MÄRZHÄUSER WETZLAR

www.marzhauser.com

TANGO DLL Documentation

Programming Interface for TANGO Controllers

Version 1.545

Documentation as of 09. September 2026

Märzhäuser Wetzlar GmbH & Co. KG · In der Murch 15 · 35579 Wetzlar · Germany · Tel. +49 6441 9116-0

Table of Contents

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, Virtual COM Port (USB, PCI and PCI-E) or Ethernet
- Supports most controller commands directly
- Up to 4 axes per TANGO
- Up to 8 TANGO controllers per DLL

1.2. System Requirements

The TANGO DLL can be used on all Windows PCs from Windows 7 to Windows 11.
It requires the *Microsoft Visual C++ 2017 Redistributable Package* to be installed, which often is already installed on Windows PCs. If not, it can be downloaded from the Microsoft website:

https://learn.microsoft.com/en-us/cpp/windows/latest-supported-vc-redist?view=msvc-170

--> 32 bit: https://aka.ms/vs/17/release/vc_redist.x86.exe
--> 64 bit: https://aka.ms/vs/17/release/vc_redist.x64.exe

1.3. Supported Development Environments

The TANGO DLL is available as 32 Bit and 64 Bit version.
It exposes a standard C function interface (flat C exports, stdcall) and can therefore be used from any development environment able to call Windows DLLs, including current Microsoft Visual Studio versions such as 2019 and 2022 with C++, C# or Visual Basic .NET.
The only prerequisite is the matching Microsoft Visual C++ runtime, independent of the development tool used.
It has been tested on operating systems Windows 7, 8, 10 and 11 using following development tools:

- Microsoft Visual Studio 2010, 2017, 2019, 2022 languages C++, C#, Visual Basic .NET
- Java
- Python
- MATLAB
- National Instruments LabVIEW
- Embarcadero Delphi 2007 and Delphi XE
- Compatibility is assumed for all other programming environments which can use a DLL.

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 the TANGO, calibrate and move, send commands, retrieve input- and output states, etc.

2.1. General Information

The DLL will require one or two more files (.h and/or.c) that are provided with the API, depending on how the DLL is used. Refer to the following examples. It is important to use the right DLL, for 64 bit programs use the 64 bit DLL and use the 32 bit version for 32 bit x86 programs.
All DLL functions return a 32-bit integer value, where 0 (zero) indicates the error free execution of a function. For other integer values refer to chapter **Error Codes**. `GetErrorString` can be used to translate an error code into a short English explanation text.

The functions described in this document (chapter **0**) use the "LSX_" commands, in which the first value stands for the TANGO ID (LSID). This ID is used to address up to 8 controllers simultaneously through one DLL. But it is recommended to use the “LS_” instructions and open the DLL for each TANGO separately. Then, in function calls the first value (the TANGO ID) is skipped, and CreateLSID / FreeLSID is not required.

Please note: instance 1 is created when connecting, that is in LS_Connect, LS_ConnectSimple and LS_ConnectEx only. An "LS_" read instruction issued before the first connect therefore returns 4003.
Use either the "LS_" or the "LSX_" instructions in a program, not both; mixing them is not encouraged.
Instance 1 is nevertheless left unused until a connect, so that the first LSX_CreateLSID returns number 1 and not number 2.

Example

"LS" Function Call:

```
LS_MoveAbs(50.0, 50.0, 50.0, 10.0, TRUE);
```

TANGO Pointer Function Call:

```
pTango->MoveAbs(50.0, 50.0, 50.0, 10.0, TRUE); // The preferred C++ solution
```

"LSX" Function Call:

```
LSX_MoveAbs(1, 50.0, 50.0, 50.0, 10.0, TRUE);  // Described under Default API
// the first value is the LSID, which is not needed with "LS_" function calls
```

With functions such as `LS_MoveAbs/LSX_MoveAbs`, values of 4 axes must be passed to the function.
If the controller only provides 1-3 axes, values of the not available axes are ignored and can be set to 0.

2.2. Default API for C++

In C++, direct access to LSX DLL functions is easily possible via the TangoLSX_API.h header file, but it is recommended to access each TANGO by an individual DLL as described in the next chapter.

Example for LSX functions: Access two TANGOs with one DLL

- The TangoLSX_API.h header file from the TANGO DLL folder must be included (by #include)
- In the C++ project, the TANGO_DLL.lib file must be added by going to: Project Properties / Linker Input / Additional Dependencies - and adding the TANGO_Dll.lib; there.
- It might be useful to copy the TANGO files from the TANGO CD or USB-Stick's "API & DLL" in a small folder structure to the project, e.g. in a „TANGO-DLL" folder with two"32" and "64" named subfolders.
  The TangoLSX_API.h directly in the TANGO-DLL folder and the corresponding 32 and 64 bit Tango_DLL.dll DLLs and their TANGO_DLL.lib files in the 32 and 64 named sub-folders. Then, the 32 and 64 bit project properties can have their TANGO_DLL.lib file added as TANGO_DLL\32\TANGO_DLL.lib and the 64 bit build as TANGO_DLL\64\TANGO_DLL.lib.

The DLL example "VS2017_Cpp_64bit" is made this way - the DLL, LIB and the API header files are there in an own "Distrib" folder, and in the Project Properties, the entry of the TANGO_DLL.lib file can be found (for 32 and for 64 bit program compile versions, the corresponding folder is specified).

```
#include "..\MyDllFolder\TangoLSX_API.h" // Include the API Header file for LSX function calls

int  IdTangoXY = 0; // To store the ID for the XY TANGO which controls the stage
int  IdTango_Z = 0; // To store the ID for the second TANGO which controls Z only
int  result;
char text[128] = "";

result = LSX_CreateLSID(&IdTangoXY); // LSX function calls require an ID (LSID) for each connection
result = LSX_CreateLSID(&IdTango_Z); // Retrieve the IDs to address the TANGOs

result = LSX_ConnectSimple(IdTangoXY, 1, "COM11", 57600, TRUE);//FALSE); // Show Protocol for COM11
result = LSX_ConnectSimple(IdTango_Z, 1, "COM7", 57600, TRUE);//FALSE); // Show Protocol for COM7

result = LSX_GetDLLVersionString(IdTangoXY, text, sizeof(text)-1); // DLL Version

result = LSX_GetTangoVersion(IdTangoXY, text, sizeof(text)-1); // Read the TANGO Version from COM11
result = LSX_GetTangoVersion(IdTango_Z, text, sizeof(text)-1); // Read the TANGO Version from COM7

result = LSX_Disconnect(IdTangoXY); // Disconnect the COM Port of the XY TANGO
result = LSX_Disconnect(IdTango_Z); // Disconnect the COM Port of the Z  TANGO

result = LSX_FreeLSID(IdTangoXY);   // LSX functions require the ID to be released at the end
result = LSX_FreeLSID(IdTango_Z);

IdTangoXY = 0;
IdTango_Z = 0;
```

2.3. Integration in Visual C++

A TANGO class is provided for C++, which 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()` or `LSX_Calibrate(lsid)`.
Only one instance of the CTango class should be created, and the TANGO DLL loaded only once.
The required files for a C/C++ Application, `TANGO.h` and `TANGO.cpp` can be found in the folder: \Software\API & DLL\DLL Files.
In some cases, the `TANGO.cpp` file must be adapted by replacing `#include "stdafx.h"` with `"pch.h"`.

Required files:

- Tango_DLL.dll
- TANGO.h
- TANGO.cpp (must also be added to the project's source code files by "add existing file")

Visual C++ example for controlling a TANGO:

```
#include "TANGO.h"
…

CTango*  pTango; // Will point to the LS_ functions, not to LSX_ with their LSIDs

pTango = new CTango();
…

pTango->ConnectSimple(1, "COM3", 57600, TRUE); // Connect to COM3
pTango->MoveAbs(30, 50, 70, 0, TRUE);          // Move
pTango->Disconnect();                          // Disconnect COM3
delete pTango;
```

2.4. Integration in C#

The TANGO DLL functions are called from C#/.NET via P/Invoke ([DllImport]). The default calling convention (Winapi = __stdcall) matches the DLL, so no extra setting is required. String parameters use a StringBuilder with CharSet.Ansi. Build the application for the matching bitness (x86 -> 32-bit DLL, x64 -> 64-bit DLL).
Required files: Tango_DLL.dll (next to the executable).

Example declarations and use:

```
[DllImport("Tango_DLL.dll")] static extern int LSX_CreateLSID(out int plID);
[DllImport("Tango_DLL.dll", CharSet=CharSet.Ansi)] static extern int LSX_ConnectSimple(int id,
int itype, string com, int baud, int showprot);
[DllImport("Tango_DLL.dll")] static extern int LSX_MoveAbs(int id, double x, double y, double z,
double a, int wait);
[DllImport("Tango_DLL.dll")] static extern int LSX_Disconnect(int id);

int id;
LSX_CreateLSID(out id);
LSX_ConnectSimple(id, 1, "COM3", 57600, 0); // Connect to COM3
LSX_MoveAbs(id, 30, 50, 70, 0, 1);          // Move
LSX_Disconnect(id);
```

Example projects: \Software\ API & DLL\DLL Examples\VS2017_Csharp\ and VS2010_Csharp\

2.5. Integration in Visual Basic

To use the functions of the TANGO DLL, the file TANGO.vb must be added to the project.
Example projects: \Software\ API & DLL\DLL Examples\Visual_Basic\SourceCode\

Required files: Tango_DLL.dll and TANGO.vb

Visual Basic example for controlling a TANGO:

```
Dim returnvalue1 As Integer
Dim returnvalue2 As Integer
Dim returnvalue3 As Integer

...
Returnvalue1 = LS_ConnectSimple(1, "COM3", 57600, 1)
Returnvalue2 = LS_MoveAbs(30, 50, 70, 0, 1)
Returnvalue3 = LS_Disconnect()
```

2.6. Integration in Delphi

The TANGO DLL functions are declared in Delphi with the stdcall calling convention. Ready-made declarations for the LS_ and LSX_ functions are provided in the file TANGO_DLL.pas.
Required files: Tango_DLL.dll and TANGO_DLL.pas.

Example declaration and use:

```
function LSX_CreateLSID(var lLSID: Integer): Integer; stdcall; external 'Tango_DLL.dll';
function LSX_ConnectSimple(lLSID, InterfaceType: Integer; ComName: PAnsiChar; BaudRate, ShowProt:
Integer): Integer; stdcall; external 'Tango_DLL.dll';

var Tango1: Integer;
LSX_CreateLSID(Tango1);
LSX_ConnectSimple(Tango1, 1, 'COM3', 57600, 1);   // Connect to COM3
LSX_MoveAbs(Tango1, 30, 50, 70, 0, True);         // Move
LSX_Disconnect(Tango1);
```

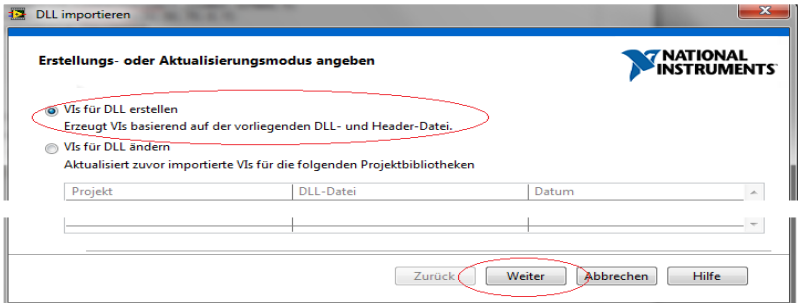
Example projects: \Software\API & DLL\DLL Examples\Delphi_7\ and Delphi_XE8\

2.7. Integration in LabVIEW

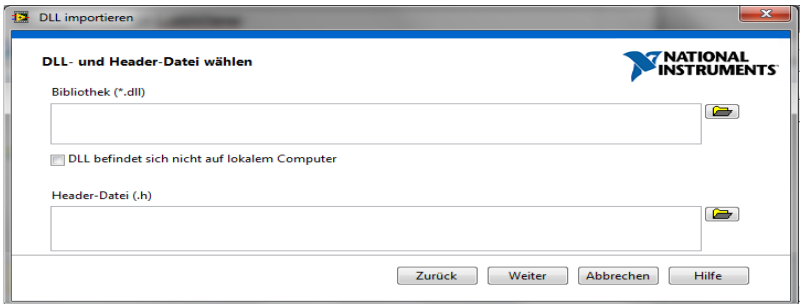
This DLL import description can be used with every LabVIEW Version, which supports DLL import functionality.

To use the TANGO DLL functions with LabVIEW, the TANGO DLL has to be imported to LabVIEW. Therefore, follow the steps listed below:

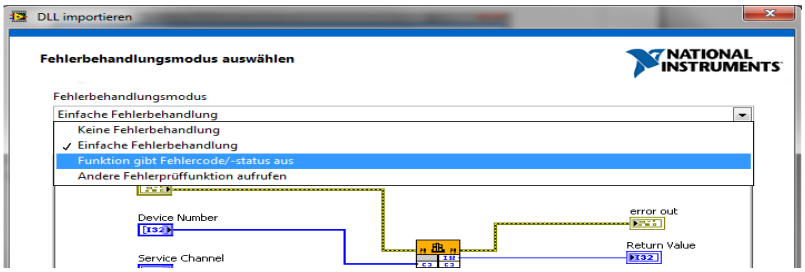
1) Start LabVIEW
2) In LabVIEW window: Tools = Import = DLL select the first radio button and press next.



3) In the 2 corresponding fields select files "Tango_DLL.dll" and "TangoLSX_API.h" from CD directory / Software / API&DLL / LabVIEW.

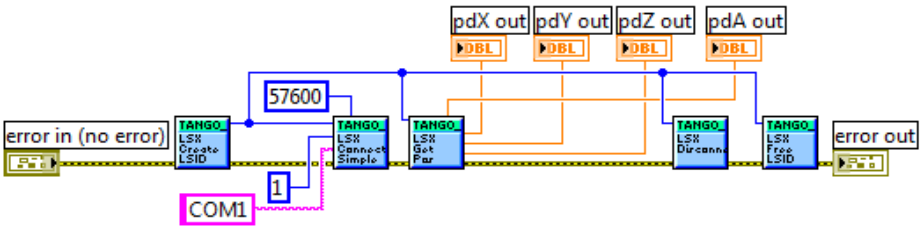


4) "Including Paths" in the next window need not to be configured.
5) In the next window the included functions of the Tango_DLL.dll are listed and selectable. It is recommended to select all functions. You may notice, that only half of the functions included in Tango_DLL.dll are found in the TangoLSX_API.h which is correct, because all functions exist in "LS_*function*" and in "LSX_*function*" notation.
6) The TangoLSX_API.h defines just the "LSX" functions, which should be preferred to use anyway.



