TANGO-DLL

Documentation

In der Murch 15
35579 Wetzlar
Germany
Tel.: +49/6441/9116-0
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

3. Information for Programming Controller via DLL 6

3.1 Initializing the Controller 7
3.2 Own Program Section 9

4. Functions 10

4.1 Table of Contents 10
4.2 DLL Configuration / Interface 16
4.3 Controller Information 20
4.4 Status Requests 21
4.5 Settings 23
4.6 Driving Commands and Positioning Management 35
4.7 Joystick and Handwheel 42
4.8 Control Console with Trackball and Joyspeed Keys 48
4.9 Limit Switches (Hardware and Software) 51
4.10 Digital and Analog Inputs and Outputs 56
4.11 Encoder Settings 59
4.12 Closed Loop Settings 63
4.13 Trigger Output 68
4.14 Snapshot Input 70

5. Error Codes 73

5.1 Tango Error Messages 73
5.2 DLL Error Messages 73

6. Document Revision History 74

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

1.2 System Requirements

The Tango-DLL can be used on all Windows PCs from Windows 98.

1.3 Supported Development Environments

The Tango-DLL has been tested with following development and run time environments:

Microsoft Visual Basic
Microsoft Visual C++
National Instruments LabVIEW
Delphi 2007

It should be compatible to all other programming environments which are able to use DLLs.

(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

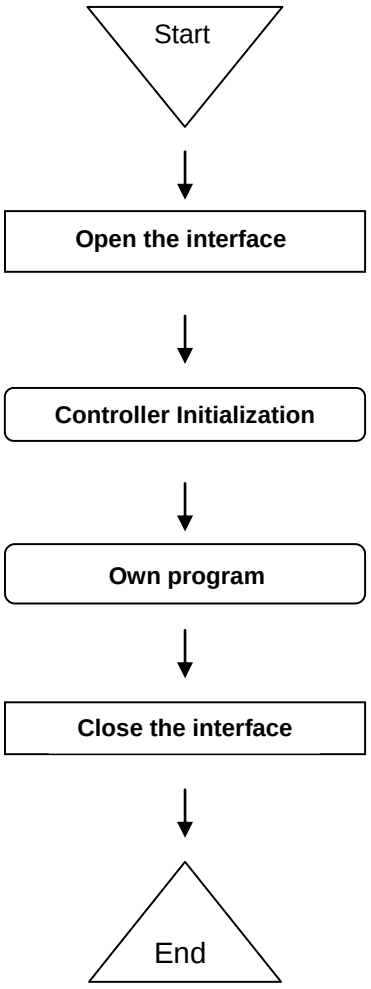
Visual Basic example for controlling a Tango:

```
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
```

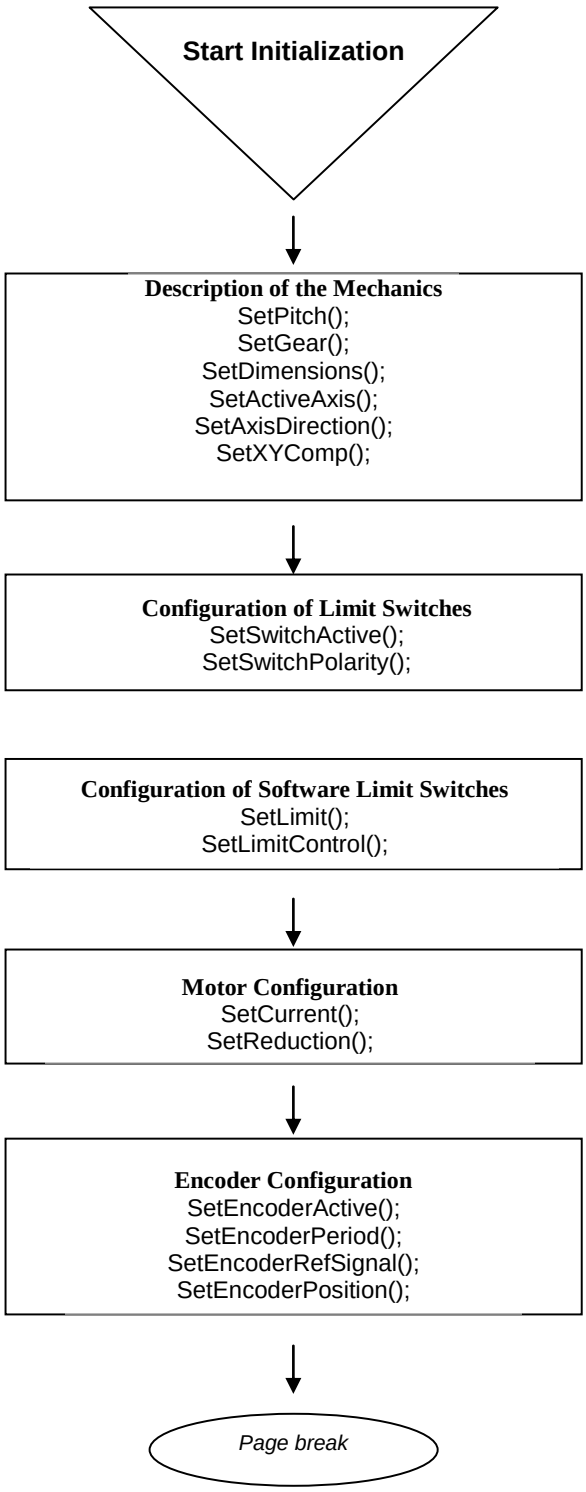
3. Information for creating own Programs when Programming Controllers via DLL

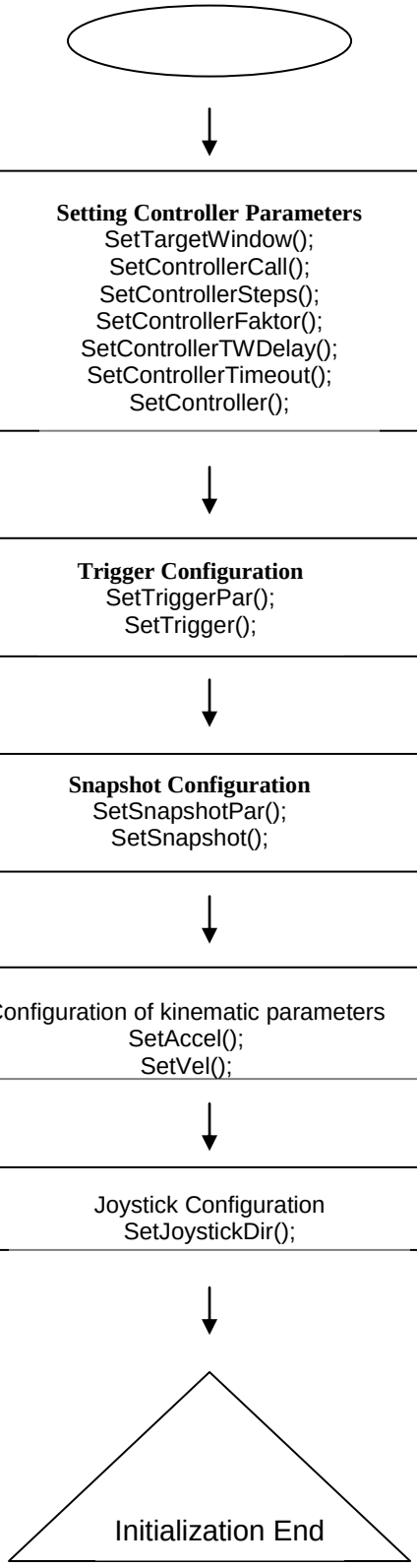
The following illustration shows the program flow, according to which programs for controlling stepper motors should be created. The used functions are listed in the Tango-DLL documentation and are described there more detailed.



