	If position difference is larger than lock-in range, a new vector will be started and corresponding counter will be increased by one. Function provides Fast Move counts.
<b>C++</b>	<code>int LSX_GetCtrFastMoveCounter (int ILSID, int *plXC, int *plYC, int *plZC, int *plAC);</code>
<b>Parameters</b>	<b><i>XC, YC, ZC, AC</i></b> : Number of carried out Fast Move functions
<b>Example</b>	<code>pTango-&gt;GetCtrFastMoveCounter(1, &amp;XC, &amp;YC,&amp;ZC,&amp;AC);</code>

#### 4.12.16 GetTargetWindow

<b>Description</b>	Retrieves closed loop target windows of all axes.
<b>C++</b>	<pre>int LSX_GetTargetWindow (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);</pre>
<b>Parameters</b>	<i>X, Y, Z, A</i> : Target window, depends on selected dimension
<b>Example</b>	<code>pTango-&gt;GetTargetWindow(1, &amp;X, &amp;Y, &amp;Z, &amp;A);</code>

#### 4.12.17 SetTargetWindow

<b>Description</b>	Set closed loop controller target windows. The closed loop controller has to settle within □ this window size for the specified delay time.
<b>C++</b>	<pre>int LSX_SetTargetWindow (int ILSID, double dX, double dY, double dZ, double dA);</pre>
<b>Parameters</b>	<i>X, Y, Z, A</i> : 1 - 25000 (motor increments) 0.1 - 1000 (μm) 0.0001 - 1 (mm) (values depend on dimension)
<b>Example</b>	<code>pTango-&gt;SetTargetWindow(1, 1.0, 0.001, 0.001, 0.0005);</code>

#### 4.12.18 SetCtrFastMoveOff

<b>Description</b>	FastMove function deactivated.
<b>C++</b>	<pre>int LSX_SetCtrFastMoveOff (int ILSID);</pre>
<b>Parameters</b>	-
<b>Example</b>	<code>pTango-&gt;SetCtrFastMoveOff(1);</code>

#### 4.12.19 SetCtrFastMoveOn

<b>Description</b>	Activate FastMove function , meaning a new vector is started if controller position difference is larger than the lock-in range.
<b>C++</b>	<pre>int LSX_SetCtrFastMoveOn (int ILSID);</pre>
<b>Parameters</b>	-
<b>Example</b>	<code>pTango-&gt;SetCtrFastMoveOn(1);</code>

## 4.13. Trigger Output

### 4.13.1 GetTrigCount

<b>Description</b>	Retrieve trigger counter value.
<b>C++</b>	int LSX_GetTrigCount (int ILSID, int *pIValue);
<b>Parameters</b>	<i>Value</i> : Number of executed triggers
<b>Example</b>	pTango->GetTrigCount(1, &Value);

### 4.13.2 SetTrigCount

<b>Description</b>	Set trigger counter value.
<b>C++</b>	int LSX_SetTrigCount (int ILSID, int IValue);
<b>Parameters</b>	<i>Value</i> : 0 to 2147483647
<b>Example</b>	pTango->SetTrigCount(1, 0);

### 4.13.3 GetTrigger

<b>Description</b>	Retrieve trigger setting.
<b>C++</b>	int LSX_GetTrigger (int ILSID, BOOL *pbATrigger);
<b>Parameters</b>	<i>ATrigger</i> : TRUE □ trigger is "On" FALSE □ trigger is "Off"
<b>Example</b>	pTango->GetTrigger(1, &ATrigger);

### 4.13.4 SetTrigger

<b>Description</b>	Switch trigger on / off.
<b>C++</b>	int LSX_SetTrigger (int ILSID, BOOL bATrigger);
<b>Parameters</b>	<i>ATrigger</i> = TRUE □ switch trigger on = FALSE □ switch trigger off
<b>Example</b>	pTango->SetTrigger(1, TRUE);



### 4.13.5 GetTriggerPar

<b>Description</b>	Retrieves trigger parameters.
<b>C++</b>	<pre>int LSX_GetTriggerPar (int ILSID, int *plAxis, int *plMode, int *plSignal, double *pdDistance);</pre>
<b>Parameters</b>	<p><b>Axis:</b> Axis 1...4</p> <p><b>Mode:</b> Trigger mode (see command !trigm)</p> <p><b>Signal:</b> Trigger signal (see command !trigs)</p> <p><b>Distance:</b> Trigger distance (see command !trigd)</p>
<b>Example</b>	pTango->GetTriggerPar(1, &Axis, &Mode, &Signal, &Distance);