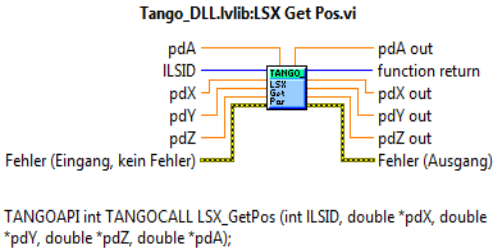
7) After selecting the path and name for the project library the error handling mode should at least contain a simple error handling or even an error handling with return function of Tango_DLL.dll included.
8) The configuration of the VIs should not be changed and the import process can start.

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.

Remark:
"Get" functions defined in Tango_DLL.dll often have pointers as parameters. These pointers 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.

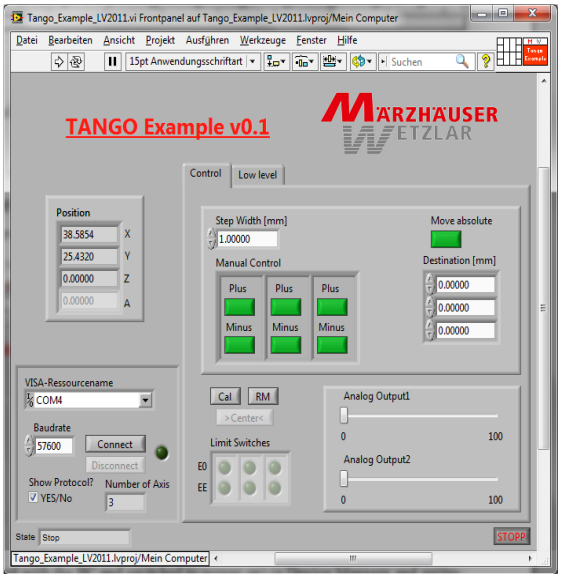


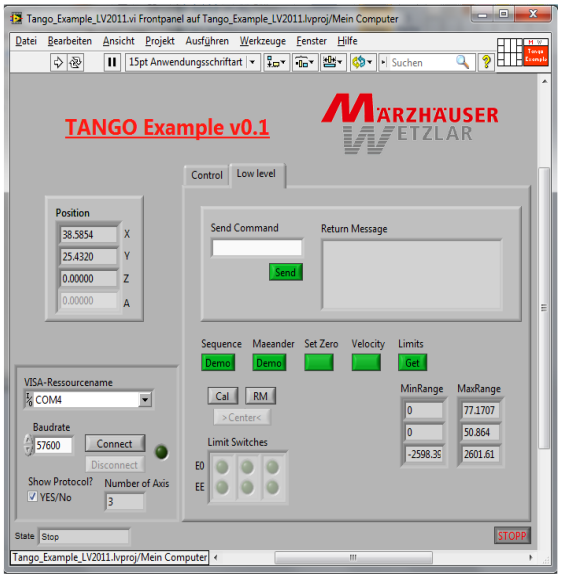
In all such "Get" VIs just connect only required output parameters. It is useless to connect input parameters because they will be ignored anyway and won't have any effect.

An example program is provided on how to control a TANGO via LabVIEW (LV2011). It is not compatible with older 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.

Example project: \Software\API&DLL\LabVIEW\TANGO_Example_LV2011\

Requires LabVIEW-Version 2011 and newer.





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-Ressourcename 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 “Meander”
- 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)

2.8. Integration in MATLAB

The TANGO DLL can be used in MATLAB. Therefore,

- the Tango_DLL.dll (usually the 64 Bit version)
- and the TANGO_DLL.h

must be included in the project (folder).

Example project: \Software\ API & DLL\DLL Examples\MatLab\

2.9. Integration in Python

The TANGO DLL is loaded in Python with the ctypes module. Because the DLL functions use the __stdcall calling convention, use ctypes.WinDLL (not CDLL). String parameters are passed as byte strings, pointer parameters via ctypes.byref(). Use a Python interpreter whose bitness matches the DLL (32/64 bit). Required files: Tango_DLL.dll.

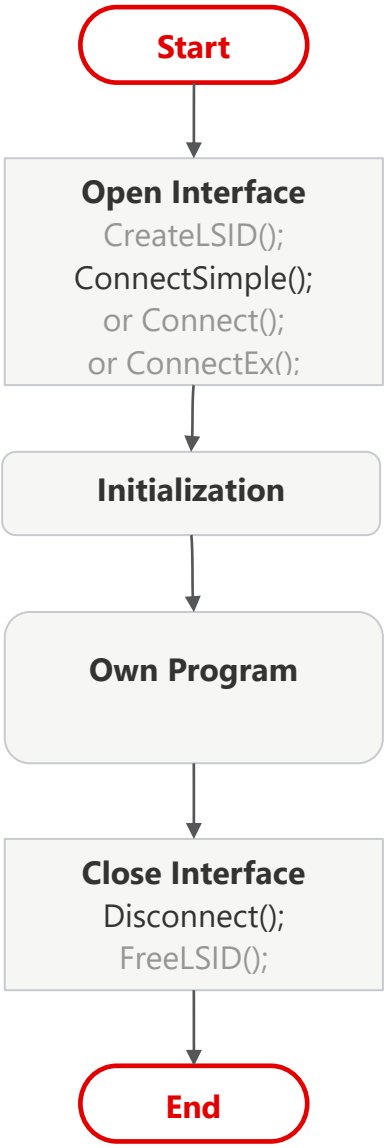
Example:

```
from ctypes import *
m = WinDLL("./Tango_DLL.dll")
lsid = c_int()
m.LSX_CreateLSID(byref(lsid))
m.LSX_ConnectSimple(lsid, 1, b"COM3", 57600, 0)      # Connect to COM3
m.LSX_MoveAbs(lsid, c_double(30), c_double(50), c_double(70), c_double(0), 1)   # Move
m.LSX_Disconnect(lsid)
```

Example project: \Software\ API & DLL\DLL Examples\Python\

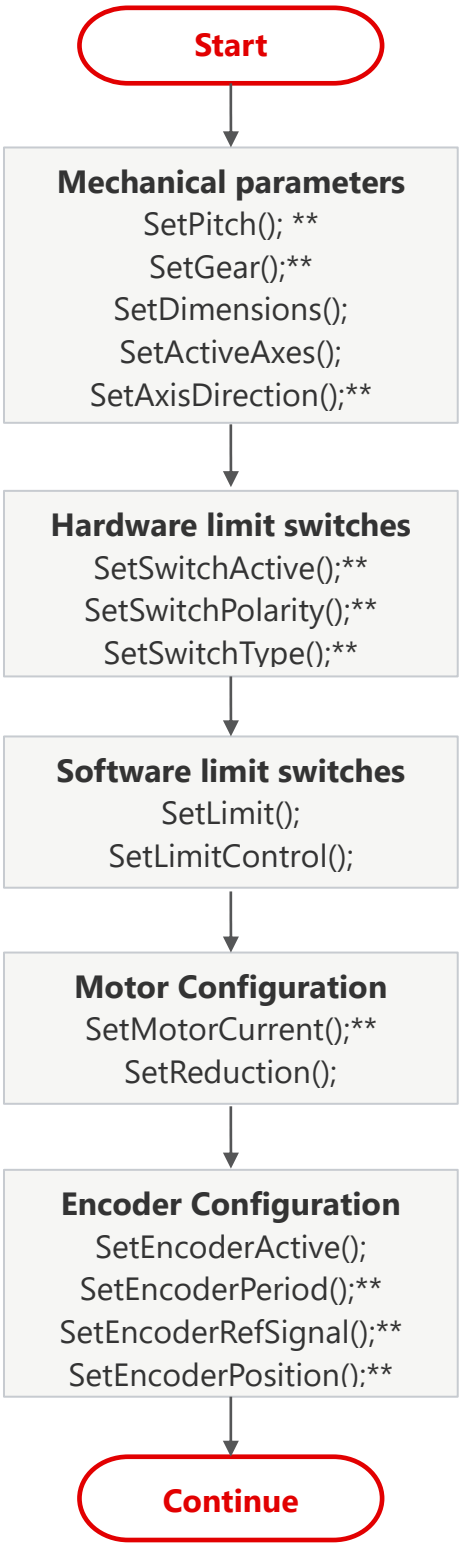
3. Getting Started

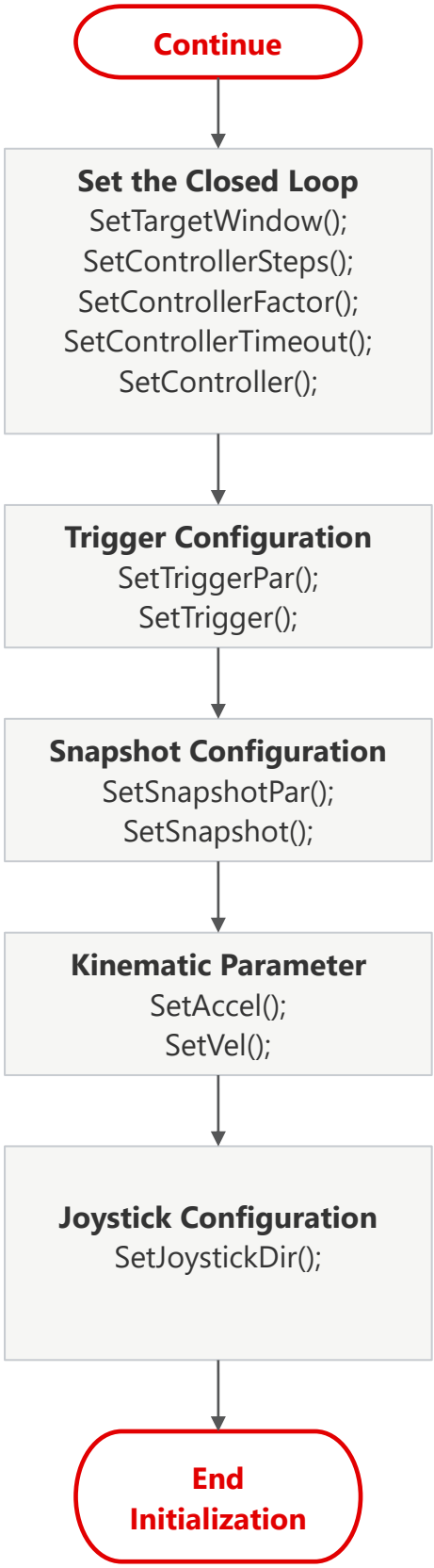
The following flow chart shows how to establish and end a DLL communication to the TANGO and is valid for all communication interfaces like RS232, USB, PCI, PCI-E or Ethernet.
The guideline is equal for all possible programming languages.
The available DLL functions are listed and described in the following chapters. For further information, the TANGO Instruction Set Documentation can be used. Therefore, the DLL function description also includes the TANGO instruction that stands behind the function call, shown in a separate TANGO CMD line, e.g. !pitch for SetPitch.



3.1. Controller Initialization

Most Märzhäuser stages and axes are ETS coded. The TANGO then uses the ETS data for initialization.
The affected settings become write protected, meaning they cannot and do not need to be modified: **See ****.
Märzhäuser products will run “from scratch”, if connected to a TANGO. Individual settings like the measuring unit (SetDimension), velocity, closed loop etc. could be made as shown on the next page.
Note: Mechanics may be damaged if wrong parameters are used. If no Märzhäuser axis is used, please take care the correct settings (******) for the axes are made (e.g. pitch, gear, direction, limit switches, encoder).




Continue
Set the Closed Loop
SetTargetWindow();
SetControllerSteps();
SetControllerFactor();
SetControllerTimeout();
SetController();
Trigger Configuration
SetTriggerPar();
SetTrigger();
Snapshot Configuration
SetSnapshotPar();
SetSnapshot();
Kinematic Parameter
SetAccel();
SetVel();
Joystick Configuration
SetJoystickDir();
End Initialization

3.2. Reconnecting after a Reset or Power Cycle

A TANGO reboots after !reset or other instructions that cause a reboot, and equally after being switched off and on again or after a USB cable has been unplugged and reconnected. The DLL does not drop the connection in these situations. It keeps the port and waits for the controller to come back, polling it until it answers.

How long the DLL waits

The waiting time is not a fixed value. From firmware 1.84 onwards the DLL adapts it to the controller: when the connection is established, it asks how long the last start-up took. The command used for this is ?boottime, which appears in the protocol window together with the result as --> Controller boot time: X ms. After a reset the DLL then waits a base of three seconds plus twice that boot time.
Twice, because a start-up step that has to be repeated can double the boot time, and the controller only ever reports the duration of its last start. The margin costs nothing in normal operation: the value is an upper limit, not a delay. The DLL continues the moment the controller answers.
How long a controller needs to start differs considerably, by controller type and, for identical hardware, also by the way it is configured. This is why the waiting time is derived from the controller itself instead of being fixed.

Upper limits

If the boot time is unknown, because the firmware does not support the query or the answer is not plausible, the DLL waits up to 20 seconds. A boot time reported by the controller is honoured beyond that, up to a hard limit of 60 seconds.
A limit is necessary because at some interfaces the port gives no indication of whether the controller is still there: an RS232 port, or the virtual COM port of a USB adapter, can be opened even when the controller has been removed or replaced. If the port cannot be opened at all, the DLL reports E_4005 after three seconds, or 7.5 seconds for Ethernet, exactly as before. A reconnect therefore never runs indefinitely.
SetAbortFlag ends the waiting immediately. This is the intended way for an application to leave a reconnect early, for example when the user cancels.

Older firmware

Below firmware 1.84 the boot time cannot be queried, and the DLL behaves exactly as it did before: three seconds for serial and USB connections, 7.5 seconds for Ethernet.