3.1 Initializing the Controller

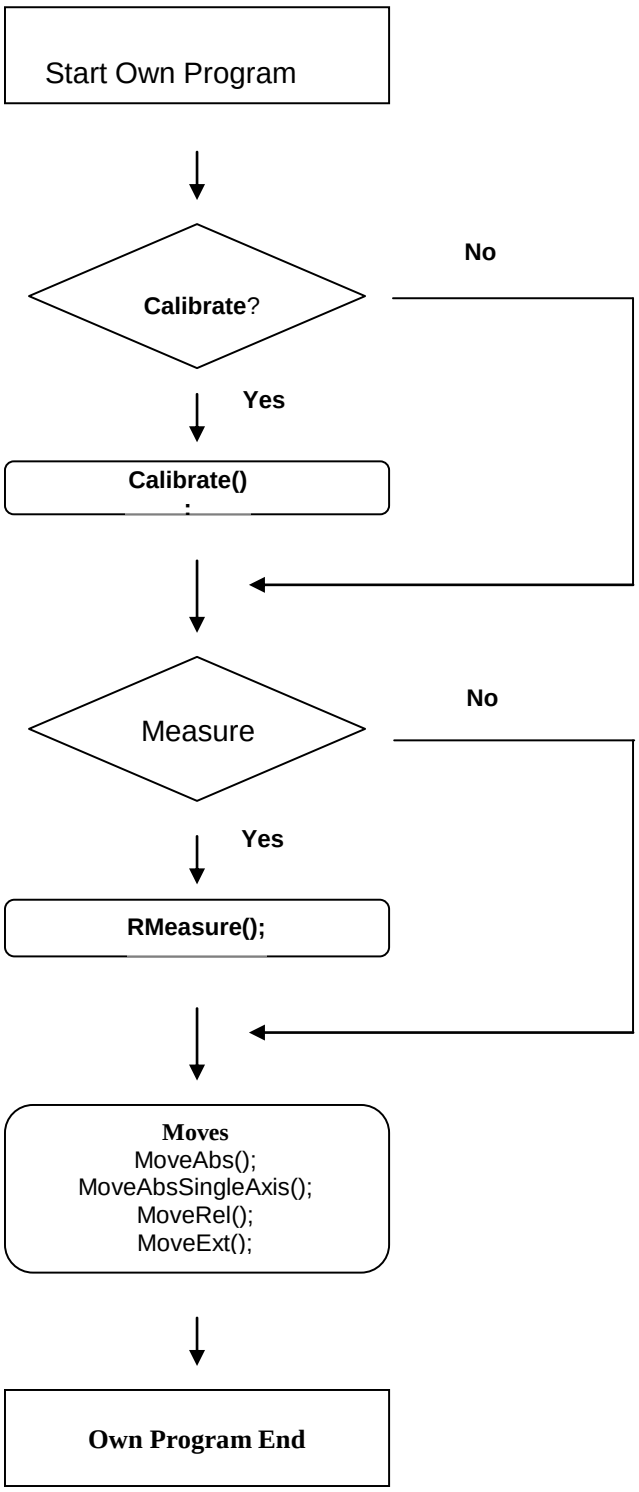
Before starting the own program section, the individual controller settings may be transmitted. If so, please follow the below shown order to guarantee fault free operation of the Tango controller.




Setting Controller Parameters
SetTargetWindow();
SetControllerCall();
SetControllerSteps();
SetControllerFaktor();
SetControllerTWDelay();
SetControllerTimeout();
SetController();
Trigger Configuration
SetTriggerPar();
SetTrigger();
Snapshot Configuration
SetSnapshotPar();
SetSnapshot();
Configuration of kinematic parameters
SetAccel();
SetVel();
Joystick Configuration
SetJoystickDir();
Initialization End

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 Table of Contents

DLL Configuration / Interface:

Command	Brief Description	Page
ConnectSimple	Connect to Tango	16
CreateLSID	Creates a Tango-ID number	16
Disconnect	Disconnects Tango Controller from DLL	16
EnableCommandRetry	This command enables switching on / off of repeated command sending in case of communication errors	17
FlushBuffer	Clears the receive buffer from possibly remaining data fragments	17
FreeLSID	Releases the previously created Tango ID-Number	17
SendString	Sends strings to Tango (enables using all commands as ASCII text)	18
SendStringPosCmd	Send an ASCII move command and wait for completion reply	18
SetAbortFlag	Set internal DLL flag to abort a (hanging) communication	19
SetShowProt	Switches communication monitoring on/off	19

Controller information:

Command	Brief Description	Page
GetSerialNr	Read out the Controller serial number	20
GetVersionStr	Provides current firmware version number	20
GetVersionStrDet	Reads detailed firmware version information	20
GetVersionStrInfo	Retrieves additional information to current version number	20

Status Requests:

Command	Brief Description	Page
GetError	Provides current error number	21
GetPos	Retrieves current position of all axes	21
GetPosEx	Retrieves values of current encoder- or motor-positions of all axes	21
GetPosSingleAxis	Retrieves current position of one axis	22
GetStatus	Provides current Controller status	22
GetStatusAxis	Provides current status of one axis	22
GetStatusLimit	Provides current status of software limits of all axes	23
SetAutoStatus	Switches Auto-Status reply on/off	23

Controller Settings:

Command	Brief Description	Page
GetAccel	Retrieves acceleration	24
GetActiveAxes	Provides enabled axes	24
GetAxisDirection	Retrieves rotation direction of axes	25
GetCalibBackSpeed	Provides speed in which is driven out of the limit switches	25
GetCalibOffset	Retrieves calibration position offset which is kept to the limit switch	26
GetCalibrateDir	Provides direction inversion when calibrating	26
GetCurrentDelay	Shows time delay for motor current reduction	27
GetDimensions	Retrieves measuring units of the axes	27
GetGear	Retrieves gear factor	28
GetMotorCurrent	Retrieves motor current	29
GetPitch	Provides spindle pitch	29
GetPowerAmplifier	Provides whether amplifiers are switched on or off	30
GetReduction	Retrieves motor current reduction factor	30
GetRMOffset	Retrieves position offset which is kept to the RM limit switch	31
GetSpeedPoti	Shows whether potentiometer function is switched on or off	31
GetStopAccel	Provides brake acceleration settings for stop input	32
GetStopPolarity	Retrieves polarity for stop input	32
GetVel	Retrieves velocity of all axes	33
GetVelFac	Retrieves velocity factor	33
LStepSave	Save current configuration in Tango (EEPROM)	34
SetAccel	Set acceleration	24
SetAccelSingleAxis	Set acceleration of single axes	34
SetActiveAxes	Set enable state for axes	25
SetAxisDirection	Set rotating direction of axes	25
SetCalibBackSpeed	Set speed in which is driven out of the limit switches	26
SetCalibOffset	Set position offset to CAL endswitch	26
SetCalibrateDir	Set direction inversion when calibrating	27
SetCurrentDelay	Set time delay for motor current reduction	27
SetDimensions	Set measuring units of axes	28
SetGear	Set gear factor	28
SetMotorCurrent	Set motor current	29
SetPitch	Set spindle pitch	29
SetPowerAmplifier	Switch amplifiers on or off	30
SetReduction	Set motor current reduction factor	40
SetRMOffset	Set position offset which is kept to the RM limit switch	31
SetSpeedPoti	Switches the speed potentiometer functionality on or off	31
SetStopAccel	Set brake acceleration for stop input	32
SetStopPolarity	Set signal polarity for stop input	32
SetVel	Set velocity of all axes	33
SetVelFac	Set velocity factor	33
SetVelSingleAxis	Set velocity for a single axis	34
SoftwareReset	Reset and reboot the controller	34