### 4.13.6 SetTriggerPar

<b>Description</b>	Set trigger parameters.
<b>C++</b>	<pre>int LSX_SetTriggerPar (int ILSID, int lAxis, int lMode, int lSignal, double dDistance);</pre>
<b>Parameters</b>	<p><b>Axis:</b> Axis 1...4</p> <p><b>Mode:</b> Trigger mode (see command !trigm)</p> <p><b>Signal:</b> Trigger signal (see command !trigs)</p> <p><b>Distance:</b> Trigger distance (see command !trigd)</p>
<b>Example</b>	pTango->SetTriggerPar(1, 1, 3, 2, 5.0);

## 4.14. Snapshot Input

### 4.14.1 GetSnapshot

<b>Description</b>	Provides current status of Snapshot.
<b>C++</b>	<code>int LSX_GetSnapshot (int ILSID, BOOL *pbASnapshot);</code>
<b>Parameters</b>	ASnapshot: TRUE □ Snapshot is "On" FALSE □ Snapshot is "Off"
<b>Example</b>	<code>pTango-&gt;GetSnapshot(1, &amp;ASnapshot);</code>

### 4.14.2 SetSnapshot

<b>Description</b>	Switch Snapshot on / off.
<b>C++</b>	<code>int LSX_SetSnapshot (int ILSID, BOOL bASnapshot);</code>
<b>Parameters</b>	ASnapshot: TRUE □ switch Snapshot "On" FALSE □ switch Snapshot "Off"
<b>Example</b>	<code>pTango-&gt;SetSnapshot(1, TRUE);</code>

### 4.14.3 GetSnapshotCount

<b>Description</b>	Snapshot counter. Counts snapshot events (captured positions)
<b>C++</b>	<code>int LSX_GetSnapshotCount (int ILSID, int *plSnsCount);</code>
<b>Parameters</b>	SnsCount: Amount of captured Snapshots.
<b>Example</b>	<code>pTango-&gt;GetSnapshotCount(1, &amp;SnsCount);</code>

### 4.14.4 GetSnapshotFilter

<b>Description</b>	Retrieve input filter times for signal chatter.
<b>C++</b>	<code>int LSX_GetSnapshotFilter (int ILSID, int *plTime);</code>
<b>Parameters</b>	Time: Filter time [ms]
<b>Example</b>	<code>pTango-&gt;GetSnapshotFilter(1, &amp;Time);</code>

### 4.14.5 SetSnapshotFilter

<b>Description</b>	Set input filter when switches chatter.
<b>C++</b>	<code>int LSX_SetSnapshotFilter (int ILSID, int lTime);</code>
<b>Parameters</b>	Time: Filter time, within 0-100 ms
<b>Example</b>	<code>pTango-&gt;SetSnapshotFilter(1, 0);</code> <i>// no filter, fast response</i>

#### 4.14.6 GetSnapshotPar

<b>Description</b>	Retrieve Snapshot parameters.
<b>C++</b>	int LSX_GetSnapshotPar (int ILSID, BOOL *pbHigh, BOOL *pbAutoMode);
<b>Parameters</b>	<p><b>High:</b> TRUE □ snapshot is high active FALSE □ snapshot is low active</p> <p><b>AutoMode:</b> TRUE □ snapshot "Automatic": Position is automatically moved to after first snapshot pulse.</p>
<b>Example</b>	pTango->GetSnapshotPar(1, &High, &AutoMode);

#### 4.14.7 SetSnapshotPar

<b>Description</b>	Set Snapshot parameters.
<b>C++</b>	int LSX_SetSnapshotPar (int ILSID, BOOL bHigh, BOOL bAutoMode);
<b>Parameters</b>	<p><b>High:</b> TRUE □ snapshot is high active FALSE □ snapshot is low active</p> <p><b>AutoMode:</b> TRUE □ snapshot "Automatic": Position is automatically moved to after first snapshot pulse.</p>
<b>Example</b>	pTango->SetSnapshotPar(1, TRUE, FALSE);

#### 4.14.8 GetSnapshotPos

<b>Description</b>	Retrieve position that was captured on the Snapshot event.
<b>C++</b>	int LSX_GetSnapshotPos (int ILSID, double *pdX, double *pdY, double *pdZ, double *pdA);
<b>Parameters</b>	<i>X, Y, Z, A</i> : Position values
<b>Example</b>	pTango->GetSnapshotPos(1, &X, &Y, &Z, &A);