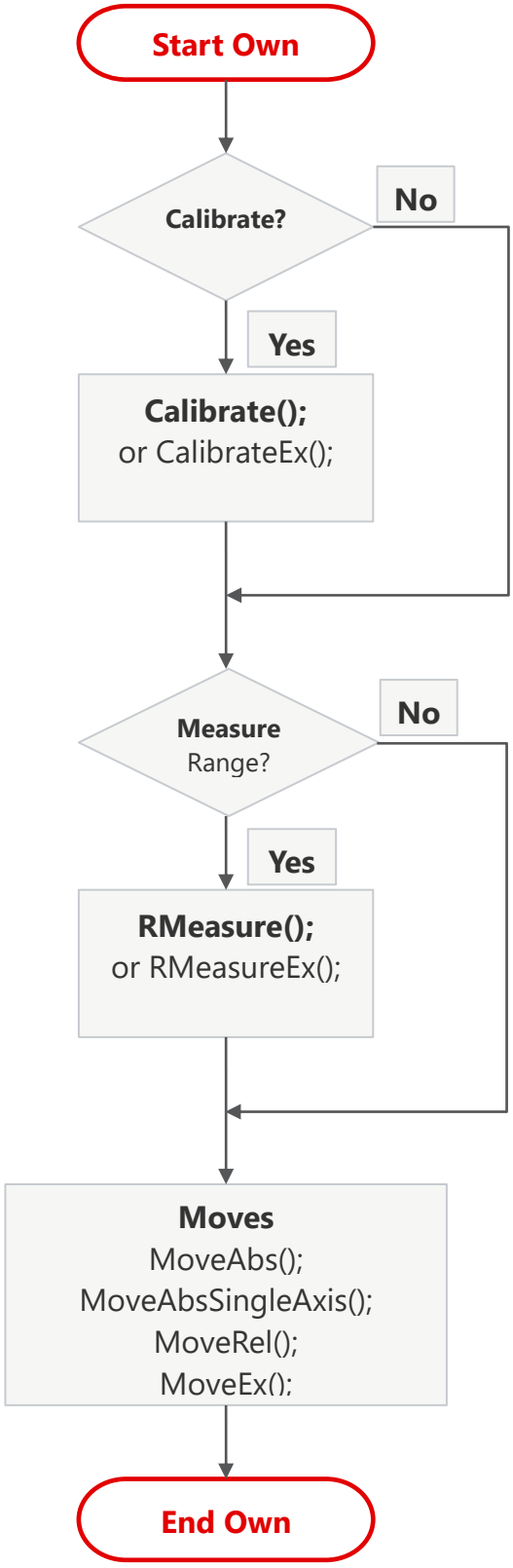
What this does not cover

A firmware update is not part of the DLL functionality. The Firmware Update Tool and SwitchBoard will take care of it when performing a TANGO firmware update.

The automatic reconnect only works while the connection is still open. If the application calls LSX_Disconnect (or LS_Disconnect) itself, the instance is torn down and no reconnect is to be expected - this is by design. In that case the application has to connect again on its own.

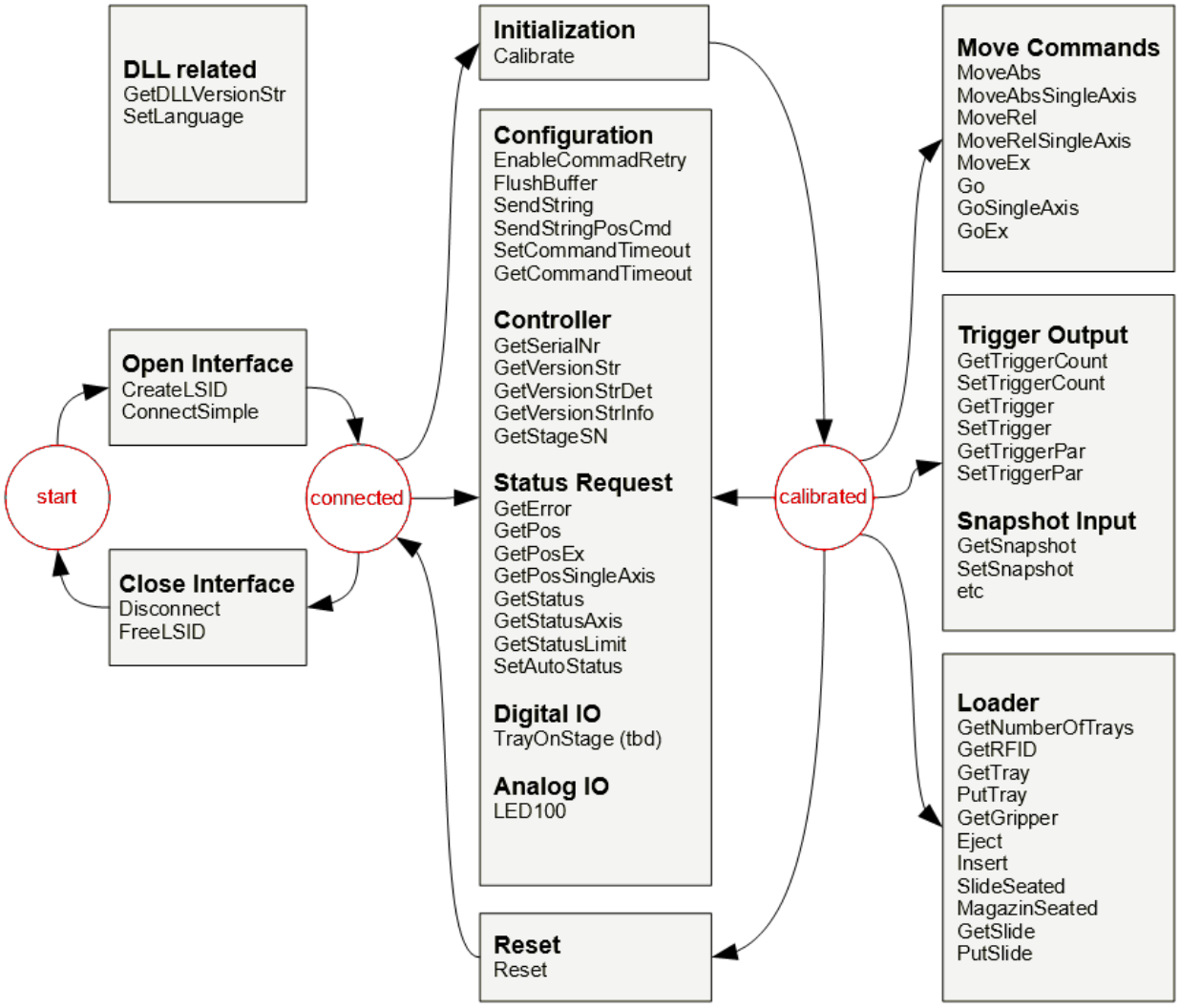
3.3. Use Case Example

In their own program, the user can implement the 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.



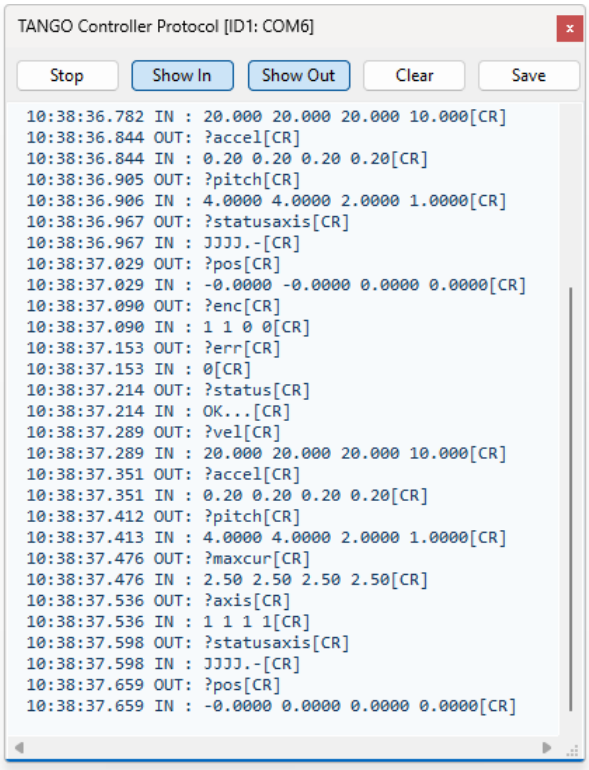
3.4. API State Diagram

The API state shows which DLL functions usually require an initialisation as precondition. This means the axes must be moved at least to a reference point. Usually, the lower limit switch is used as reference.



3.5. Protocol Window

The protocol window logs the complete communication with the controller: every command the DLL sends and every reply the TANGO returns, line by line and with a timestamp (different timestamps are available). It is the quickest way to find out what an application really sends and how the controller answers.
Each TANGO-ID has its own protocol window, so several controllers can be monitored at the same time.
The protocol window keeps displaying while the calling application is busy, even inside a blocking DLL call.
The protocol can follow even fast communication and only has minimal impact on throughput.



Switching the window on and off

The window can be opened directly when connecting, using the bShowProt parameter of ConnectSimple, Connect or ConnectEx, or at any time with SetShowProt. The value passed to SetShowProt also selects the

timestamp format:	0	Protocol window is switched off
	1	Time of day (default)
	2	Time difference to the previous line
	3	No timestamp

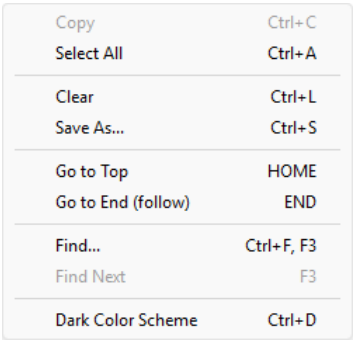
Values from 11 to 13 select the dark color scheme, with the same timestamp modes as 1 to 3 (also switchable during operation with Ctrl+D). Any negative value is treated as 1, so a caller whose TRUE represents 1 or 0xFFFFFFFF gets the normal light window. The color scheme can also be changed while the window is open: a further call to SetShowProt switches the colors immediately and keeps the contents of the window, and Ctrl+D does the same directly in the window. Only the text area is dark - title bar, buttons and scroll bars keep the colors of the operating system.

ClearProtocolWindow deletes the content of the window. SetLanguage switches the language of the window, its context menu and the search dialog (English, German, French).

Operating the window

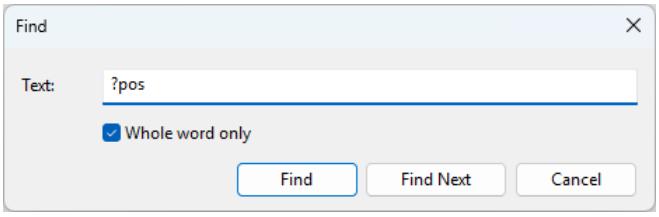
The buttons above the protocol are: Stop (pause logging), Show In and Show Out (log received or sent data), Clear, and Save.
The text can be selected freely with the mouse. A triple click, or a click into the narrow bar at the left edge, selects a complete line. The display follows the running communication as long as the view is at the bottom; scrolling up pauses it, scrolling back to the end lets it follow again.

Context menu



A right click into the protocol window opens the context menu for Copy, Find, etc.

Searching



"Find..." or (Ctrl+F) opens a dialog for text search. The search starts at the current position and wraps around to the top after the last line. The hit is selected and scrolled into view.
Find Next in the dialog jumps to the following hit and leaves the dialog open, so a protocol can be stepped through hit by hit. Find Next in the context menu repeats the last search without opening the dialog, but it can also be performed by F3 or by the Find Next button.
The "Whole word only" option narrows down a search, e.g. searching for ?pos still finds ?pos and ?pos x, but no longer ?posshift.

Keyboard shortcuts

When the protocol window has the input focus, it supports these typical Windows shortcuts:

- Ctrl+A Select all
- Ctrl+C Copy the selection
- Ctrl+F Find
- F3 Find Next
- Ctrl+L Clear the entire protocol
- Ctrl+S Save As
- Ctrl+D Switch between the light and the dark color scheme
- HOME Go to Top
- END Go to End and follow again

The Delete key and Ctrl+X are not assigned. The protocol is read-only, so nothing can be cut or deleted by accident, and clearing it is a deliberate step: Ctrl+L or Clear from the context menu.
Remarks: An application whose message loop redirects keyboard input to its own windows can intercept those shortcuts before they arrive; the buttons and the context menu are unaffected and remain available in such a case.

3.6. Sending Macros

A TANGO macro is command text that the controller stores instead of executing it. When that text is transferred with SendString, the DLL treats it like any other command: after a set command it reads a few values back to keep its own state up to date. Those extra requests would be inserted into the text stream and would end up inside the macro.

Pure text mode suppresses them. It is not a separate function: it is switched with ConnectSimple, by passing an interface type instead of a real interface. The existing connection is kept and is not re-opened.
101 = enable pure text mode. 100 = switch it off again and return to a normal connection.
Both values require an existing connection; without one, 4006 is returned. The change is written into the protocol window, so it can still be seen in a saved protocol.

Always switch back with 100 once the transfer is finished. While pure text mode is active the DLL no longer follows up commands such as encpos, autostatus, configmaxaxis or reset, so its internal state can become outdated.

Example:

```
// switch from a regilar connection to pure text mode, the connection stays open
LSX_ConnectSimple(Tango1, 101, NULL, 0, 0);

// transfer the macro text, line by line
LSX_SendString(Tango1, pcMacroLine, NULL, 0, FALSE, 0);

// switch back to a normal connection
LSX_ConnectSimple(Tango1, 100, NULL, 0, 0);
```

4. DLL-Functions

4.1. Quick Reference

DLL Configuration / Interface:

Controller information:

Status Requests:

Function	Brief Description	Page
GetError	Retrieves error state of the TANGO (error number as listed in chapter 9.1)	51
GetErrorString	Retrieves information text about the specified error number	51
GetPos	Read the position of all axes	51
GetPosEx	Read encoder- or motor-positions of all axes	51
GetPosSingleAxis	Read position of one axis	52
GetStatus	Retrieves current Controller status	53
GetStatusAxis	Read status of one axis	54
GetStatusLimit	Read status of software limits of all axes	54
GetStA	Read the detailed axis state (flags that include almost all states at once)	55
SetAutoStatus	Switches Auto-Status mode (e.g., status reply on/off)	56
GetAutoStatus	Read current Auto-Status mode	56
IsVel	Read actual velocities at which the axes are currently travelling	57
IsVelSingleAxis	Read actual velocity of specified axis	57

Controller Settings:

Function	Brief Description	Page
GetPowerAmplifier	Retrieves actual state of power amplifier	58
SetPowerAmplifier	Set required state of power amplifier (on/off)	58
GetActiveAxes	Retrieve axes state	58
SetActiveAxes	Set axes state	59
GetAxisDirection	Read axis direction	59
SetAxisDirection	Set axis direction	59
GetCalibOffset	Read calibration offset	60
SetCalibOffset	Set calibration offset	60
GetRMOffset	Read range measure offset	60
SetRMOffset	Set range measure offset	61
GetCalibBackSpeed	Read calibration backward speed	61
SetCalibBackSpeed	Set calibration backward speed	61
GetCalibrateDir	Read calibration direction	62
SetCalibrateDir	Set calibration direction	62
GetDimensions	Read the measuring units of all axes	63
SetDimensions	Set measuring units of all axes	63
GetDimensionsSingleAxis	Read measuring unit (dimension) of a single axis	64
SetDimensionsSingleAxis	Set measuring unit (dimension) of a single axis	64
GetResolution	Get digits after decimal point	64
SetResolution	Set digits after decimal point	65
GetPitch	Read actual lead screw pitch	65
SetPitch	Set required lead screw pitch	66
GetPitchSingleAxis	Read lead screw pitch of a single axis	66

Move Commands and Position Management:

Function	Brief Description	Page
StopAxes	Stops all moving axes	94
StopAxesEx	Stop the specified axis	94
WaitForAxisStop	Function returns as when all axes in bit mask have stopped/arrived	95

HDI Input Devices (Joystick etc.):