Move Commands and Position Management:

Command	Brief Description	Page
Calibrate	Calibrate enabled axes to the CAL limit switches	35
CalibrateEX	Calibrates single axes	35
ClearPos	Sets position values to zero	35
GetDelay	Provides delay of vector start	36
GetDistance	Provides distance started with MoveRelShort	36
MoveAbs	Moves to absolute position of all axes	37
MoveAbsSingleAxis	Moves to absolute position of single axis	37
MoveEx	Extended move/move relative command with axis bit mask	38
MoveRel	Move by relative vector for all axes	38
MoveRelShort	Relative positioning (short command)	39
MoveRelSingleAxis	Move single axis relatively	39
RMeasure	Measure maximum travel range of all axes	39
RMeasureEx	Measure max. travel range of axes selected by the axis bit mask	40
SetDelay	Causes delay of vector start	36
SetDistance	Sets distance for MoveRelShort command	36
SetPos	Set current position to the desired value	40
StopAxes	Stops all moving axes	40
WaitForAxisStop	Function returns as soon as all axes chosen in bit mask have reached their end position	41

Joystick and Handwheel:

Command	Brief Description	Page
GetDigJoySpeed	Retrieves current digital joystick speed	42
GetHandWheel	Retrieves handwheel status	42
GetJoystick	Retrieves analog joystick status	42
GetJoystickDir	Retrieves revolve direction of motor for joystick	43
GetJoystickWindow	Retrieves joystick window	44
GetHwFactor	Retrieves handwheel factor	45
GetHwFactorB	Retrieves second handwheel factor	46
GetZwTravel	Retrieves z-wheel travel distances	46
GetKey	Retrieves key state	47
GetKeyLatch	Retrieves and clears latched key states	47
SetDigJoySpeed	Start a move at constant speed (commanded digital joystick)	42
SetHandWheelOff	Switches handwheel off	44
SetHandWheelOn	Switches handwheel on	45
SetJoystickDir	Sets analog joystick direction	44
SetJoystickOff	Switches analog joystick off	45
SetJoystickOn	Switches analog joystick on	45
SetJoystickWindow	Set analog joystick idle window	44
SetHwFactor	Set handwheel factor	46
SetHwFactorB	Set second handwheel factor	46
SetZwTravel	Set z-wheel travel distances	47
ClearKeyLatch	Clears latched key states	47

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

Command	Brief Description	Page
GetBPZ	Retrieves status of control console	48
GetBPZJoyspeed	Retrieves control console joystick speed	48
GetBPZTrackballBackLash	Retrieves control console trackball backlash	49
GetBPZTrackballFactor	Retrieves control console trackball factor	49
SetBPZ	Switches control console on / off	48
SetBPZJoyspeed	Set control console joystick speed	49
SetBPZTrackballBackLash	Set control console trackball backlash	49
SetBPZTrackballFactor	Set control console trackball factor	50

Limit Switches (Hardware and Software):

Command	Brief Description	Page
GetAutoLimitAfterCalibRM	Provides, whether internal software limits are set when calibrating or measuring stage travel range	51
GetLimit	Provides travel range limits of single axes	51
GetLimitControl	Retrieves whether area control is switched on or off	52
GetSwitchActive	Provides, whether limit switches are active	53
GetSwitches	Retrieves status of all limit switches	54
GetSwitchPolarity	Retrieves polarity of limit switches	54
SetAutoLimitAfterCalibRM	Prevents setting internal software limits by calibration or range measure	51
SetLimit	Sets travel range limits of single axes	52
SetLimitControl	Switches area control on / off	52
SetSwitchActive	Enable/disable limit switches	53
SetSwitchPolarity	Sets polarity of limit switches	55

Digital and Analog Inputs and Outputs:

Command	Brief Description	Page
GetAnalogInput	Retrieves current level of analogue input signals	56
GetDigitalInputs	Retrieve all digital input pin levels	56
GetDigitalInputsE	Retrieve additional digital inputs 16-31	56
SetAnalogOutput	Set analogue output voltage	56
SetDigIO_Distance	Activate an output, depending on set distance before or after reaching determined position	57
SetDigIO_EmergencyStop	Assign Emergency-Stop pin	58
SetDigIO_Off	Switch off digital I/O functionality	57
SetDigIO_Polarity	Set polarity	58
SetDigitalOutput	Set digital output	58
SetDigitalOutputs	Set digital outputs 0-15	58
SetDigitalOutputsE	Set additional digital outputs 16-31	58

Encoder Settings:

Command	Brief Description	Page
ClearEncoder	Set encoder position to zero	58
GetEncoder	Retrieves all encoder positions	58
GetEncoderActive	Retrieves which encoder is activated after calibration (*encmask*)	58
GetEncoderMask	Retrieve status of encoders (“*enc” command!*)	59
GetEncoderPeriod	Retrieves length of encoder signal period	59
GetEncoderPosition	Provides, whether encoder- or motor- position is displayed	60
GetEncoderRefSignal	Provides if reference signal from encoder shall be evaluated when calibrating	60
SetEncoderActive	Select encoder to be activated after calibration	58
SetEncoderMask	Activates / deactivates encoders	59
SetEncoderPeriod	Set length of encoder period	60
SetEncoderPosition	Switches encoder value display on / off	60
SetEncoderRefSignal	Evaluate encoder reference signal when calibrating.	61

Closed Loop Settings:

Command	Brief Description	Page
ClearCtrFastMoveCounter	Resets number of executed FastMove functions to 0	62
GetController	Retrieve controller mode	62
GetControllerCall	Provides controller call interval	63
GetControllerFactor	Retrieve setting of controller factor	63
GetControllerSteps	Retrieve controller steps	64
GetControllerTimeout	Retrieves setting of controller monitoring timeout	64
GetControllerTWDelay	Retrieve controller delay for target window	65
GetCtrFastMove	Retrieves whether FastMove function is switched on or off	65
GetCtrFastMoveCounter	Retrieves number of executed FastMove functions	65
GetTargetWindow	Retrieves target windows of all axes	66
SetController	Set controller mode	62
SetControllerCall	Set controller call time	63
SetControllerFactor	Set controller factor	63
SetControllerSteps	Set controller steps	64
SetControllerTimeout	Set controller monitoring timeout	64
SetControllerTWDelay	Set controller delay	65
SetCtrFastMoveOff	Switch off FastMove function	66
SetCtrFastMoveOn	Switch on FastMove function	66
SetTargetWindow	Set controller target windows	66