#### 4.14.9 GetSnapshotPosArray

<b>Description</b>	Retrieve Snapshot position from Array.
<b>C++</b>	<pre>int LSX_GetSnapshotPosArray (int ILSID, int IIndex, double *pdX, double *pdY, double *pdZ, double *pdA);</pre>
<b>Parameters</b>	<p><b>Index:</b> Index of snapshot positions (1-200)</p> <p><b>X, Y, Z, A:</b> Position values</p>
<b>Example</b>	<pre>pTango-&gt;GetSnapshotPosArray(1, 2, &amp;X, &amp;Y, &amp;Z, &amp;A); // Read positions captured on the second snapshot event</pre>

## 5. SlideExpress Interface

This chapter describes additional DLL functions usable with SlideExpress.

### 5.1. Eject

<b>Description</b>	Eject magazine(s)
<b>C++</b>	<code>int LSX_Eject (int ILSID, int maga, int keep);</code>
<b>Parameters</b>	maga □ magazine number [1..4] keep □ 0 to empty gripper before eject magazine(s) or 1 to keep slide(s) in gripper
<b>Example</b>	<code>pTango-&gt;Eject(1, 1, 0);</code>

### 5.2. Insert

<b>Description</b>	Magazine(s) are inserted and tested if seated and which slides are present. This function is precondition to use SlideSeated() and MagazinSeated()
<b>C++</b>	<code>int LSX_Insert (int ILSID);</code>
<b>Parameters</b>	-
<b>Example</b>	<code>pTango-&gt;Insert(1);</code>

### 5.3. SlideSeated

<b>Description</b>	Query if slide is present (seated) or not or unknown.
<b>C++</b>	<code>int LSX_SlideSeated (int ILSID, int maga, int slot, int *status);</code>
<b>Parameters</b>	maga □ magazine number [1..4] slot □ slot number [1..50] status □ returns slide status (-1 = unknown, 0 = empty, 1 = seated)
<b>Example</b>	<code>pTango-&gt;SlideSeated (1, 4, 50, &amp;status);</code>

### 5.4. MagazinSeated

<b>Description</b>	Query if magazin is present (seated) or not or unknown.
<b>C++</b>	<code>int LSX_MagazinSeated (int ILSID, int maga, int *status);</code>
<b>Parameters</b>	maga □ magazine number [1..4] status □ returns magazine status (-1 = unknown, 0 = empty, 1 = seated)
<b>Example</b>	<code>pTango-&gt;MagazinSeated (1, 1, &amp;status); //check if magazine 1 is seated</code>

## 5.5. GetGripper

<b>Description</b>	Query gripper status information. Returns status of gripper 1 and 2.
<b>C++</b>	<code>int LSX_GetGripper (int ILSID, int *c1, int *s1, int *c2, int *s2);</code>
<b>Parameters</b>	c1 □ magazine number [-1, 0, 1..4] of slide in gripper 1 s1 □ slot number [-1, 0, 1..50] of slide in gripper 1 c2 □ magazine number [-1, 0, 1..4] of slide in gripper 2 s2 □ slot number [-1, 0, 1..50] of slide in gripper 2
<b>Example</b>	pTango-> GetGripper (1, &c1, &s1, &c2, &s2); //check status of gripper 1 and 2 c1, c2 □ -1 = unknown, 0 = empty or 1 to 4 for magazine number s1, s2 □ -1 = unknown, 0 = empty or 1 to 50 for slot number c1=1,s1=0 indicates priority slide in gripper 1 c2=1,s2=0 indicates priority slide in gripper 2

## 5.6. SetGripper

<b>Description</b>	Set gripper status information. (possibly useful for slide sorting tasks)
<b>C++</b>	<code>int LSX_SetGripper (int ILSID, int c1, int s1, int c2, int s2);</code>
<b>Parameters</b>	c1 □ magazine number [-1, 0, 1..4] of slide in gripper 1 s1 □ slot number [-1, 0, 1..50] of slide in gripper 1 c2 □ magazine number [-1, 0, 1..4] of slide in gripper 2 s2 □ slot number [-1, 0, 1..50] of slide in gripper 2
<b>Example</b>	pTango->SetGripper (1, 0, 0, 0, 0); //set gripper to "empty"

## 5.7. GetSlide

<b>Description</b>	Get slide(s) from addressed position in magazine or priority handler.
<b>C++</b>	<code>int LSX_GetSlide (int ILSID, int maga, int slot, int mode);</code>
<b>Parameters</b>	maga □ magazine number [1..4] slot □ slot number [1..50] or [0] for priority handler mode □ (0 = inspection, 1 = oiler, 2 = bar code reader)
<b>Example</b>	pTango-> GetSlide (1, 1, 1, 0);