Function	Brief Description	Page
SetJoystickOff	Globally disable HDI device (e.g., joystick)	96
SetJoystickOn	Globally enable HDI device (e.g., joystick)	96
GetJoystickDir	Read HDI (joystick) directions	97
SetJoystickDir	Set HDI (joystick) directions (per axis: default, reversed off)	97
GetJoystickDirSingleAxis	Read joystick direction of a single axis	98
SetJoystickDirSingleAxis	Set joystick direction of a single axis	98
GetJoyChangeAxis	Read joystick X-Y assignment swap state	99
JoyChangeAxis	Set joystick X-Y assignment swap state	99
GetJoystick	Read joystick status	98
GetHandWheel	Not supported by TANGO: Read handwheel status	99
SetHandWheelOff	Not supported by TANGO: Switches handwheel off	100
SetHandWheelOn	Not supported by TANGO: Switches handwheel on	100
GetJoystickWindow	Read analog joystick window (not used for digital HDI)	100
SetJoystickWindow	Set analog joystick window (not used for digital HDI)	101
GetHwFactor	Read handwheel factor of all axes	101
SetHwFactor	Set handwheel factor of all axes	101
GetHwFactorSingleAxis	Read handwheel factor of one axis	102
SetHwFactorSingleAxis	Set handwheel factor of one axis	102
GetHwFactorB	Read second handwheel factor of all axes	102
SetHwFactorB	Set second handwheel factor of all axes	103
GetHwFactorBSingleAxis	Read second handwheel factor of one axis	103
SetHwFactorBSingleAxis	Set second handwheel factor of one axis	103
GetZwTravel	Read z-wheel (multifunction wheel) travel distances	104
SetZwTravel	Set z-wheel (multifunction wheel) travel distances	104
GetHdiKeys	Read key states	105
GetKey	Read key states	105
GetKeyLatch	Read and clear latched key states	105
ClearKeyLatch	Clear latched key states of an individual key or of all keys	105
GetHdiSpeedIndex	Read selected HDI speed index (switched by HDI keys or by Set)	106
SetHdiSpeedIndex	Manipulate HDI speed index	106
GetHdiSpeedIndexSingleAxis	Read selected HDI speed index of the specified axis	106
SetHdiSpeedIndexSingleAxis	Manipulate HDI speed index of the specified axis	107

Custom Control Console with Trackball and Joyspeed Keys (BPZ device):

Function	Brief Description	Page
GetBPZ	Read status of control console	108
SetBPZ	Switches control console on / off	108
GetBPZJoyspeed	Read control console joystick speed	108
SetBPZJoyspeed	Set control console joystick speed	109
GetBPZTrackballBackLash	Read control console trackball backlash	109
SetBPZTrackballBackLash	Set control console trackball backlash	109
GetBPZTrackballFactor	Read control console trackball factor	110
SetBPZTrackballFactor	Set control console trackball factor	110

Limit Switches, Hard and Soft Limits:

Function	Brief Description	Page
GetAutoLimitAfterCalibRM	Retrieves, whether internal software limits are set when calibrating or measuring stage travel range	111
SetAutoLimitAfterCalibRM	Prevents setting internal software limits by calibration or range measure	111
GetLimit	Retrieves travel range limits of single axes	111
SetLimit	Sets travel range limits of single axes	112
GetLimitControl	Retrieves whether area control is switched on or off	112
SetLimitControl	Switches area control on / off	112
GetSwitchActive	Retrieves, whether limit switches are active	113
SetSwitchActive	Enable/disable limit switches	113
GetSwitchPolarity	Retrieves polarity of limit switches	114
SetSwitchPolarity	Sets polarity of limit switches	114
GetSwitchType	Retrieves status of pull up or pull-down resistor array (NPN or PNP)	115
SetSwitchType	Set resistor pull-up or pull down to match NPN or PNP switches	115
GetSwitches	Retrieves status of all limit switches	116

Digital and Analog Inputs and Outputs:

Function	Brief Description	Page
GetAnalogInput	Retrieves current level of analog input signals	117
SetLedBright	Set the brightness of the LED100 illumination OFF/0-100%	118
SetAnalogOutput	Set analog output voltage	117
GetAnalogOutputMode	Read analog output anamode function	118
SetAnalogOutputMode	Set analog output anamode function	118
SetAuxDigitalOutput	Set individual digital outputs of AUX-I/O connector	119
GetDigitalInputs	Retrieve all digital input pin levels of IO1 interface	120
GetDigitalInputsE	Retrieve digital inputs of IO2 / Multi-IO interface	121
SetDigitalOutput	Set individual digital output of IO1-Module	121
SetDigitalOutputs	Set digital outputs 0-7 of IO1-Module	121
SetDigitalOutputE	Set individual digital output of IO2 / Multi-IO module	122
SetDigitalOutputsE	Set digital outputs 0-7 of IO2 / Multi-IO module	122
SetDigIO_Distance	Not supported by TANGO: Activate an output, depending on set distance before or after reaching determined position	123
SetDigIO_EmergencyStop	Not supported by TANGO: Assign Emergency-Stop pin	123
SetDigIO_Off	Not supported by TANGO: Switch off digital I/O functionality	124
SetDigIO_Polarity	Not supported by TANGO: Set polarity	124

TVR Clock & Direction Input and Output:

Function	Brief Description	Page
GetTVRMode	Read TVR mode	125
SetTVRMode	Set TVR mode	125
GetFactorTVR	Read AUX-IO TVR clock and direction input factor	125
SetFactorTVR	Set AUX-IO TVR clock and direction input factor	126

Encoder Settings:

Function	Brief Description	Page
ClearEncoder	Set encoder TTL-count position to zero	127
GetEncoder	Read all encoder TTL-count positions	127
GetEncoderActive	Read which encoder is activated after calibration (*"encmask"!*)	127
SetEncoderActive	Select encoder to be activated after calibration (*"encmask"*)	128
GetEncoderMask	Read current activation status of encoders (*"enc" command!*)	128
SetEncoderMask	Activates / deactivates encoders (*"enc"*)	129
GetEncoderSingleAxis	Read the main settings of the encoder (type, period, ref-signal)	129
SetEncoderSingleAxis	Set the main settings of the encoder (type, period, ref-signal)	130
GetEncoderPeriod	Read length of encoder signal period	130
SetEncoderPeriod	Set length of encoder period	131
GetEncoderPeriodSingleAxis	Read encoder period of a single axis	131

Function	Brief Description	Page
SetEncoderPeriodSingleAxis	Set encoder period of a single axis	131
GetEncoderPosition	Retrieves, whether encoder- or motor-position is displayed	132
SetEncoderPosition	Switches encoder value display on / off	132
GetEncoderRefSignal	Read if reference signal from encoder shall be used when calibrating	132
SetEncoderRefSignal	Set if encoder reference signal shall be used when calibrating	133
GetRefSpeed	Read velocity for searching the encoder reference mark (old cmd.)	133
SetRefSpeed	Set velocity for searching the encoder reference mark (old cmd.)	133

Closed Loop Settings:

Function	Brief Description	Page
GetController	Read controller mode	134
SetController	Set controller mode	134
GetControllerCall	Read controller call interval	135
SetControllerCall	Set controller call time	135
GetControllerFactor	Read setting of closed loop controller factor (old, backward compatib.)	135
SetControllerFactor	Set closed loop controller factor (old, backward compatible)	136
GetControllerFactorSingleAxis	Read setting of closed loop controller factors (recommended function)	136
SetControllerFactorSingleAxis	Set closed loop controller factor (recommended function)	136
GetControllerSteps	Read controller steps	137
SetControllerSteps	Set controller steps	137
GetControllerTimeout	Read setting of closed loop control global timeout	137
SetControllerTimeout	Set closed loop control global timeout	138
GetControllerTWDelaySingleAxis	Read closed loop control time to remain in target window of individual axes	138
SetControllerTWDelaySingleAxis	Set closed loop control time to remain in target window of individual axes	138
GetControllerTWDelay	Old: Read closed loop control time to remain in target window	139
SetControllerTWDelay	Old: Set closed loop control time to remain in target window	139
GetTargetWindow	Read target windows of all axes	139
SetTargetWindow	Set controller target windows	140
GetTargetWindowSingleAxis	Read target window of a single axis	140
SetTargetWindowSingleAxis	Set target window of a single axis	140
SetCtrFastMoveOff	Switch off FastMove function	141
SetCtrFastMoveOn	Switch on FastMove function	141

Function	Brief Description	Page
GetCtrFastMove	Read whether FastMove function is switched on or off	141
GetCtrFastMoveCounter	Read number of executed FastMove functions	142
ClearCtrFastMoveCounter	Resets number of executed FastMove functions to 0	142

Trigger Output:

Function	Brief Description	Page
GetTriggerPositionList	Read the trigger position list (for trigger modes 20,21)	152
SetTriggerPositionList	Set individual trigger positions, trigger mode turns to 20,21 autom.	152
GetTriggerPositionListIndex	Read the current index of the position list (where the trigger is)	152
SetTriggerPositionListIndex	Manipulate the current trigger list index	153
GetTriggerPositionListEntries	Read number of position entries in the trigger list (of mode 20,21)	153
SetTriggerPositionListEntries	Delete the list (0) or reduce the number of list entries	153
GetTriggerLevel	Read the trigger level for trigger modes 20,21	154
SetTriggerLevel	Set the trigger level for trigger modes 20,21 to active high or low	154
ForceTriggerPulse	Fire the trigger signal manually (0 = one pulse, 1...127 = pulse train)	154

Snapshot-Input:

Function	Brief Description	Page
GetSnapshot	Retrieve current on/off status of Snapshot	155
SetSnapshot	Switch Snapshot on / off	155
GetSnapshotMode	Retrieve Snapshot mode	155
SetSnapshotMode	Set Snapshot mode	156
GetSnapshotCount	Read Snapshot counter (number of PosArray entries)	156
SetSnapshotCount	Set Snapshot counter to less entries (truncate/discard the last entries)	156
GetSnapshotFilter	Retrieve input filter debounce delay	156
SetSnapshotFilter	Set input filter debounce delay	157
GetSnapshotPar	Retrieve Snapshot parameters (signal polarity and modes 0,1)	157
SetSnapshotPar	Set Snapshot parameters (signal polarity and modes 0,1)	158
GetSnapshotPos	Retrieve current Snapshot position	158
GetSnapshotPosArray	Retrieve a Snapshot position from the position array	159
SetSnapshotPosArray	Add or change a position of the position array	159
ClearSnapshotPosArray	Delete all position array entries	160
GetSnapshotIndex	Read Snapshot index (current pointer position in array (0...n-1)	160
SetSnapshotIndex	Set Snapshot index (current pointer position in array (0...n-1)	160

SlideExpress Interface:

Function	Brief Description	Page
Eject	Eject magazines	162
Insert	Magazines are inserted and tested if seated on which slides are present.	162
SlideSeated	Query if slide is present (seated) or not or unknown.	162
MagazinSeated	Query if magazine is present (seated) or not or unknown.	163
GetGripper	Set input filterQuery gripper status information. Returns status of gripper 1 and 2.	163
SetGripper	Set gripper status information. (Possibly useful for slide sorting tasks)	164
GetClipType	Read the clip type that is currently in the gripper	164
GetSlide	Get slide(s) from addressed position in magazine or priority handler	164
PutSlide	Put slide(s) back to addressed position in magazine or priority handler	165
GetPrioHandlerPosition	Query actual priority handler position.	165
SetPrioHandlerPosition	Enables user to shift priority handler to required position. Handler is locked at destination or after 30s timeout	165

TrayExpress Interface:

Function	Brief Description	Page
Eject	Eject magazine	166
Insert	Magazine is inserted and tested if seated and which trays are present	166
GetGripper	Retrieve gripper status, e.g. which tray is gripped	167
SetGripper	Set gripper status information	168
GetTray	Get tray from a magazine slot and put it e.g. under the microscope	168
PutTray	Put tray back to a magazine slot	169
GetRFID	Retrieve RFID of addressed tray (if properly seated in magazine)	169
GetNumberOfSlots	Retrieve max available number of slots in magazine	170
GetNumberOfMagazins	Retrieve max available number of magazines	170

Additional Handling System Functions:

Function	Brief Description	Page
GetLoaderType	Read configured type of handling system	171
GetNumberOfRows	Read available number of slide or tray slots	171
GetNumberOfColumns	Read available number of slides per tray or row	171
GetTraySN	Read serial number of the tray	172
GetTrayType	Read type of tray	172
SetTrayType	Set type of tray	172
SetCabinLED	Switch cabin LED on or off	173
GetCabinLED	Read state of cabin LED	173

Function	Brief Description	Page
SetLabelLED	Switch barcode LED on or off	173
GetLabelLED	Read state of barcode LED	174

xPos Module Functions:

Function	Brief Description	Page
Xpos3_GetPosSingleAxis	Read axis position from xPos module	175
Xpos3_SetPosSingleAxis	Set axis position of xPos module	175
Xpos3_MoveAbsSingleAxis	Move xPos module axis to a position	175
Xpos3_MoveRelSingleAxis	Move xPos axis by relative distance	176
Xpos3_GetVelSingleAxis	Read move velocity of an xPos axis	176
Xpos3_SetVelSingleAxis	Set move velocity of an xPos axis	176
Xpos3_GetSpeedSingleAxis	Read speed-move velocity of an xPos axis	177
Xpos3_SetSpeedSingleAxis	Start a speed move of an xPos axis	177
Xpos3_GetAccelSingleAxis	Read acceleration of an xPos axis	177
Xpos3_SetAccelSingleAxis	Set acceleration of an xPos axis	178
Xpos3_GetStopAccelSingleAxis	Read emergency-stop deceleration of an xPos axis	178
Xpos3_SetStopAccelSingleAxis	Set emergency-stop deceleration of an xPos axis	178
Xpos3_GetPitchSingleAxis	Read lead-screw pitch of an xPos axis	179
Xpos3_SetPitchSingleAxis	Set lead-screw pitch of an xPos axis	179
Xpos3_GetGearSingleAxis	Read gear ratio of an xPos axis	179
Xpos3_SetGearSingleAxis	Set gear ratio of an xPos axis	180
Xpos3_GetAxisDirSingleAxis	Read axis direction of an xPos axis	180
Xpos3_SetAxisDirSingleAxis	Set axis direction of an xPos axis	180
Xpos3_AbortSingleAxis	Abort a move on one or all xPos axes	181
Xpos3_GetStatusAxisSingleAxis	Read the state of a single xPos axis	181
Xpos3_GetStatusBits	Read the internal xPos module status bits	181
Xpos3_GetTemp	Read the xPos board temperature [degC]	182
Xpos3_GetTempMCU	Read the xPos CPU temperature [degC]	182
Xpos3_GetError	Read the xPos module error number	182
Xpos3_ClearError	Clear the xPos module error state	182

4.2. DLL Configuration / Interface

CreateLSID

When using LSX functions, CreateLSID must always be the first command before establishing a new connection. CreateLSID requests a unique ID from the DLL, which must be used as first parameter of all LSX functions to identify the connection.
This way, up to eight individual TANGO controllers can be accessed through one DLL.
After disconnecting, a call to FreeLSID frees the occupied ID for further connections.
(When using the LS functions, only one TANGO can be connected and the LSID parameter is neither required nor available in the LS function calls)

SYNTAX

```
int LSX_CreateLSID (int *plLSID);
```

TANGO CMD

-

PARAMETERS

LSID: Returns a new TANGO ID-Number (1 to 8) after calling CreateLSID. If 0 is returned, then no new ID could be created (all IDs already occupied).
The returned ID must be used for all subsequent commands belonging to this device.