Trigger Output:

Command	Brief Description	Page
GetTrigCount	Retrieve trigger counter value	67
GetTrigger	Retrieve trigger setting	67
GetTriggerPar	Retrieve trigger parameters	68
SetTrigCount	Set trigger counter value	67
SetTrigger	Switch trigger on / off	67
SetTriggerPar	Set trigger parameters	68

Snapshot-Input:

Command	Brief Description	Page
GetSnapshot	Provides current status of Snapshot	69
GetSnapshotCount	Read Snapshot counter	69
GetSnapshotFilter	Retrieve input filter	69
GetSnapshotPar	Retrieve Snapshot parameters	70
GetSnapshotPos	Retrieve Snapshot position	70
GetSnapshotPosArray	Retrieve Snapshot position from array	71
SetSnapshot	Switch Snapshot on / off	69
SetSnapshotFilter	Set input filter	70
SetSnapshotPar	Set Snapshot parameters	70

4.2 DLL Configuration / Interface

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

LSX_CreateLSID	
Description:	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.
C++:	int LSX_CreateLSID(int *plLSID);
Parameters:	***LSID***: Contains a new Tango ID-Number after calling CreateLSID, which can then be used for commands such as connect, move and others
Example:	int Tango1, Tango2; pTango->CreateLSID(&Tango1); *// create ID for first Tango* pTango->CreateLSID(&Tango2); *// create ID for second Tango*

LSX_Disconnect	
Description:	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.
C++:	int LSX_Disconnect(int lLSID);
Parameters:	-
Example:	pTango->Disconnect(1);

LSX_EnableCommandRetry	
Description:	This function enables/disables repeated sending of commands in case of errors (Default enabled).
C++:	int LSX_EnableCommandRetry (int lLSID, BOOL bAValue);
Parameters:	***AValue***: TRUE → in case of errors Tango DLL repeats sending certain command (especially in case of WaitForAxisStop) FALSE → disable repeated sending
Example:	pTango->EnableCommandRetry(1, FALSE);

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

LSX_FreeLSID	
Description:	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.
C++:	int LSX_FreeLSID(int lLSID);
Parameters:	***LSID***: The Tango ID-Number, which is to be set free. Do not use the ID after after a FreeLSID.

Example:	int Tango1;
	
	pTango->CreateLSID(&Tango1);
	pTango->ConnectSimple(Tango1, …);
	
	pTango->Disconnect(Tango1);
	pTango-> FreeLSID(Tango1);

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

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

LSX_SetAbortFlag	
Description:	Set flag so that communication with Tango is cut off. 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.
C++:	int LSX_SetAbortFlag (int lLSID);
Parameters:	-

Example:	pTango->SetAbortFlag(1);
	pTango->StopAxes(1);
	// closes communication with Tango and sends stop command for all axes

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

4.3 Controller Information

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

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

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

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

4.4 Status Requests:

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

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

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

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

LSX_GetStatus	
Description:	Provides current status of the controller.
C++:	int LSX_GetStatus (int lLSID, char *pcStat, int lMaxLen);
Parameters:	***Stat***: Pointer to a buffer, in which the status string will be returned ***MaxLen***: Max. amount of characters allowed to be copied into buffer
Example:	pTango->GetStatus(1, &Stat, 16);

LSX_GetStatusAxis	
Description:	Provides current status of the axes.
C++:	int LSX_GetStatusAxis (int lLSID, char *pcStatusAxisStr, int lMaxLen);
Parameters:	***StatusAxisStr***: Pointer to a buffer in which status string will be returned ***MaxLen***: 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
Example:	pTango->GetStatusAxis(1, &StatusAxisStr, 16);

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

LSX_SetAutoStatus	
Description:	Switches Auto-Status on/off. 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.
C++:	int LSX_SetAutoStatus (int lLSID, int lValue);
Parameters:	***Value***: AutoStatus mode: 0 → Controller sends no status 1 → Controller automatically sends ´´Position reached´´ messages 2 → Controller automatically sends ´´Position reached´´ and status messages 3 → There is only one carriage return sent for ´´Position reached´´
Example:	pTango->SetAutoStatus(1, 1);

4.5 Settings

LSX_GetAccel	
Description:	Retrieves acceleration.
C++:	int LSX_GetAccel (int lLSID, double *pdX, double *pdY, double *pdZ, double *pdA);
Parameters:	***X, Y, Z, A***: Acceleration values [m/s²]
Example:	pTango->GetAccel(1, &X, &Y, &Z, &A);

LSX_SetAccel	
Description:	Set acceleration.
C++:	int LSX_SetAccel (int lLSID, double dX, double dY, double dZ, double dA);
Parameters:	**X, Y, Z, A**: 0.01 - 20.00 [m/s²]
Example:	pTango->SetAccel(1, 1.0, 1.5, 0, 0);

LSX_GetActiveAxes	
Description:	Provides the axis enable states.
C++:	int LSX_GetActiveAxes (int lLSID, int *plFlags);
Parameters:	***Flags***: 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
Example:	pTango->GetActiveAxes(1, &Flags);

LSX_SetActiveAxes	
Description:	Enable or disable axes.
C++:	int LSX_SetActiveAxes (int lLSID, int lFlags);
Parameters:	***Flags***: Bit mask, bits 0 to 4 represent axes X to A Bit 0 = 1 → X-Axis disabled Bit 2 = 0 → Z-Axis enabled
Example:	pTango->SetActiveAxes(1, 3); *// X- and Y-Axis cleared (Bits 0 and 1 set),* *Z-Axis not cleared (Bit 2 = 0)*

LSX_GetAxisDirection	
Description:	Retrieves axis directions.
C++:	int LSX_GetAxisDirection (int lLSID, int *plXD, int *plYD, int *plZD, int *plAD);
Parameters:	***XD, YD, ZD, AD***: 4 32-Bit Integers 0 → normal rotating direction 1 → reversed rotating direction
Example:	pTango->GetAxisDirection(1, &XD, &YD,&ZD,&AD);

LSX_SetAxisDirection	
Description:	Set axis directions.
C++:	int LSX_SetAxisDirection (int lLSID, int lXD, int lYD, int lZD, int lAD);
Parameters:	***XD, YD, ZD, AD***: 4 32-Bit Integers 0 → normal motor turning direction 1 → reverse reversed motor turning direction
Example:	pTango->SetAxisDirection(1, 1, 0, 0, 0); *// reverse direction of X-Axis*

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

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

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

LSX_SetCalibOffset	
Description:	Sets zero position offset of axes. The axis zero position is moved from the hardware cal limit switch by this amount.
C++:	int LSX_SetCalibOffset (int lLSID, double dX, double dY, double dZ, double dA);
Parameters:	***X, Y, Z, A***: typically 0-5 [mm]
Example:	pTango->SetCalibOffset(1, 1, 1, 1, 1); *// 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)*