## 5.8. PutSlide

<b>Description</b>	Put slide(s) back to addressed position in magazine or priority handler.
<b>C++</b>	<code>int LSX_PutSlide (int ILSID, int maga, int slot);</code>
<b>Parameters</b>	maga □ magazine number [1..4] slot □ slot number [1..50] or [0] for priority handler If both parameters are 0 the DLL transmits !putslide without arguments. In this case Tango uses known gripper information to put slides back.
<b>Example</b>	pTango->PutSlide (1, 4, 50); //put slide to magazine 4 slot 50.

## 5.9. GetPrioHandlerPos

<b>Description</b>	Query actual priority handler position.
<b>C++</b>	<code>int LSX_GetPrioHandlerPos (int ILSID, int *php);</code>
<b>Parameters</b>	php <input type="checkbox"/> return value of actual priority handler position (55 = unknown, 0 = middle, -1 = shift in, 1 = pulled out)
<b>Example</b>	<code>pTango-&gt; GetPrioHandlerPos (1, &amp;php);</code>

## 5.10. SetPrioHandlerPos

<b>Description</b>	Enables user to shift priority handler to required position. Handler is locked at destination or after 30s timeout
<b>C++</b>	<code>int LSX_SetPrioHandlerPos (int ILSID, int php);</code>
<b>Parameters</b>	php <input type="checkbox"/> specify destination 0 = middle, -1 = shift in, 1 = pulled out
<b>Example</b>	<code>pTango-&gt; SetPrioHandlerPos (1, 1); //enable user to pull out priority handler</code>

## 6. Error Codes

### 6.1. Tango Error Messages

0	no error
1	no valid axis name
2	no executable instruction
3	too many characters in command line
4	invalid instruction
5	number is not inside allowed range
6	wrong number of parameters
7	either ! or ? is missing
8	no TVR possible, while axis active
9	no ON or OFF of axis possible, while TVR active
10	function not configured
11	no move instruction possible, while manual joystick enabled
12	limit switch active
14	Error while calibrating (limit switch could not be released)
20	Driver relay broken (safety circuit K3/K4)
21	multiple axis moves are forbidden (e.g. during initialization)
22	automatic or manual move is not allowed (e.g. door open or initialization)
23	Security error X axis
24	Security error Y axis
25	Security error Z axis
26	Security error A axis
27	emergency STOP is active
29	servo amplifier are disabled (switched OFF)
30	safety circuit out of order
50	one argument only expected
51	argument is not a number
52	keyword BEGIN or EOF missing
53	unexpected geo type
58	unexpected sequence
59	alpha and beta must not be equal
70	wrong CPLD data
71	ETS error
72	parameter is write protected (check lock bits)
73	internal error, e.g. eeprom data corruption
74	closed loop switched off due to parameter change
75	could not enable axis correction, or axis correction was disabled
76	io extension card error

### 6.2. DLL Error Messages

0:	no error
4001:	internal error
4002:	internal error
4003:	undefined error
4004:	Unknown interface type (may appear with Connect...)
4005:	Error while initializing interface
4006:	No connection with controller (e.g. if SetPitch is called before Connect)
4007:	Timeout while reading from interface
4008:	Error during command transmission to Tango controller
4009:	Command aborted (with SetAbortFlag)
4010:	Command is not supported by Tango controller
4011:	Manual Joystick mode switched on (may appear with SetJoystickOn/Off)
4012:	No move command possible, because manual joystick enabled
4013:	Closed Loop Controller Timeout (could not settle within target window)





4014:

4015: Limit switch activated in travel direction

4016: Repeated vector start!! (Closed Loop controller)

4017: Error while calibrating (Limit switch not correctly released)

## 7. Document Revision History

No.	Revision	Date	Changes	Remarks
01	A	26. Feb. 2009	Initial version	
02	B	27. Oct. 2011	New MW logo and appearance, Added new error codes, Added HwFactor, HwFactorB, ZwFactor, GetKey, GetKeyLatch, ClearKeyLatch	
03	C	22. Mar. 2013	added: GetAccelFunc, SetAccelFunc GetSwitchType, SetSwitchType GetMotorSteps, SetMotorSteps Chapter 5: SlideExpress Interface	
04	D	08. Nov. 2013	added: Chapter 2.4 LabView Support	