EXAMPLE

```
int Tango1, Tango2;
LSX_CreateLSID(&Tango1);   // create ID for first TANGO
LSX_CreateLSID(&Tango2);   // create ID for second TANGO
```

ConnectSimple

The default way to connect to a TANGO controller.
(other options to connect are: ConnectEx or LoadConfig+Connect).
In case of LSX functions, CreateLSID() must be called before connecting.
The DLL can connect to RS232, USB and PCI/PCI-E ports via a COM port interface
either by specifying the COM port number and using InterfaceType = 1
or by auto-connect to the first TANGO USB/PCI/PCI-E interface the DLL finds on the computer:
Then use InterfaceType -1
The DLL can also connect to a TANGO Desktop HE via Ethernet (IPv4).

- Therefore, instead of the ComName must contain the TANGO IP-Address xxx.xxx.xxx.xxx instead of COMx and the InterfaceType must be 6 instead of 1.
- In the special case of "Bootloader Connect" (InterfaceType 5, the DLL decides automatically between the interfaces COM or Ethernet.

SYNTAX

```
int LSX_ConnectSimple (int   lLSID,
                       int   lAnInterfaceType,
                       char *pcAComName,
                       int   lABaudRate,
                       int   lAShowProt);
```

TANGO CMD -

PARAMETERS

lAnInterfaceType: Interface type = 1 (always 1 for RS232, PCI, PCI-E and USB)
Interface type = -1 (connects the DLL to the first USB or PCI TANGO found on the computer, without specifying a COM port)
Interface type = 6 (connects the DLL via Ethernet)
Interface type = 101 (switches an existing connection into pure text mode, for transferring macro text, see "Sending Macros")
Interface type = 100 (switches pure text mode off again and returns to a normal connection)

pcAComName: Name of COM-Interface, e.g. "COM2"
lABaudRate: e.g. 57600 Baud (only used for RS232, else don't care)
lAShowProt (int / BOOL):

0 or FALSE	= hide;
1 or TRUE	= show with timestamp (default);
2	= show with time difference;
3	= show without timestamp.

The dark color scheme uses the same timestamp modes, offset by ten:

11	= dark with timestamp;
12	= dark with time difference;
13	= dark without timestamp.

Any negative value is treated as 1 (show with timestamp), so a caller whose TRUE represents 1 or 0xFFFFFFFF gets the normal light window. That covers Delphi LongBool, Java via JNA and Visual Basic.

EXAMPLE

```
LSX_ConnectSimple(Tango1, 1, "COM2", 57600, TRUE); // Connect to COM2
LSX_ConnectSimple(Tango1, -1, NULL, 57600, TRUE); // Auto-connect with
the first found USB or PCI TANGO in the system
LSX_ConnectSimple(Tango1, 6, "192.168.1.162", 57600, TRUE); // Connect
to IPv4
LSX_ConnectSimple(Tango1, 1, "COM2", 57600, 11); // Connect to COM2,
protocol window in the dark color scheme
```

ConnectEx

Another way to connect to a TANGO controller.
ConnectEx requires the "TLS_ControlInitPar" parameter structure to connect,
as defined in "Tango.h". This structure must contain the required connection setup.
(other options to connect are: ConnectSimple or LoadConfig+Connect).
Hint: Use parameter ID given from command CreateLSID(), when LSX commands shall be used,
CreateLSID() is not required for the LS commands.
Without connection setup, connection is not possible.

The DLL can also connect to a TANGO Desktop HE via Ethernet (IPv4).
- Therefore, instead of the ComName must contain the TANGO IP-Address xxx.xxx.xxx.xxx instead of COMx and the InterfaceType must be 6 instead of 1.
- In the special case of "Bootloader Connect" (InterfaceType 5", the DLL decides automatically between the interfaces COM or Ethernet.

SYNTAX
```
int LSX_ConnectEx (int                 lLSID,
                   TLS_ControlInitPar *pAControlInitPar);
```

TANGO CMD -

PARAMETERS **pAControlInitPar**: Structure with baud rate, port, protocol etc. information

EXAMPLE
```
LSX_ConnectEx (Tango1, &ControlInitPar);
```

LoadConfig

Load configuration data file (SwitchBoard ".ini" file)
If the file was not found or has invalid content, the function returns error 4001.

SYNTAX
```
int LSX_LoadConfig (int   lLSID,
                    char *pcFileName);
```

TANGO CMD -

PARAMETERS **pcFileName:** File name to be used to read data from.
The ini file data structure should be generated from SwitchBoard (ASCII text).

EXAMPLE
```
REQUIRE(LSX_CreateLSID(&Tango1) == 0);
char* inifile = "C:\\Users\\me\\Desktop\\mytest.ini";
REQUIRE(LSX_LoadConfig(Tango1, inifile) == 0);
REQUIRE(LSX_Connect(Tango1) == 0);
//LSX_SetShowProt(Tango1, TRUE); //overwritten from ini file
REQUIRE(LSX_SetLanguage(Tango1, "germ") == 0);
```

Connect

The 3rd way to connect to a TANGO controller.
(other options to connect are: ConnectSimple or ConnectEx).
Connect using previously loaded configuration data.
The COM Port is taken from the loaded ini file and the ini setup parameters are sent to the TANGO controller after connecting.

SYNTAX
```
int LSX_Connect (int  lLSID);
```

TANGO CMD -

PARAMETERS -

EXAMPLE
```
LSX_CreateLSID(&Tango1);   // create ID for first TANGO
LSX_LoadConfig (Tango1, inifile_name_string);
LSX_Connect (Tango1); // Connect with the ini file information of
LoadConfig
```

SaveConfig

DUMMY Save configuration data to certain file. The function is not implemented yet and returns without writing anything.

SYNTAX
```
int LSX_SaveConfig (int   lLSID,
                    char *pcFileName);
```

TANGO CMD -

PARAMETERS **pcFileName:** File name to write data to. Data is plain ASCII text.

EXAMPLE
```
LSX_SaveConfig (Tango1, pcFileName);
```

Disconnect

Disconnect from TANGO controller.
After calling this function, commands can no longer be sent to the TANGO controller.
This function should be called just before closing the program.
It is only required when using the LS**X**_ functions

SYNTAX
```
int LSX_Disconnect (int  lLSID);
```

TANGO CMD -

PARAMETERS -

EXAMPLE
```
LSX_Disconnect(Tango1); // Disconnect the controller Tango1
LSX_FreeLSID(Tango1);   // And free the LSID (only if LSX_ is used)
```

FreeLSID

Free a TANGO ID-Number that was created by CreateLSID.
FreeLSID should only be called after executing Disconnect.
The LSID is used as an additional parameter in TANGO DLL LSX function calls
to select the TANGO to which command is aimed at from a range of connected Tangos.

SYNTAX

```
int LSX_FreeLSID (int  lLSID);
```

TANGO CMD -

PARAMETERS **LSID**: The given TANGO ID-Number, which is to be set free.
The ID must not be used after FreeLSID has been executed.

EXAMPLE

```
int Tango1;

LSX_CreateLSID(&Tango1);
LSX_ConnectSimple(Tango1, …);
…
LSX_Disconnect(Tango1);
LSX_FreeLSID(Tango1);
```

SetShowProt

Switches the interface protocol window on / off.

SYNTAX

```
int LSX_SetShowProt (int  lLSID,
                     int  lShowProt);
```

TANGO CMD -

PARAMETERS **lShowProt** (int / BOOL):
0 or FALSE = hide;
1 or TRUE = show with timestamp (default);
2 = show with time difference;
3 = show without timestamp.
The dark color scheme uses the same timestamp modes, offset by ten:
11 = dark with timestamp;
12 = dark with time difference;
13 = dark without timestamp.
Any negative value is treated as 1 (show with timestamp), so a caller whose TRUE represents 1 or 0xFFFFFFFF gets the normal light window. That covers Delphi LongBool, Java via JNA and Visual Basic.

EXAMPLE

```
LSX_SetShowProt(Tango1, TRUE);
// Show interface protocol for Tango1, in case not already visible
LSX_SetShowProt(Tango1, 11);
// Switch the protocol window of Tango1 to the dark color scheme
```

ClearProtocolWindow

Clears the content of the protocol window.

SYNTAX
```
int LSX_ClearProtocolWindow (int  lLSID);
```

TANGO CMD -

PARAMETERS **-**

EXAMPLE
```
LSX_ClearProtocolWindow (Tango1); // Delete protocol window list
content of Tango1
```

SetWriteLogText

Switches writing of the interface protocol to a file on and off. The file is named TANGO.log and is created in the working directory of the calling application; if it cannot be created there, the Desktop is used instead. The path actually used is reported once in the protocol window. Writing is switched off by default. Implemented since version 1.541.

SYNTAX
```
int LSX_SetWriteLogText (int  lLSID,
                         BOOL bAWriteLogText);
```

TANGO CMD -

PARAMETERS

bAWriteLogText:
TRUE = write the interface protocol to TANGO.log;
FALSE = stop writing (default).

EXAMPLE
```
LSX_SetWriteLogText(Tango1, TRUE);
// Write the interface protocol of Tango1 to TANGO.log
```

SetWriteLogTextFN

Like SetWriteLogText, but with a file name of your choice. An empty, missing or unusable name falls back to TANGO.log in the working directory and, if that fails as well, to the Desktop. The path actually used is reported once in the protocol window. Writing is switched off by default. Implemented since version 1.541.

SYNTAX
```
int LSX_SetWriteLogTextFN (int   lLSID,
                           BOOL  bAWriteLogText,
                           char *pcALogFN);
```

TANGO CMD -

PARAMETERS

bAWriteLogText:
TRUE = write the interface protocol to the given file;
FALSE = stop writing (default).

pcALogFN: File name, with path if required.
An empty name or NULL uses TANGO.log.

EXAMPLE
```
LSX_SetWriteLogTextFN(Tango1, TRUE, "D:\\Logs\\tango.log");
// Write the interface protocol of Tango1 to a file of your choice
```

SetLanguage

Set language of protocol window.

SYNTAX

```
int LSX_SetLanguage (int   lLSID,
                     char *pcPLN);
```

TANGO CMD -

PARAMETERS

pcPLN: if string contains “germ” or “deut” language is switched to german
if string contains “fren” or “fran” language is switched to french
all other strings switch to english

EXAMPLE

```
LSX_SetLanguage (Tango1, „french“); // Switch Tango1 protocol language to french
```

GetCommandTimeout

Read current DLL timeout for read, move and calibration.

SYNTAX

```
int LSX_GetCommandTimeout (int  lLSID,
                           int *toRead,
                           int *toMove,
                           int *toCal);
```

TANGO CMD -

PARAMETERS

toRead: DLL standard timeout to get a reply from the controller (default 1000 ms)
toMove: DLL timeout for axes moves in [ms]
toCal: DLL timeout for calibration in [ms]

EXAMPLE

```
LSX_GetCommandTimeout(Tango1, &tR, &tM, &tC);
```

SetCommandTimeout

Set DLL timeout for read, move and calibration.

SYNTAX

```
int LSX_SetCommandTimeout (int  lLSID,
                           int  toRead,
                           int  toMove,
                           int  toCal);
```

TANGO CMD -

PARAMETERS

toRead: do not modify DLL standard timeout default 1000 ms
toMove: timeout for move in [ms] (consider speed and acceleration)
toCal: timeout for calibration in [ms] (consider axes length, speed and acceleration)

EXAMPLE

```
LSX_SetCommandTimeout(Tango1, tR, tM, tC);
```

EnableCommandRetry

DUMMY This function is intended to enable/disable repeated sending of commands in case of errors. It currently has no effect; the retry stays enabled.

SYNTAX
```
int LSX_EnableCommandRetry (int   lLSID,
                            BOOL  bAValue);
```

TANGO CMD -

PARAMETERS **bAValue**: TRUE = in case of errors, the TANGO DLL repeats sending certain command (especially in case of WaitForAxisStop)
FALSE = disable repeated sending

EXAMPLE
```
LSX_EnableCommandRetry(Tango1, FALSE);
```

SetDllNumOfAxes

Manipulate the DLL-internal information about the available TANGO axes.
E.g., instructions sent from the DLL to the TANGO will be limited to the corresponding amount of axis parameters.
Avoid changing internal DLL states unless strictly required.

SYNTAX
```
int LSX_SetDllNumOfAxes (int  lLSID,
                         int  lNumOfAxes);
```

TANGO CMD -

PARAMETERS **lNumOfAxes**: 1, 2, 3 or 4

EXAMPLE
```
LSX_SetDllNumOfAxes(Tango1, 3); // Force the DLL to use 3 axes
```

GetSwapZA

Read if the axis Z and A are swapped by the TANGO DLL
(Z parameters redirected to A and vice versa).

SYNTAX
```
int LSX_GetSwapZA (int   lLSID,
                   BOOL *pbValue);
```

TANGO CMD -

PARAMETERS **pbValue**: TRUE = Z and A are swapped,
FALSE = not swapped (default)

EXAMPLE
```
BOOL bSwapped;
LSX_GetSwapZA(Tango1, &bSwapped); // Read the TANGO DLL Z-A swap
setting
```

SetSwapZA

Set if the axis Z and A should be swapped by the TANGO DLL
(Z parameters redirected to A and vice versa) or not.

SYNTAX
```
int LSX_SetSwapZA (int   lLSID,
                   BOOL  bValue);
```

TANGO CMD -

PARAMETERS **bValue**: TRUE = swap Z and A,
FALSE = not swapped (default)

EXAMPLE
```
LSX_SetSwapZA(Tango1, TRUE); // swap Z and A
```

FlushBuffer

Clear the communication input buffer.
Can be used in error situations to remove remaining feedback messages from the input buffer.

SYNTAX
```
int LSX_FlushBuffer (int  lLSID,
                     int  lAValue);
```

TANGO CMD -

PARAMETERS **lAValue**: not used, can be set to 0

EXAMPLE
```
LSX_FlushBuffer(Tango1, 0);
```

GetDLLVersionString

Get DLL version string.