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

LSX_SetCalibrateDir	
Description:	Set calibrating direction.
C++:	int LSX_SetCalibrateDir (int lLSID, int lXD, int lYD, int lZD, int lAD);
Parameters:	***XD, YD, ZD, AD***: 32-Bit Integer 0 → normal calibration direction 1 → reverse calibration direction
Example:	pTango->(1, 1, 1, 0, 0);

LSX_GetCurrentDelay	
Description:	Provides time delay for motorcurrent reduction.
C++:	int LSX_GetCurrentDelay (int lLSID, int *plX, int *plY, int *plZ, int *plA);
Parameters:	***X, Y, Z, A***: Time delay [ms]
Example:	pTango->GetCurrentDelay(1, &X, &Y,&Z,&A);

LSX_SetCurrentDelay	
Description:	Sets the time delay, after which the motor current is reduced.
C++:	int LSX_SetCurrentDelay (int lLSID, int lX, int lY, int lZ, int lA);
Parameters:	***X, Y, Z, A***: 010000 [ms] (A delay of 0 disables the current reduction)
Example:	pTango->SetCurrentDelay(1, 100, 300, 1000, 0);

LSX_GetDimensions	
Description:	Provides the applied measuring units of axes
C++:	int LSX_GetDimensions (int lLSID, int *plXD, int *plYD, int *plZD, int *plAD);
Parameters:	***XD, YD, ZD, AD***: 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)
Example:	pTango->GetDimensions(1, &XD, &YD,&ZD,&AD);

LSX_SetDimensions	
Description:	Set measuring units of axes.
C++:	int LSX_SetDimensions (int lLSID, int lXD, int lYD, int lZD, int lAD);
Parameters:	***XD, YD, ZD, AD***: 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)
Example:	pTango->SetDimensions(1, 3, 2, 2, 1); *// X-Axis in degree, Y- and Z-Axis in mm and A-Axis in µm*

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

LSX_SetGear	
Description:	Set gear ratio.
C++:	int LSX_SetGear (int lLSID, double dX, double dY, double dZ, double dA);
Parameters:	***X, Y, Z, A:*** 0.01 - 1000
Example:	pTango->SetGear(1, 4.0, 2.0, 1.0, 1.0); *// programs gear ratios ¼ for Z, ½ for Y and 1/1 for Z and A*

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

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

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

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

LSX_GetPowerAmplifier	
Description:	Provides, whether amplifiers are switched on or off.
C++:	int LSX_GetPowerAmplifier (int lLSID, BOOL *pbAmplifier);
Parameters:	***Amplifier***: TRUE → Amplifiers are switched on FALSE → Amplifiers are switched off
Example:	pTango->GetPowerAmplifier(1, &Amplifier);

LSX_SetPowerAmplifier	
Description:	Switch amplifier on / off.
C++:	int LSX_SetPowerAmplifier (int lLSID, BOOL bAmplifier);
Parameters:	***Amplifier***: TRUE → Switch amplifiers on FALSE → Switch amplifiers off
Example:	pTango->SetPowerAmplifier(1, TRUE); *// switches amplifiers on*

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

LSX_SetReduction	
Description:	Set reduction factor of motor current.
C++:	int LSX_SetReduction (int lLSID, double dX, double dY, double dZ, double dA);
Parameters:	***X, Y, Z, A:*** 0 - 1.0
Example:	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*

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

LSX_SetRMOffset	
Description:	Sets RM position offset of axes. The axis stops this amount before the hardware RM endswitch.
C++:	int LSX_SetRMOffset (int lLSID, double dX, double dY, double dZ, double dA);
Parameters:	***X, Y, Z, A***: typically 0-5 [mm]
Example:	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*

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

LSX_SetSpeedPoti	
Description:	Switches Speed Potentiometer functionality on or off.
C++:	int LSX_SetSpeedPoti (int lLSID, BOOL bSpeedPoti);
Parameters:	***SpeedPoti*** = FALSE → pre-set speed (vel) is used as movement speed = TRUE → pre-set speed (vel) can be reduced depending on the speed-potentiometer deflection
Example:	pTango->SetSpeedPoti(1, TRUE); *// potentiometer is switched on*

LSX_GetStopAccel	
Description:	Provides deceleration for error conditions.
C++:	int LSX_GetStopAccel (int lLSID, double *pdXD, double *pdYD, double *pdZD, double *pdAD);
Parameters:	***XD, YD, ZD, AD***: Deceleration values [m/s²]
Example:	pTango->GetStopAccel(1, &XD, &YD, &ZD, &AD);

LSX_SetStopAccel	
Description:	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.
C++:	int LSX_SetStopAccel (int lLSID, double dX, double dY, double dZ, double dA);
Parameters:	***X, Y, Z, A***: Brake acceleration, within parameters 0.01 to 20 [m/s²]
Example:	pTango->SetStopAccel(1, 1.5, 1.5, 1.5, 1.5);

LSX_GetStopPolarity	
Description:	Retrieves active polarity of the stop input signal.
C++:	int LSX_GetStopPolarity (int lLSID, BOOL *pbHighActiv);
Parameters:	***HighActiv***: TRUE → stop input is high active FALSE → stop input is low active
Example:	pTango->GetStopPolarity(1, &HighActiv);

LSX_SetStopPolarity	
Description:	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.
C++:	int LSX_SetStopPolarity (int lLSID, BOOL bHighActiv);
Parameters:	***HighActiv***: TRUE→ stop input high active FALSE → stop input low active
Example:	pTango->SetStopPolarity(1, FALSE); *// stop input is low active (e.g. normally open switch to ground)*

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

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

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

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

LSX_LStepSave	
Description:	Save current configuration in Tango (EEPROM).
C++:	int LSX_LStepSave (int lLSID);
Parameters:	-
Example:	pTango->LStepSave(1);

LSX_SetAccelSingleAxis	
Description:	Set acceleration of a single axis.
C++:	int LSX_SetAccelSingleAxis (int lLSID, int lAxis, double dAccel);
Parameters:	***Axis***: X, Y, Z, A numbered from 1 to 4 ***Accel***: Acceleration 0.01 - 20.00 [m/s²]
Example:	pTango->SetAccelSingleAxis(1, 3, 1,0); *// sets acceleration of Z-Axis to 1.0 m/s²*

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

LSX_SoftwareReset	
Description:	Software is reset to starting condition (reboot).
C++:	int LSX_SoftwareReset (int lLSID);
Parameters:	-
Example:	pTango->SoftwareReset(1);

4.6 Move Commands and Positioning Management

LSX_Calibrate	
Description:	All enabled axes will be calibrated. 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.
C++:	int LSX_Calibrate (int lLSID);
Parameters:	-
Example:	pTango->Calibrate(1);

LSX_CalibrateEx	
Description:	Calibrates single axes. Only calibrates axes with corresponding Bit set in transferred Integer value.
C++:	int LSX_CalibrateEx (int lLSID, int lFlags);
Parameters:	***Flags***: Bit mask Bit 0=X, Bit 1=Y, Bit 2=Z, Bit 3=A If Bit 2 = 1 → calibrate Z-Axis If Bit 2 = 0 → do not calibrate Z-Axis
Example:	pTango->CalibrateEx(1, 6); *// only calibrate Y- and Z-Axis (Bit 1 and 2 set)*