SYNTAX
```
int LSX_GetDLLVersionString (int   lLSID,
                             char *pcVers,
                             int   lMaxLen);
```

TANGO CMD -

PARAMETERS **pcVers** = Buffer, containing return message from DLL
lMaxLen = Limits the max. number of characters to be copied into buffer

EXAMPLE
```
char cVersionString[128];
LSX_GetDLLVersionString(Tango1, cVersionString, 127);
// Example reply: "DLL64 Version 1.403, Oct 12, 2021, 12:34:33"
```

SendString

SendString is the main and most flexible way for communicating with the TANGO.

ASCII commands can be sent and/or received, providing access to all instructions of the TANGO Instruction Set.

Different use cases are possible:
a) just sending an instruction, b) sending and receiving, or c) receiving only

- **For just sending (a)**, *ReadLine* must be FALSE,
  Ret, *MaxLen* and *TimeOut* then are don't care and can be NULL and 0.
- **For send+receive (b)**, *ReadLine* must be TRUE and *Ret*, *MaxLen* and *TimeOut* provided.
  The timeout should be greater than 0, e.g. 100 to avoid sporadic truncation of the received text. TimeOut = 0 might only make sense if it is used for e.g. a Terminal function, where SendString is called in a constant interval to send and read whatever is there or not (without blocking). For normal use, timeout must be set to ensure a complete receive. Usually 100ms will work, maybe large returns require more.
- **Receive only (c)** might make sense in Terminal applications for a constant listener, or in cases where a SendString forced several requests at once and they must be read (because a SendString can send several instructions but can only receive one until the first '\r').

SYNTAX

```
int LSX_SendString (int   lLSID,
                    char *pcStr,
                    char *pcRet,
                    int   lMaxLen,
                    BOOL  bReadLine,
                    int   lTimeOut);
```

TANGO CMD

?ver

PARAMETERS

pcStr Zero-terminated string, which is to be sent to controller.
The zero-terminated string must end with a carriage return (\r).
SendString never appends one by itself. That is intentional: it allows a command to be composed from several calls, or a long text to be sent in parts. Without the final \r the controller keeps waiting for the rest of the line, so a request returns 4007 and an empty reply text.

pcRet Buffer, for receiving the return message from the TANGO,
if ReadLine = TRUE
Or zero (NULL), in case ReadLine = FALSE

lMaxLen Max. number of characters allowed to be copied into buffer
usually the [buffer length -1] or anything less

bReadLine TRUE = read return message from TANGO
FALSE = don't wait for return message

lTimeOut Max. waiting period for return message [ms]
A timeout of 0 would only read what is already in the input buffer, and incoming characters of a request might be not included. A good timeout value might be 100ms or more, depending on the requested data.

EXAMPLE

```
LSX_SendString(Tango1, "?version\r", pcVer, 256, TRUE, 250);
// Read version number, allow max. 256 characters, 1/4 Second Timeout
LSX_SendString(Tango1, "!speed 10.5\r", NULL, 0, FALSE, 0);
// just send, start speed move for X-axis
LSX_SendString(Tango1, NULL, pcVer, 256, TRUE, 250);
// just read, wait 250ms if a reply arrives
```

SendStringPosCmd

Send a move command to the TANGO as a string, and wait for return message.

The parameters allow options of wait / do not wait, return the reply or not required (the reply string might be of interest in case of @, E, T, A, D, S ... was returned by the move)

With ReadLine TRUE, the function always waits until a reply was sent from the TANGO.
And if a return buffer is specified, the reply is copied into this buffer, else not.

If autostatus is set to 0, no reply will be sent by the TANGO, and the wait will timeout.

SYNTAX

```
int LSX_SendStringPosCmd (int   lLSID,
                          char *pcStr,
                          char *pcRet,
                          int   lMaxLen,
                          BOOL  bReadLine,
                          int   lTimeOut);
```

TANGO CMD -

PARAMETERS

pcStr = Zero-terminated ASCII string, which is to be sent to the controller
The last character must be an "\r" (the CR character)

pcRet = Buffer, containing return message from TANGO, in case ReadLine = TRUE
or ZERO (NULL), in case the autostatus "@@@" reply string is not required

lMaxLen = Max. amount of characters allowed copied into pcRet buffer

bReadLine = TRUE = wait for return message (@@@ reply) from TANGO
= FALSE = don't wait for return message / move complete

lTimeOut = Max. waiting period for return message [ms]

EXAMPLE

```
char reply_text[16];
LSX_SetAutoStatus(Tango1, 1); // Ensure the TANGO will reply on the
move (@@@)
LSX_SendStringPosCmd((Tango1, "!moa 1.5\r", &reply_text[0], 16, TRUE,
10000);
LSX_SendStringPosCmd(Tango1, "!mor z -0.05\r", NULL, 0, TRUE, 2000);
LSX_SendStringPosCmd(Tango1, "!cal x\r", &reply_text[0], 16, TRUE,
60000);
```

SetAbortFlag

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.

SYNTAX

```
int LSX_SetAbortFlag (int  lLSID);
```

TANGO CMD

-

PARAMETERS

-

EXAMPLE

```
LSX_SetAbortFlag(Tango1);
LSX_StopAxes(Tango1);
// closes communication with TANGO and sends stop command for all axes
```

4.3. Controller Information

GetSerialNr

Reads out the TANGO serial number.

SYNTAX

```
int LSX_GetSerialNr (int   lLSID,
                     char *pcSerialNr,
                     int   lMaxLen);
```

TANGO CMD ?readsn

PARAMETERS **pcSerialNr**: Pointer to a buffer, in which the serial number will be returned
lMaxLen: Limits the max. number of characters to be copied into buffer

EXAMPLE

```
char TangoSN[16]; // The SN consists of 9 ASCII characters (0-9, A-Z)
LSX_GetSerialNr(Tango1, TangoSN, 15); // Example reply: 190103001
// 190103001 =  19 = YY, 01 = WW, 0 = Controller Type, 3 = 3Axes max.,
001 Index
```

GetTangoVersion

Get TANGO version string.
Read the TANGO Controller Version information as ASCII text.

SYNTAX

```
int LSX_GetTangoVersion (int   lLSID,
                         char *pcVers,
                         int   lMaxLen);
```

TANGO CMD ?version

PARAMETERS **pcVers:** Pointer to the char buffer, in which the TANGO version text is returned
lMaxLen: Limits the max. number of characters to be copied into buffer

EXAMPLE

```
char cVersionString[128];
LSX_GetTangoVersion (Tango1, cVersionString, 127);
// Example reply: “TANGO-Desktop, Version 1.73, Mar 11 2021, 13:28:26”
```

GetVersionStr

Returns the backward-compatible firmware, axis and motorcurrent information.

SYNTAX

```
int LSX_GetVersionStr (int   lLSID,
                       char *pcVers,
                       int   lMaxLen);
```

TANGO CMD ?ver

PARAMETERS **pcVers**: Pointer to the char buffer, in which the TANGO version text is returned
lMaxLen: Limits the max. number of characters to be copied into buffer

EXAMPLE

```
LSX_GetVersionStr(Tango1, pcVers, 64); // retrieve compatible version
information
```

GetVersionStrDet

Retrieves detailed configuration of TANGO as decimal ASCII digits.

SYNTAX

```
int LSX_GetVersionStrDet (int   lLSID,
                          char *pcVersDet,
                          int   lMaxLen);
```

TANGO CMD ?det

PARAMETERS

pcVersDet: Pointer to a buffer, in which the string will be returned
lMaxLen: Limits the max. number of characters to be copied into buffer

EXAMPLE

```
LSX_GetVersionStrDet(Tango1, pcVersDet, 16); // retrieve detailed configuration
```

GetVersionStrInfo

Retrieves optional internal information on the controller version.

SYNTAX

```
int LSX_GetVersionStrInfo (int   lLSID,
                           char *pcVersInfo,
                           int   lMaxLen);
```

TANGO CMD ?iver

PARAMETERS

pcVersInfo: Pointer to a buffer, in which the string will be returned
lMaxLen: Limits the max. number of characters to be copied into buffer

EXAMPLE

```
LSX_GetVersionStrInfo(Tango1, pcVersInfo, 16);
```

GetStageSN

Retrieves optional internal information on the stage serial number.

SYNTAX

```
int LSX_GetStageSN (int   lLSID,
                    char *pcSN,
                    int   lMaxLen);
```

TANGO CMD ?stagesn

PARAMETERS

pcSN: Pointer to a buffer, in which the string will be returned
lMaxLen: Limits the max. number of characters to be copied into buffer

EXAMPLE

```
LSX_GetStageSN(Tango1, pcSN, 16);
```

4.4. Status Requests

GetError

Readout the current error state of the controller.

SYNTAX
```
int LSX_GetError (int  lLSID,
                  int *plErrorCode);
```

TANGO CMD ?err

PARAMETERS **plErrorCode**: Error number (as described in chapter 9.1)

EXAMPLE
```
LSX_GetError(Tango1, &ErrorCode);
```

GetErrorString

Retrieves ASCII text explanation of the here specified error number.
TANGO errors 0...255 can be requested as well as DLL error codes >= 4001.

SYNTAX
```
int LSX_GetErrorString (int   lLSID,
                        int   lError,
                        char *pcErrorString,
                        int   lMaxLen);
```

TANGO CMD ?help

PARAMETERS **lError**: Error number of which the explanation shall be returned 0..255, 4001...
pcErrorString: Character array pointer to receive the text
lMaxLen: Limits the max. number of characters to be copied into the array

EXAMPLE
```
LSX_GetErrorString(Tango1, 29, &text_array[0], 64); // read
explanation of error 29
```

GetPos

Retrieves current position of all axes. Also refer to SetEncoderPosition.

SYNTAX
```
int LSX_GetPos (int    lLSID,
                double *pdX,
                double *pdY,
                double *pdZ,
                double *pdA);
```

TANGO CMD ?pos

PARAMETERS **pdX...pdA**: Axis position values

EXAMPLE
```
LSX_GetPos(Tango1, &X, &Y, &Z, &A);
```

GetPosEx

Retrieves encoder or motor positions of all axes.
If an axis is not available, 0.0 is returned.

SYNTAX

```
int LSX_GetPosEx (int     lLSID,
                  double *pdX,
                  double *pdY,
                  double *pdZ,
                  double *pdA,
                  BOOL    bEncoder);
```

TANGO CMD !encpos, ?pos

PARAMETERS

pdX...pdA: Position parameter
bEncoder TRUE = Provide encoder positions *
FALSE = Provide motor position values
* when no encoder available, TRUE returns the motor pos.

EXAMPLE

```
LSX_GetPosEx(Tango1, &X, &Y, &Z, &A, TRUE);
```

GetPosSingleAxis

Retrieves current position of a single axis.
If the motor or encoder position is returned depends on SetEncoderPosition.
If the axis is not available, 0.0 is returned.

SYNTAX

```
int LSX_GetPosSingleAxis (int     lLSID,
                          int     lAxis,
                          double *pdPos);
```

TANGO CMD ?pos x,y,z,a

PARAMETERS

lAxis: Axis of which the position parameters shall be retrieved from, 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
pdPos: Positions

EXAMPLE

```
LSX_GetPosSingleAxis(Tango1, 2, &Pos); // retrieves position of Y-Axis
```

SetPos

Set position.

SYNTAX

```
int LSX_SetPos (int     lLSID,
                double  dX,
                double  dY,
                double  dZ,
                double  dA);
```

TANGO CMD !pos

PARAMETERS

dX...dA: Position within min./max. travel range, depends on dimension

EXAMPLE

```
LSX_SetPos(Tango1, 10, 10, 0, 0); // Set current position to this
values
```

SetPosSingleAxis new in 1.540

Sets the current position of a single axis. The axis is not moved, only its position value is redefined. Single-axis variant of SetPos.

SYNTAX

```
int LSX_SetPosSingleAxis (int     lLSID,
                          int     lAxis,
                          double  dPos);
```

TANGO CMD !pos x,y,z,a

PARAMETERS

lAxis: 1=X, 2=Y, 3=Z, 4=A

dPos: position within min./max. travel range, depends on dimension

EXAMPLE

```
LSX_SetPosSingleAxis(Tango1, 2, 10.0);
```

ClearPos

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. In such case, !zero (x,y,z,a) must be sent.

SYNTAX

```
int LSX_ClearPos (int  lLSID,
                  int  lFlags);
```

TANGO CMD !clearpos

PARAMETERS

lFlags: 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 for Y-Axis.

GetStatus

Retrieves current status of the controller.

SYNTAX

```
int LSX_GetStatus (int   lLSID,
                   char *pcStat,
                   int   lMaxLen);
```

TANGO CMD ?status

PARAMETERS

pcStat: Pointer to a buffer, in which the status string will be returned

lMaxLen: Limits the max. number of characters to be copied into buffer

EXAMPLE

```
LSX_GetStatus(Tango1, &Stat, 16);
```

GetStatusAxis

Retrieves current status of the axes.
Only the character M indicates a moving axis. At all other characters, the axis is stopped/idle.

SYNTAX

```
int LSX_GetStatusAxis (int   lLSID,
                       char *pcStatusAxisStr,
                       int   lMaxLen);
```

TANGO CMD ?statusaxis

PARAMETERS

pcStatusAxisStr: Pointer to a buffer in which status string will be returned

lMaxLen: Limits the max. number of characters 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

```
LSX_GetStatusAxis(Tango1, &StatusAxisStr, 16);
```

GetStatusLimit

Retrieves current status of software limits of each axis.

SYNTAX

```
int LSX_GetStatusLimit (int   lLSID,
                        char *pcLimit,
                        int   lMaxLen);
```

TANGO CMD ?statuslimit

PARAMETERS

pcLimit: Pointer to a buffer, in which the status of the axes will be returned

e.g.: AAA-DD-- LLLL

A = Axis has been calibrated (cal)

D = Stage travel range has been measured (rm)

L = Software limit has been set (lim)

- = Limit not yet applied

lMaxLen: The max. number of characters that can be copied into the buffer

EXAMPLE

```
LSX_GetStatusLimit(Tango1, &Limit, 32);
```

GetStA

Read the detailed axis states (sta).
Returns all states in which the axis can be as a set of 32 individual flags.
For more details, refer to the sta description of the TANGO Instruction Set and the corresponding TANGO firmware version.

```
00000001 !axis is set to 0 or 1 (motor current is on)
00000002 !axis is set to 1      (enabled)
00000004 motor power amplifier is on
00000008 motor power amplifier error

00000010 corresponds to statusaxis 'M' state of the axis
00000020 the axis travels due to HDI deflection (e.g.joystick)
00000040 cal is running
00000080 rm  is running

00000100 cal already executed ('A' in statuslimit)
00000200 rm  already executed ('D' in statuslimit)
00000400 lower limit switch E0 actuated (1 in readsw)
00000800 upper limit switch EE actuated (1 in readsw)

00001000 axis move waits for snapshot signal
00002000 calrequired prevents axis move, no cal/rm yet
00004000 1D position correction active
00008000 2D position correction active (@ Z reply: 2D+z)

00010000 encoder is active
00020000 encoder was activated, even if currently disabled
00040000 reserved
00080000 encoder error (encerr)

00100000 closed loop is on
00200000 closed loop is active, regulating
00400000 closed loop is in target window (set by twi)
00800000 closed loop is in lock-in range (set by ctrs)

01000000 stop signal is active
02000000 HDI is enabled for this axis (joy+joydir)
04000000 Thermal compensation temperature is updated, no error
08000000 Thermal compensation is applied, was activated by cal

10000000 Macro execution (macro is running)     2nd gen. TANGOs
20000000 Cal Learn data for encoder (callrnpos) Firmw. ≥ 1.782
40000000 Cal Learn data for motor   (callrnposmot) Fw. ≥ 1.782
80000000 reserved

Example: sta x returns a hex number for the x axis state flags:
02030307  --> axis is not traveling, no closed loop, no errors
 | | | |
 | | | +- axis is on
 | | +--- cal and rm are executed
 | +----- encoder is active (and was/is active)
 +------- HDI is enabled (joy+joydir)
```

SYNTAX

```
int LSX_GetStA (int  lLSID,
                int *plStaX,
                int *plStaY,
                int *plStaZ,
                int *plStaA);
```

TANGO CMD sta

PARAMETERS **plStaX...plStaA**: Pointers to integer, in which the state flags of the axes are returned

EXAMPLE

```
LSX_GetStA(Tango1, &lStaX, &lStaY, &lStaZ, &lStaA);
```

SetAutoStatus

Sets the Auto-Status mode.

Please note:
AutoStatus mode should not be changed, as TANGO DLL sets correct mode for travel commands etc. (except the SendStringPosCmd, where AutoStatus 1 is required, if the wait for reply option is used).

SYNTAX

```
int LSX_SetAutoStatus (int  lLSID,
                       int  lValue);
```

TANGO CMD !autostatus

PARAMETER ***Value***: AutoStatus mode:
0 = Controller sends no status
1 = Controller automatically sends „Position reached" messages (the @@@, default)
2 = Controller automatically sends „Position reached" and status messages
This mode might cause false behavior of the TANGO DLL.
3 = There is only one carriage return sent for „Position reached"

EXAMPLE

```
LSX_SetAutoStatus(Tango1, 1);
```

GetAutoStatus

Reads the current Auto-Status mode.

SYNTAX

```
int LSX_GetAutoStatus (int  lLSID,
                       int *plStatus);
```

TANGO CMD ?autostatus

PARAMETER **plStatus:** Status: Pointer to integer, in which the current setting of Auto-Status will be returned
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

```
LSX_GetAutoStatus(Tango1, &autostatus);
```

IsVel

Read the actual velocities at which the axes are currently travelling.
Unlike '**?vel**' or '**?speed**' this instruction returns the currently travelled (true) speed of the axes, even when controlled by an HDI device.

SYNTAX

```
int LSX_IsVel (int     lLSID,
               double *pdX,
               double *pdY,
               double *pdZ,
               double *pdA);
```

TANGO CMD isvel

PARAMETERS **pdX...pdA**: actual axes velocities in [mm/s]

EXAMPLE

```
double vx, vy, vz, va;
LSX_IsVel(Tango1, &vx, &vy, &vz, &va);
```

IsVelSingleAxis

Read the actual velocity at which an axis is currently travelling.
Unlike '**?vel**' or '**?speed**' this instruction returns the currently travelled (true) speed of the axes, even when controlled by an HDI device.

SYNTAX

```
int LSX_IsVelSingleAxis (int     lLSID,
                         int     lAxis,
                         double *pdVel);
```

TANGO CMD isvel x,y,z,a

PARAMETERS

lAxis: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
pdVel: actual axis velocity in [mm/s]

EXAMPLE

```
LSX_IsVelSingleAxis(Tango1, 2, &vel); // returns actual velocity of y
axis
```

4.5. Settings

GetPowerAmplifier

Retrieves the amplifier states, on or off.

SYNTAX

```
int LSX_GetPowerAmplifier (int   lLSID,
                           BOOL *pbAmplifier);
```

TANGO CMD ?pa

PARAMETERS **pbAmplifier**: TRUE = Amplifiers are switched on
FALSE = Amplifiers are switched off

EXAMPLE

```
LSX_GetPowerAmplifier(Tango1, &Amplifier);
```

SetPowerAmplifier

Switch amplifier on / off.

SYNTAX

```
int LSX_SetPowerAmplifier (int   lLSID,
                           BOOL  bAmplifier);
```

TANGO CMD !pa

PARAMETERS **bAmplifier**: TRUE = Switch amplifiers on
FALSE = Switch amplifiers off

EXAMPLE

```
LSX_SetPowerAmplifier(Tango1, TRUE); // switches amplifiers on
```

GetActiveAxes

Retrieves the axis enable states.

SYNTAX

```
int LSX_GetActiveAxes (int  lLSID,
                       int *plFlags);
```

TANGO CMD ?axis

PARAMETERS **plFlags**: 32-Bit Integer. After calling this function the axis bitmask is returned in Bits 0-4
Bit 0 = X, Bit 1 = Y, Bit 2 = Z, Bit 3 = A / 1=axis is on, 0 = axis off or disabled

EXAMPLE

```
LSX_GetActiveAxes(Tango1, &Flags);
if (Flags & 0x01) …;            // Flags  Bit 0 = 1 = X-Axis is on
if ((Flags & 0x04) == 0) …;     // Flags  Bit 2 = 0 =  Z-Axis is off
or disabled
```

SetActiveAxes

Enable or disable axes.

SYNTAX

```
int LSX_SetActiveAxes (int  lLSID,
                       int  lFlags);
```

TANGO CMD !axis

PARAMETERS **lFlags**: Bit mask, bits 0 to 3 represent axes X (0x01) to A (0x08)
Bit 0 = 1 = X-Axis enabled, Bit 1 = 2= Y-Axis enabled
Bit 2 = 4 = Z-Axis enabled, Bit 3 = 8 = A-Axis enabled

EXAMPLE

```
LSX_SetActiveAxes(Tango1, 3);
// X- and Y-Axes are enabled (Bits 0 and 1 set),
// Z-Axis and A-Axis switched off (Bit 2 = 0, Bit 3 = 0)
```

GetAxisDirection

Retrieves axis directions.

SYNTAX

```
int LSX_GetAxisDirection (int  lLSID,
                          int *plXD,
                          int *plYD,
                          int *plZD,
                          int *plAD);
```

TANGO CMD ?axisdir

PARAMETERS **plXD...plAD**: 32-Bit Integers
0 = normal turning direction
1 = reversed turning direction

EXAMPLE

```
LSX_GetAxisDirection(Tango1, &XD, &YD,&ZD,&AD);
```

SetAxisDirection

Set axis directions.

SYNTAX

```
int LSX_SetAxisDirection (int  lLSID,
                          int  lXD,
                          int  lYD,
                          int  lZD,
                          int  lAD);
```

TANGO CMD !axisdir

PARAMETERS **lXD...lAD**: 32-Bit Integers
0 = normal motor turning direction
1 = reverse reversed motor turning direction

EXAMPLE

```
LSX_SetAxisDirection(Tango1, 1, 0, 0, 0);
// Set direction of X-Axis to reversed (1), other axes not reversed
```

GetCalibOffset

Retrieves zero position offset of axes.

SYNTAX
```
int LSX_GetCalibOffset (int     lLSID,
                        double *pdX,
                        double *pdY,
                        double *pdZ,
                        double *pdA);
```

TANGO CMD ?caliboffset

PARAMETERS **pdX...pdA**: zero position offset from cal switch, depends on dimension

EXAMPLE
```
LSX_GetCalibOffset(Tango1, &X, &Y, &Z, &A);
```

SetCalibOffset

Sets zero position offset of axes.
The axis zero position is moved from the hardware cal limit switch by this amount.

SYNTAX
```
int LSX_SetCalibOffset (int     lLSID,
                        double  dX,
                        double  dY,
                        double  dZ,
                        double  dA);
```

TANGO CMD !caliboffset

PARAMETERS **dX...dA**: typically 0-5 [mm]

EXAMPLE
```
LSX_SetCalibOffset(Tango1, 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)
```

GetRMOffset

Retrieves axis position offsets to RM limit switch.

SYNTAX
```
int LSX_GetRMOffset (int     lLSID,
                     double *pdX,
                     double *pdY,
                     double *pdZ,
                     double *pdA);
```

TANGO CMD ?rmoffset

PARAMETERS **pdX...pdA**: Limit switch position offset, depending on measuring unit (dimension).

EXAMPLE
```
LSX_GetRMOffset(Tango1, &X, &Y, &Z, &A);
```

SetRMOffset

Sets RM position offset of axes.
The axis stops this amount before the hardware RM endswitch.

SYNTAX

```
int LSX_SetRMOffset (int     lLSID,
                     double  dX,
                     double  dY,
                     double  dZ,
                     double  dA);
```

TANGO CMD !rmoffset

PARAMETERS **dX...dA**: typically 0-5 [mm]

EXAMPLE

```
LSX_SetRMOffset(Tango1, 1, 1, 1, 1);
// limit positions of axes are each moved for 1mm (at dimension 2 2 2 2)
towards stage center
```

GetCalibBackSpeed

Retrieves revolving speed at which axes are driven from limit switches when calibrating.
Speed is equivalent to issued value * 0.01 rev/sec.

SYNTAX

```
int LSX_GetCalibBackSpeed (int  lLSID,
                           int *plSpeed);
```

TANGO CMD ?calbspeed

PARAMETERS **plSpeed**: Speed value in 1/100 revolutions/second

EXAMPLE

```
LSX_GetCalibBackSpeed(Tango1, &lSpeed);
```

SetCalibBackSpeed

Sets revolving speed at which axes are driven from limit switches when calibrating.
Speed is equivalent to issued value * 0.01 rev/sec.

SYNTAX

```
int LSX_SetCalibBackSpeed (int  lLSID,
                           int  lSpeed);
```

TANGO CMD !calbspeed

PARAMETERS **lSpeed**: Speed value in 1/100 revolutions/second (within parameters of 1 to 100)

EXAMPLE

```
LSX_SetCalibBackSpeed(Tango1, 10);
// when calibrating, limit switches are left at 0.1 rev/sec
```

GetCalibrateDir

Retrieves calibrating direction.

SYNTAX

```
int LSX_GetCalibrateDir (int  lLSID,
                         int *plXD,
                         int *plYD,
                         int *plZD,
                         int *plAD);
```

TANGO CMD ?caldir

PARAMETERS **plXD...plAD**: 32-Bit Integer
0 = normal calibration direction
1 = (reserved, don't use!)
2, ... reserved for center reference modes, refer to TANGO Instruction Set

EXAMPLE

```
LSX_GetCalibrateDir(Tango1, &XD, &YD,&ZD,&AD);
```

SetCalibrateDir

Set calibrating direction.

SYNTAX

```
int LSX_SetCalibrateDir (int  lLSID,
                         int  lXD,
                         int  lYD,
                         int  lZD,
                         int  lAD);
```

TANGO CMD !caldir

PARAMETERS **lXD...lAD**: 32-Bit Integer
0 = normal calibration direction
1 = (reserved, don't use!)
2, ... reserved for center reference modes, refer to TANGO Instruction Set

EXAMPLE

```
LSX_(Tango1, 0, 0, 0, 0); // Set all axes to caldir = 0
```

GetDimensions

Retrieves the applied measuring units of axes

SYNTAX

```
int LSX_GetDimensions (int  lLSID,
                       int *plXD,
                       int *plYD,
                       int *plZD,
                       int *plAD);
```

TANGO CMD ?dim

PARAMETERS **plXD...plAD**: Dimension units

0 = Microsteps
1 = µm
2 = mm
3 = Degree
4 = Revolutions
5 = cm
6 = m
7 = Inch
8 = mil (1/1000 Inch)
9 = position in mm and speed in mm/s (also the preferred setting)
10 = position in µm and speed in mm/s (behaves as dim 1 at a 1mm pitch)

EXAMPLE

```
LSX_GetDimensions(Tango1, &XD, &YD,&ZD,&AD);
```

SetDimensions

Set measuring units of axes.

SYNTAX

```
int LSX_SetDimensions (int  lLSID,
                       int  lXD,
                       int  lYD,
                       int  lZD,
                       int  lAD);
```

TANGO CMD !dim

PARAMETERS **lXD...lAD**: 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)
9 = position in mm and speed in mm/s
10 = position in µm and speed in mm/s (behaves as dim 1 at a 1mm pitch)

EXAMPLE

```
LSX_SetDimensions(Tango1, 3, 2, 2, 1); // X = degree, Y+Z = mm, A = µm
```

GetDimensionsSingleAxis new in 1.533

Read measuring unit (dimension) of a single axis. Single-axis variant of GetDimensions.

SYNTAX
```
int LSX_GetDimensionsSingleAxis (int  lLSID,
                                 int  lAxis,
                                 int *plDimension);
```

TANGO CMD ?dim x,y,z,a

PARAMETERS
lAxis: 1=X, 2=Y, 3=Z, 4=A
plDimension: measuring unit code

EXAMPLE
```
LSX_GetDimensionsSingleAxis(Tango1, 1, &lDimension);
```

SetDimensionsSingleAxis new in 1.533

Set measuring unit (dimension) of a single axis. Single-axis variant of SetDimensions.

SYNTAX
```
int LSX_SetDimensionsSingleAxis (int  lLSID,
                                 int  lAxis,
                                 int  lDimension);
```

TANGO CMD !dim x,y,z,a

PARAMETERS
lAxis: 1=X, 2=Y, 3=Z, 4=A
lDimension: measuring unit code

EXAMPLE
```
LSX_SetDimensionsSingleAxis(Tango1, 1, 2);
```

GetResolution

Retrieves the applied number of “digits after the millimetre”.
This command is used for dimensions 1, 2, 9 or 10 (mm and µm).
The TANGO default is “4 digits after the millimetre” (0.1µm resolution).
The TANGO transmits (replies) Position values with this resolution to the DLL.

SYNTAX
```
int LSX_GetResolution (int  lLSID,
                       int *plValue);
```

TANGO CMD ?resolution

PARAMETERS
plValue: Resolution units
3 = 1 µm resolution
4 = 0.1 µm resolution (Default)
5 = 10 nm resolution
6 = 1 nm resolution

EXAMPLE
```
LSX_GetResolution(Tango1, &resolution);
```

SetResolution

Sets the applied number of “digits after the millimetre”.
This command is used for dimensions 1, 2, 9 or 10 (mm and µm).
The TANGO default is “4 digits after the millimetre” (1/10µm resolution).
The TANGO transmits (replies) Position values with this resolution to the DLL.
Specify 5 (=10nm) or 6 (=1nm) digits to get higher position resolution from TANGO.