LSX_ClearPos	
Description:	Sets current position and internal position counter to 0. This function is needed for endless axes, as controller can only process ±1,000 motor revolutions within its parameters. This instruction will be ignored for axes with encoders.
C++:	int LSX_ClearPos (int lLSID, int lFlags);
Parameters:	***Flags***: Bit mask Bit 0=X, Bit 1=Y, Bit 2=Z, Bit 3=A Bit 0 = 1 → position of X-Axis is set to zero. Bit 1 = 0 → function is not executed fofor Y-Axis.
Example:	pTango->ClearPos(1, 5); *// positions of X- and Z-Axis are set to zero (Bit 0 and 2 set)*

LSX_GetDelay	
Description:	Retrieves time delay (wait time) until a commanded move is executed.
C++:	int LSX_GetDelay (int lLSID, int *plDelay);
Parameters:	***Delay***: Delay [ms]
Example:	pTango->GetDelay(1, &Delay);

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

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

LSX_SetDistance	
Description:	Set distance. Sets distance parameters for command LSX_MoveRelShort. This enables very fast equal distance relative positioning without the need of communication overhead.
C++:	int LSX_SetDistance (int lLSID, double dX, double dY, double dZ, double dA);
Parameters:	***X, Y, Z, A***: Min-/max- travel range, values depend on measuring unit.
Example:	pTango->SetDistance(1, 1, 2, 0, 0); *// 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*

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

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

LSX_MoveEx	
Description:	Extended move command. 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.
C++:	int LSX_MoveEx (int lLSID, double dX, double dY, double dZ, double dA, BOOL bRelative, BOOL bWait, int lAxisCount);
Parameters:	***X, Y, Z, A***: Position vectors ***Relative***: 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 ***Wait***: 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. ***AxisCount***: 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 ...
Example:	pTango->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*

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

LSX_MoveRelShort	
Description:	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.
C++:	int LSX_MoveRelShort (int lLSID);
Parameters:	-

Example:	pTango->SetDistance(1, 1.0, 1.0, 0, 0);
	for (i = 0; i < 10; i++) pTango->MoveRelShort(1);
	// position X- and Y-Axis 10 times relatively by 1mm

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

LSX_RMeasure	
Description:	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.
C++:	int LSX_RMeasure (int lLSID);
Parameters:	-
Example:	pTango->RMeasure(1);

LSX_RMeasureEx	
Description:	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.
C++:	int LSX_RMeasureEx (int lLSID, int lFlags);
Parameters:	***Flags***: Bit mask Bit 2 = 1 → calibrate Z-Axis Bit 2 = 0 → Do not calibrate Z-Axis ...
Example:	pTango->RMeasureEx(1, 2); *// only measure maximum position of Y-Axis*

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

LSX_StopAxes	
Description:	Abort. Stops all moving axes.
C++:	int LSX_StopAxes (int lLSID);
Parameters:	-
Example:	pTango->StopAxes(1);

LSX_WaitForAxisStop	
Description:	Function returns as soon as the axes selected by the bit mask “lAFlags” have reached their target positions or the timeout is exceeded. LSX_WaitForAxisStop uses '?statusaxis', to poll axis status.
C++:	int LSX_WaitForAxisStop (int lLSID, int lAFlags, int lATimeoutValue, BOOL *pbATimeout);
Parameters:	***AFlags***: Bit mask Bit 0: X-Axis Bit 1: Y-Axis Bit 2: Z-Axis Bit 3: A-Axis ***AtimeoutValue***: Timeout in milliseconds WaitForAxisStop returns latest after this period of time pbATimeout is set to “TRUE”, if axes are still in motion. Setting lATimeoutValue = 0 disables the Timeout (wait infinite) ***pbATimeout*** Flag: Shows whether a Timeout has occurred

Example:	pTango->WaitForAxisStop(1, 3, 0, flag);
	// wait until X- and Y-Axes have stopped, no Timeout
	
	pTango->WaitForAxisStop(1, 7, 10000, flag);
	// wait until X-, Y- and Z-Axis has stopped, 10 sec. Timeout

4.7 Joystick and Handwheel

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

LSX_SetDigJoySpeed	
Description:	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.
C++:	int LSX_SetDigJoySpeed (int lLSID, double dX, double dY, double dZ, double dA);
Parameters:	***X, Y, Z, A***: Speed [r/sec], within parameter range: + max. speed
Example:	pTango->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´´*

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

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

LSX_GetJoystickDir	
Description:	Retrieves axis direction for the analog Joystick and other HDI input devices.
C++:	int LSX_GetJoystickDir (int lLSID, int *plXD, int *plYD, int *plZD, int *plAD);
Parameters:	***XD, YD, ZD, AD:*** 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
Example:	pTango->GetJoystickDir(1, &XD, &YD, &ZD, &AD);

LSX_SetJoystickDir	
Description:	Sets axis direction for Joystick and other HDI input devices.
C++:	int LSX_SetJoystickDir (int lLSID, int lXD, int lYD, int lZD, int lAD);
Parameters:	***XD, YD, ZD, AD:*** 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
Example:	pTango->SetJoystickDir(1, 1, 1, -1, 0); *// X- and Y-Axis positive direction, Z-Axis negative direction, A-Axis blocked*

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

LSX_SetJoystickWindow	
Description:	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.
C++:	int LSX_SetJoystickWindow (int lLSID, int lAValue);
Parameters:	***AValue***: Analogoue signal range (as digits) in which axes do not move. 0 ... 100
Example:	pTango->SetJoystickWindow(1, 30);

LSX_SetHandWheelOff	
Description:	Switch handwheel off.
C++:	int LSX_SetHandWheelOff (int lLSID);
Parameters:	-
Example:	pTango->SetHandWheelOff(1);

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

LSX_SetJoystickOff	
Description:	Switch analog Joystick off.
C++:	int LSX_SetJoystickOff (int lLSID);
Parameters:	-
Example:	pTango->SetJoystickOff(1);

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

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

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

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

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

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

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

LSX_GetKey	
Description:	Get current key states
C++:	int LSX_GetKey (int lLSID, BOOL *pbKey1, BOOL *pbKey2, BOOL *pbKey3, BOOL *pbKey4);
Parameters:	Pointers to double
Example:	pTango-> GetKey(&bKey[0], &bKey[1], &bKey[2], &bKey[3]);;

LSX_GetKeyLatch	
Description:	Get and clear latched key states
C++:	int LSX_GetKeyLatch (int lLSID, BOOL *pbKey1, BOOL *pbKey2, BOOL *pbKey3, BOOL *pbKey4);
Parameters:	Pointers to double
Example:	pTango-> GetKeyLatch(&bKey[0], &bKey[1], &bKey[2], &bKey[3]);

LSX_ClearKeyLatch	
Description:	Clear latched key state(s)
C++:	int LSX_ClearKeyLatch (int lLSID, int lKey);
Parameters:	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
Example:	pTango-> ClearKeyLatch(0); // Clear all

4.8 Control Console with Trackball and Joyspeed Keys

LSX_GetBPZ	
Description:	Retrieves status of a custom-built control console with trackball.
C++:	int LSX_GetBPZ (int lLSID, int *plAValue);
Parameters:	***AValue***: 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.
Example:	pTango->GetBPZ(1, &AValue);

LSX_SetBPZ	
Description:	Switches custom-built control console on / off.
C++:	int LSX_SetBPZ (int lLSID, int lAValue);
Parameters:	***AValue***: 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.
Example:	pTango->SetBPZ(1, 1);

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

LSX_SetBPZJoyspeed	
Description:	Set custom-built control console joystick speed.
C++:	int LSX_SetBPZJoyspeed (int lLSID, int lAPar, double dAValue);
Parameters:	***APar***: 1, 2 or 3 (console keys for speed selection: slow, medium, fast) ***AValue***: ±max. speed [r/sec]
Example:	pTango->SetBPZJoyspeed(1, 1, 25); *// Set key 1 parameter (slow) to speed 25*

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

LSX_SetBPZTrackballBackLash	
Description:	Set custom-built control console trackball backlash.
C++:	int LSX_SetBPZTrackballBackLash (int lLSID, double dX, double dY, double dZ, double dA);
Parameters:	***X, Y, Z, A***: 0.001 to 0.15 mm
Example:	pTango->SetBPZTrackballBackLash(1, 0.01, 0.01, 0.01, 0.01); // Set backlash for all axes to 10µm

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

LSX_SetBPZTrackballFactor	
Description:	Set custom-built control console trackball factor.
C++:	int LSX_SetBPZTrackballFactor (int lLSID, double dAValue);
Parameters:	***AValue***: 0.01 ... 100 AValue = 1 → Trackball factor = 1, i.e. one trackball impulse results in one motor increment
Example:	pTango->SetBPZTrackballFactor(1, 1,0);

4.9 Limit Switches (Hardware and Software)

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

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

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

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

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

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

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

LSX_SetSwitchActive	
Description:	Switches limit switches on / off.
C++:	int LSX_SetSwitchActive (int lLSID, int lXA, int lYA, int lZA, int lAA);
Parameters:	A bit mask is supplied for each axis: Bit 0 → zero limit switch (cal, “E0”) Bit 1 → reference limit switch (unused) Bit 2 → end limit switch (rm, “EE”) The limit switch is enabled if the corresponding bit is set.
Example:	pTango->SetSwitchActive(1, 7, 1, 5, 0); *// X-Axis: All limit switches enabled, Y-Axis: Only Zero limit switch enabled,* *// Z-Axis: E0 and EE switches enabled (default,) A-Axis: All limit switches ignored*

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

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

LSX_SetSwitchPolarity	
Description:	Sets polarity of limit switches.
C++:	int LSX_SetSwitchPolarity (int lLSID, int lXP, int lYP, int lZP, int lAP);
Parameters:	A bit mask is supplied for each axis: Bit 0 → zero limit switch (cal, "E0") Bit 1 → reference limit switch (unused) Bit 2 → end limit switch (rm, "EE") If bit is set (1), the corresponding switch is interpreted active when high. If bit is reset (0), the corresponding switch is active low.
Example:	pTango->SetSwitchPolarity(1, 7, 0, 0, 0); *// all limit switches of X-Axis are high active,* *all limit switches of Y-, Z- and A-Axis are low active*

4.10 Digital and Analog Inputs and Outputs

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

LSX_GetDigitalInputs	
Description:	Retrieve signal level of all 16 digital input pins (I/O extension).
C++:	int LSX_GetDigitalInputs (int lLSID, int *plValue);
Parameters:	***Value***: Pointer to Integer value, to which the status of all inputs is written (as bit mask). LSB = Digital input 0

Example:	int inputs;
	pTango->GetDigitalInputs(1, &inputs);
	if (Inputs & 16) ... *// if input 4 is set ...*

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

LSX_SetAnalogOutput	
Description:	Set analog output signals.
C++:	int LSX_SetAnalogOutput (int lLSID, int lIndex, int lValue);
Parameters:	***Index***: 0,1 (analog circuits) ***Value***: 0...100 [%]
Example:	pTango->SetAnalogOutput(1, 0, 100); *// set analog output 0 to max. voltage (10V)*

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

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

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

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

LSX_SetDigitalOutput	
Description:	Set individual digital output pin.
C++:	int LSX_SetDigitalOutput (int lLSID, int lIndex, BOOL bValue);
Parameters:	***Index***: 0 to 15 ***Value***: Set pin level to FALSE = low TRUE = high
Example:	pTango->SetDigitalOutput(1, 0, TRUE); *// set output pin 0 to´1´*

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

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

4.11 Encoder Settings

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

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

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

LSX_SetEncoderActive	
Description:	Retrieves which encoder is activated after calibration Please note: This function is corresponding to „!encmask“ command.
C++:	int LSX_SetEncoderActive (int lLSID, int lFlags);
Parameters:	***Value***: Encoder mask (flags) Bit 0 = X encoder will be activated Bit 1 = Y encoder will be activated Bit 2 = Z encoder will be activated

Example:	pTango->SetEncoderActive(1, 0);
	// No encoder will be used
	pTango->SetEncoderActive(1, 2);
	// encoder of Y-Axis will be activated after calibration

LSX_GetEncoderMask	
Description:	Retrieve status of encoders. Please note: This function is corresponding to „?enc“ command.
C++:	LSX_GetEncoderMask (int lLSID, int *plFlags);
Parameters:	***Flags***: 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

Example:	int EncMask;
	pTango->GetEncoderMask(1, &EncMask);
	if (EncMask & 2) ...
	// if encoder of Y-Axis connected + active ...

LSX_SetEncoderMask	
Description:	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.
C++:	int LSX_SetEncoderMask (int lLSID, int lValue);
Parameters:	***Value***: Active encoder mask (flags) Bit 0 = (activate)/deactivate X encoder Bit 1 = (activate)/deactivate Y encoder Bit 2 = (activate)/deactivate Z encoder
Example:	pTango->SetEncoderMask(1, 0); *// deactivate all encoders* pTango->SetEncoderMask (1, 2); *// deactivate X and Z encoders, activate Y-Axis encoder*

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

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

LSX_GetEncoderPosition	
Description:	Retrieves position response type.
C++:	int LSX_GetEncoderPosition (int lLSID, BOOL *pbValue);
Parameters:	***Value***: TRUE → axis position values will be read from the encoder, if activated. Else the position will be taken from the motor position. FALSE → Position will be taken from the motor position.
Example:	pTango->GetEncoderPosition(1, &Value);

LSX_SetEncoderPosition	
Description:	Switches encoder value display on / off.
C++:	int LSX_SetEncoderPosition (int lLSID, BOOL bValue);
Parameters:	***Value***: TRUE → axis position values will be read from the encoder, if activated. Else the position will be taken from the motor position. FALSE → Position will be taken from the motor position.
Example:	pTango->SetEncoderPosition(1, TRUE);

LSX_GetEncoderRefSignal	
Description:	Retrieves whether the encoder reference signal is evaluated when calibrating.
C++:	int LSX_GetEncoderRefSignal (int lLSID, int *plXR, int *plYR, int *plZR, int *plAR);
Parameters:	1 → encoder reference signal is evaluated while calibrating 0 → reference signal is not evaluated, zero position is set at the CAL end switch
Example:	pTango->GetEncoderRefSignal(1, &X, &Y, &Z, &A);

LSX_SetEncoderRefSignal	
Description:	Evaluate reference signal from encoder when calibrating.
C++:	int LSX_SetEncoderRefSignal (int lLSID, int lXR, int lYR, int lZR, int lAR);
Parameters:	***XR, YR, ZR, AR***: 0 (encoder reference signal is evaluated while calibrating) or 1 (reference signal is not evaluated, zero position is set at the CAL end switch)
Example:	pTango->SetEncoderRefSignal(1, 1, 1, 0, 0); *// when calibrating, reference signals of encoders X and Y are evaluated*

4.12 Controller Settings

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

LSX_GetController	
Description:	Retrieve Closed Loop mode.
C++:	int LSX_GetController (int lLSID, int *plXC, int *plYC, int *plZC, int *plRC);
Parameters:	***Controller mode XC, YC, ZC, AC***: 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
Example:	pTango->GetController(1, &X, &Y, &Z, &A);

LSX_SetController	
Description:	Set Closed Loop mode.
C++:	int LSX_SetController (int lLSID, int lXC, int lYC, int lZC, int lAC);
Parameters:	Controller mode ***XC, YC, ZC, AC***: 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
Example:	pTango->SetController(1, 2, 2, 0, 0); *// Enable permanent closed loop for X and Y axes*

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

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

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

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

LSX_GetControllerSteps	
Description:	Retrieves length of controller steps.
C++:	int LSX_GetControllerSteps (int lLSID, double *pdX, double *pdY, double *pdZ, double *pdA);
Parameters:	***X, Y, Z, A***: Length of controller steps [mm]
Example:	pTango->GetControllerSteps(1, &X, &Y, &Z, &A);

LSX_SetControllerSteps	
Description:	Set controller steps.
C++:	int LSX_SetControllerSteps (int lLSID, double dX, double dY, double dZ, double dA);
Parameters:	***X, Y, Z, A***: 1 - spindle pitch (values depend on dimension)
Example:	pTango->SetControllerSteps(1, 4, 5, 7, 9);

LSX_GetControllerTimeout	
Description:	Retrieves controller timeout.
C++:	Int LSX_GetControllerTimeout (int lLSID, int *plACtrTimeout);
Parameters:	***ACtrTimeout***: Timeout [ms], 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).
Example:	pTango->GetControllerTimeout(1, &ACtrTimeout);

LSX_SetControllerTimeout	
Description:	Set controller timeout.
C++:	int LSX_SetControllerTimeout (int lLSID, int lACtrTimeout);
Parameters:	***ACtrTimeout***: Timeout 0 – 10000 ms, 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).
Example:	pTango->SetControllerTimeout(1, 500); *// Abort after trying to settle in the target window for 500ms*

LSX_GetControllerTWDelay	
Description:	Retrieve controller delay.
C++:	int LSX_GetControllerTWDelay (int lLSID, int *plCtrTWDelay);
Parameters:	***CtrTWDelay***: Controller delay [ms]
Example:	pTango->GetControllerTWDelay(1, &CtrTWDelay);

LSX_SetControllerTWDelay	
Description:	Set controller delay.
C++:	int LSX_SetControllerTWDelay (int lLSID, int lCtrTWDelay);
Parameters:	***CtrTWDelay***: 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.
Example:	pTango->SetControllerTWDelay(1, 0); *// controller delay switched off, closed loop end position will be inaccurate*

LSX_GetCtrFastMove	
Description:	Retrieves setting of FastMove function.
C++:	int LSX_GetCtrFastMove (int lLSID, BOOL *pbActive);
Parameters:	***Active***: TRUE → FastMove function active
Example:	pTango->GetCtrFastMove(1, &Active);

LSX_GetCtrFastMoveCounter	
Description:	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.
C++:	int LSX_GetCtrFastMoveCounter (int lLSID, int *plXC, int *plYC, int *plZC, int *plAC);
Parameters:	***XC, YC, ZC, AC***: Number of carried out Fast Move functions
Example:	pTango->GetCtrFastMoveCounter(1, &XC, &YC,&ZC,&AC);

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

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

LSX_SetCtrFastMoveOff	
Description:	FastMove function deactivated.
C++:	int LSX_SetCtrFastMoveOff (int lLSID);
Parameters:	-
Example:	pTango->SetCtrFastMoveOff(1);

LSX_SetCtrFastMoveOn	
Description:	Activate FastMove function , meaning a new vector is started if controller position difference is larger than the lock-in range.
C++:	int LSX_SetCtrFastMoveOn (int lLSID);
Parameters:	-
Example:	pTango->SetCtrFastMoveOn(1);

4.13 Trigger Output

LSX_GetTrigCount	
Description:	Retrieve trigger counter value.
C++:	int LSX_GetTrigCount (int lLSID, int *plValue);
Parameters:	***Value***: Number of executed triggers
Example:	pTango->GetTrigCount(1, &Value);

LSX_SetTrigCount	
Description:	Set trigger counter value.
C++:	int LSX_SetTrigCount (int lLSID, int lValue);
Parameters:	***Value***: 0 to 2147483647
Example:	pTango->SetTrigCount(1, 0);

LSX_GetTrigger	
Description:	Retrieve trigger setting.
C++:	int LSX_GetTrigger (int lLSID, BOOL *pbATrigger);
Parameters:	***ATrigger:*** TRUE → trigger is ´´On´´ FALSE → trigger is ´´Off´´
Example:	pTango->GetTrigger(1, &ATrigger);

LSX_SetTrigger	
Description:	Switch trigger on / off.
C++:	int LSX_SetTrigger (int lLSID, BOOL bATrigger);
Parameters:	***ATrigger*** = TRUE → switch trigger on = FALSE → switch trigger off
Example:	pTango->SetTrigger(1, TRUE);

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

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

4.14 Snapshot-Input

LSX_GetSnapshot	
Description:	Provides current status of Snapshot.
C++:	int LSX_GetSnapshot (int lLSID, BOOL *pbASnapshot);
Parameters:	ASnapshot: TRUE → Snapshot is ´´On´´ FALSE → Snapshot is ´´Off´´
Example:	pTango->GetSnapshot(1, &ASnapshot);

LSX_SetSnapshot	
Description:	Switch Snapshot on / off.
C++:	int LSX_SetSnapshot (int lLSID, BOOL bASnapshot);
Parameters:	***ASnapshot***: TRUE → switch Snapshot ´´On´´ FALSE → switch Snapshot ´´Off´´
Example:	pTango->SetSnapshot(1, TRUE);

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

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

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

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

LSX_SetSnapshotPar	
Description:	Set Snapshot parameters.
C++:	int LSX_SetSnapshotPar (int lLSID, BOOL bHigh, BOOL bAutoMode);
Parameters:	***High***: TRUE → snapshot is high active FALSE → snapshot is low active ***AutoMode***: TRUE → snapshot ´´Automatic´´: Position is automatically moved to after first snapshot pulse.
Example:	pTango->SetSnapshotPar(1, TRUE, FALSE);

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

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

5. Error Codes

5.1 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

5.2 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)

6. 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	