SYNTAX

```
int LSX_SetResolution (int  lLSID,
                       int  lValue);
```

TANGO CMD !resolution

PARAMETERS **lValue**: Resolution units
3 = 1 µm resolution
4 = 0.1 µm resolution (Default)
5 = 10 nm resolution
6 = 1 nm resolution

EXAMPLE

```
LSX_SetResolution(Tango1, 5); // set 5 digits after the decimal point for all axes
```

GetPitch

Retrieves lead screw pitch.

SYNTAX

```
int LSX_GetPitch (int    lLSID,
                  double *pdX,
                  double *pdY,
                  double *pdZ,
                  double *pdA);
```

TANGO CMD ?pitch

PARAMETERS **pdX...pdA**: Lead screw pitch [mm]

EXAMPLE

```
LSX_GetPitch(Tango1, &X, &Y, &Z, &A);
```

SetPitch

Sets the lead screw pitch.
If a Märzhäuser stage or axis is used, the pitch of the axes is usually pre-defined and does not need to be set.

SYNTAX

```
int LSX_SetPitch (int     lLSID,
                  double  dX,
                  double  dY,
                  double  dZ,
                  double  dA);
```

TANGO CMD !pitch

PARAMETERS **dX...dA**: 0.0001 to 100 [mm per motor revolution]

EXAMPLE

```
LSX_SetPitch(Tango1, 4, 4, 1, 2); // Set pitch of X+Y to 4mm, Z to 1mm, A to 2mm
```

GetPitchSingleAxis new in 1.533

Reads the lead screw pitch of a single axis. Single-axis variant of GetPitch. Read resolution is firmware-dependent (neo = double64, up to 15 / old = float32, up to 7 significant digits).

SYNTAX

```
int LSX_GetPitchSingleAxis (int     lLSID,
                            int     lAxis,
                            double *pdPitch);
```

TANGO CMD ?pitch x,y,z,a

PARAMETERS **lAxis:** 1=X, 2=Y, 3=Z, 4=A
pdPitch: lead screw pitch [mm]

EXAMPLE

```
LSX_GetPitchSingleAxis(Tango1, 1, &dPitch);
```

SetPitchSingleAxis new in 1.533

Sets the lead screw pitch of a single axis. Single-axis variant of SetPitch.

SYNTAX

```
int LSX_SetPitchSingleAxis (int     lLSID,
                            int     lAxis,
                            double  dPitch);
```

TANGO CMD !pitch x,y,z,a

PARAMETERS **lAxis:** 1=X, 2=Y, 3=Z, 4=A
dPitch: lead screw pitch [mm]

EXAMPLE

```
LSX_SetPitchSingleAxis(Tango1, 1, 4.0);
```

GetGear

Retrieves the gear ratio.

SYNTAX

```
int LSX_GetGear (int     lLSID,
                 double *pdX,
                 double *pdY,
                 double *pdZ,
                 double *pdA);
```

TANGO CMD ?gear

PARAMETERS **pdX...pdA**: Gear ratio values

EXAMPLE

```
LSX_GetGear(Tango1, &X, &Y, &Z, &A);
```

SetGear

Sets the gear ratio.
If a Märzhäuser stage or axis is used, the gear of the axes X+Y is usually pre-defined and does not need to be set. The default for most applications is 1.

SYNTAX

```
int LSX_SetGear (int     lLSID,
                 double  dX,
                 double  dY,
                 double  dZ,
                 double  dA);
```

TANGO CMD !gear

PARAMETERS **dX...dA:** 0.01 – 1000, default = 1

EXAMPLE

```
LSX_SetGear(Tango1, 4.0, 2.0, 1.0, 1.0);
// programs gear ratios ¼ for Z, ½ for Y and 1/1 for Z and A
```

GetGearSingleAxis new in 1.533

Reads the gear ratio of a single axis. Single-axis variant of GetGear. Read resolution is firmware-dependent (neo = double64, up to 15 / old = float32, up to 7 significant digits).

SYNTAX

```
int LSX_GetGearSingleAxis (int     lLSID,
                           int     lAxis,
                           double *pdGear);
```

TANGO CMD ?gear x,y,z,a

PARAMETERS **lAxis:** 1=X, 2=Y, 3=Z, 4=A
pdGear: gear ratio

EXAMPLE

```
LSX_GetGearSingleAxis(Tango1, 1, &dGear);
```

SetGearSingleAxis new in 1.533

Sets the gear ratio of a single axis. Single-axis variant of SetGear.

SYNTAX

```
int LSX_SetGearSingleAxis (int     lLSID,
                           int     lAxis,
                           double  dGear);
```

TANGO CMD !gear x,y,z,a

PARAMETERS **lAxis:** 1=X, 2=Y, 3=Z, 4=A
dGear: gear ratio

EXAMPLE

```
LSX_SetGearSingleAxis(Tango1, 1, 1.0);
```

GetMotorSteps

Retrieves the number of motor steps.

SYNTAX

```
int LSX_GetMotorSteps (int  lLSID,
                       int *lX,
                       int *lY,
                       int *lZ,
                       int *lA);
```

TANGO CMD ?motorsteps

PARAMETERS **lX...lA**: Number of motor steps

EXAMPLE

```
LSX_GetMotorSteps(Tango1, &X, &Y, &Z, &A);
```

SetMotorSteps

Sets the number of motor steps. Default 200 for 1.8° (50 pole pairs) stepper motors.

SYNTAX

```
int LSX_SetMotorSteps (int  lLSID,
                       int  lX,
                       int  lY,
                       int  lZ,
                       int  lA);
```

TANGO CMD !motorsteps

PARAMETERS **lX...lA**: Motor steps X, Y, Z and A-Axis

EXAMPLE

```
LSX_SetMotorSteps(Tango1, 200, 200, 400, 24);
// set X, Y to default 200, Z to 400 and A to 24 steps motor type
(1.8°, 0.9° and 15° mot.)
```

GetMotorStepsSingleAxis new in 1.533

Reads the number of motor steps of a single axis. Single-axis variant of GetMotorSteps.

SYNTAX
```
int LSX_GetMotorStepsSingleAxis (int  lLSID,
                                 int  lAxis,
                                 int *plMotorSteps);
```

TANGO CMD ?motorsteps x,y,z,a

PARAMETERS
lAxis: 1=X, 2=Y, 3=Z, 4=A
plMotorSteps: motor steps per revolution

EXAMPLE
```
LSX_GetMotorStepsSingleAxis(Tango1, 1, &lMotorSteps);
```

SetMotorStepsSingleAxis new in 1.533

Sets the number of motor steps of a single axis. Single-axis variant of SetMotorSteps.

SYNTAX
```
int LSX_SetMotorStepsSingleAxis (int  lLSID,
                                 int  lAxis,
                                 int  lMotorSteps);
```

TANGO CMD !motorsteps x,y,z,a

PARAMETERS
lAxis: 1=X, 2=Y, 3=Z, 4=A
lMotorSteps: motor steps per revolution

EXAMPLE
```
LSX_SetMotorStepsSingleAxis(Tango1, 1, 200);
```

GetMotorCurrent

Reads the motor current that will be forced to the motor.

SYNTAX
```
int LSX_GetMotorCurrent (int     lLSID,
                         double *pdX,
                         double *pdY,
                         double *pdZ,
                         double *pdA);
```

TANGO CMD ?cur

PARAMETERS
pdX...pdA: Electrical motor currents in [A]

EXAMPLE
```
LSX_GetMotorCurrent(Tango1, &X, &Y, &Z, &A);
```

SetMotorCurrent

Sets the current of motor.

SYNTAX

```
int LSX_SetMotorCurrent (int     lLSID,
                         double  dX,
                         double  dY,
                         double  dZ,
                         double  dA);
```

TANGO CMD !cur

PARAMETERS **dX...dA**: Motor current X, Y, Z and A-Axis in [A]

EXAMPLE

```
LSX_SetMotorCurrent(Tango1, 1.0, 1.0, 0.8, 0.8);
// motor current X- and Y-Axis 1 Ampere; Z- and A-Axis 0.8 Ampere
```

GetMotorCurrentSingleAxis new in 1.533

Reads the motor current of a single axis. Single-axis variant of GetMotorCurrent.

SYNTAX

```
int LSX_GetMotorCurrentSingleAxis (int     lLSID,
                                   int     lAxis,
                                   double *pdCurrent);
```

TANGO CMD ?cur x,y,z,a

PARAMETERS **lAxis:** 1=X, 2=Y, 3=Z, 4=A
pdCurrent: motor current [A]

EXAMPLE

```
LSX_GetMotorCurrentSingleAxis(Tango1, 1, &dCurrent);
```

SetMotorCurrentSingleAxis new in 1.533

Sets the motor current of a single axis. Single-axis variant of SetMotorCurrent.

SYNTAX

```
int LSX_SetMotorCurrentSingleAxis (int     lLSID,
                                   int     lAxis,
                                   double  dCurrent);
```

TANGO CMD !cur x,y,z,a

PARAMETERS **lAxis:** 1=X, 2=Y, 3=Z, 4=A
dCurrent: motor current [A]

EXAMPLE

```
LSX_SetMotorCurrentSingleAxis(Tango1, 1, 1.0);
```

GetReduction

Reads the motor current reduction factor for idle axes.

Info: A reduction of 0.7 (to 70%) will reduce the power and produced heat by 50% and a reduction of 0.5 (50%) will reduce the power and heat of idle axes to about 25%.

SYNTAX

```
int LSX_GetReduction (int     lLSID,
                      double *pdX,
                      double *pdY,
                      double *pdZ,
                      double *pdA);
```

TANGO CMD ?reduction

PARAMETERS **pdX...pdA**: Electrical motor current reduction 0.00 ... 1.00 (= 0...100%)

EXAMPLE

```
LSX_GetReduction(Tango1, &X, &Y, &Z, &A);
```

SetReduction

Sets the reduction factor of the motor current in idle state.

Info: A reduction of 0.7 (to 70%) will reduce the power and produced heat by 50% and a reduction of 0.5 (50%) will reduce the power and heat of idle axes to about 25%. If the reduction is set below 0.3 (< 30%), the closed loop will be disabled while applied.

SYNTAX

```
int LSX_SetReduction (int     lLSID,
                      double  dX,
                      double  dY,
                      double  dZ,
                      double  dA);
```

TANGO CMD !reduction

PARAMETERS **dX...dA**: Electrical motor current reduction 0.00 ... 1.00 (= 0...100%)

EXAMPLE

```
LSX_SetReduction(Tango1, 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
```

GetReductionSingleAxis new in 1.533

Reads the current reduction of a single axis. Single-axis variant of GetReduction.

SYNTAX

```
int LSX_GetReductionSingleAxis (int     lLSID,
                                int     lAxis,
                                double *pdReduction);
```

TANGO CMD ?reduction x,y,z,a

PARAMETERS

lAxis: 1=X, 2=Y, 3=Z, 4=A

pdReduction: current reduction factor

EXAMPLE

```
LSX_GetReductionSingleAxis(Tango1, 1, &dReduction);
```

SetReductionSingleAxis new in 1.533

Sets the current reduction of a single axis. Single-axis variant of SetReduction.

SYNTAX

```
int LSX_SetReductionSingleAxis (int     lLSID,
                                int     lAxis,
                                double  dReduction);
```

TANGO CMD !reduction x,y,z,a

PARAMETERS **lAxis:** 1=X, 2=Y, 3=Z, 4=A
dReduction: current reduction factor

EXAMPLE

```
LSX_SetReductionSingleAxis(Tango1, 1, 1.0);
```

GetCurrentDelay

Reads the time delay for motor current reduction.

SYNTAX

```
int LSX_GetCurrentDelay (int  lLSID,
                         int *plX,
                         int *plY,
                         int *plZ,
                         int *plA);
```

TANGO CMD ?curdelay

PARAMETERS **plX...plA**: Time delay [ms]

EXAMPLE

```
LSX_GetCurrentDelay(Tango1, &X, &Y,&Z,&A);
```

SetCurrentDelay

Sets the time delay, after which the motor current reduction is applied.

SYNTAX

```
int LSX_SetCurrentDelay (int  lLSID,
                         int  lX,
                         int  lY,
                         int  lZ,
                         int  lA);
```

TANGO CMD !curdelay

PARAMETERS **lX...lA**: 0...65000 [ms] (A delay of 0 disables the current reduction = default)

EXAMPLE

```
LSX_SetCurrentDelay(Tango1, 100, 300, 1000, 0);
```

GetCurrentDelaySingleAxis new in 1.533

Reads the current reduction delay of a single axis. Single-axis variant of GetCurrentDelay.

SYNTAX
```
int LSX_GetCurrentDelaySingleAxis (int  lLSID,
                                   int  lAxis,
                                   int *plCurrentDelay);
```

TANGO CMD ?curdelay x,y,z,a

PARAMETERS
lAxis: 1=X, 2=Y, 3=Z, 4=A
plCurrentDelay: current reduction delay [ms]

EXAMPLE
```
LSX_GetCurrentDelaySingleAxis(Tango1, 1, &lCurrentDelay);
```

SetCurrentDelaySingleAxis new in 1.533

Sets the current reduction delay of a single axis. Single-axis variant of SetCurrentDelay.

SYNTAX
```
int LSX_SetCurrentDelaySingleAxis (int  lLSID,
                                   int  lAxis,
                                   int  lCurrentDelay);
```

TANGO CMD !curdelay x,y,z,a

PARAMETERS
lAxis: 1=X, 2=Y, 3=Z, 4=A
lCurrentDelay: current reduction delay [ms]

EXAMPLE
```
LSX_SetCurrentDelaySingleAxis(Tango1, 1, 0);
```

GetSpeedPoti

Shows whether the speed potentiometer functionality is switched on or off.
Speed potentiometer shall not be used. It slows down the controller execution times and is not supported on all Tangos and firmware versions.

SYNTAX
```
int LSX_GetSpeedPoti (int   lLSID,
                      BOOL *pbSpePoti);
```

TANGO CMD ?pot

PARAMETER: **pbSpePoti:** This flag shows whether potentiometer is switched on (1) or off (0)

EXAMPLE
```
LSX_GetSpeedPoti(Tango1, &flag);
```

SetSpeedPoti

Switches the Speed Potentiometer functionality on or off.
Speed potentiometer shall not be used. It slows down the controller execution times and is not supported on all Tangos and firmware versions.

SYNTAX

```
int LSX_SetSpeedPoti (int   lLSID,
                      BOOL  bSpeedPoti);
```

TANGO CMD !pot

PARAMETERS **bSpeedPoti** = FALSE = pre-set speed (velocity) is used as movement speed
= TRUE = pre-set speed (velocity) can be reduced depending on the speed-potentiometer deflection

EXAMPLE

```
LSX_SetSpeedPoti(Tango1, TRUE); // switch potentiometer mode on
```

GetStopPolarity

Reads the active polarity of the stop input signal.

SYNTAX

```
int LSX_GetStopPolarity (int   lLSID,
                         BOOL *pbHighActiv);
```

TANGO CMD ?stoppol

PARAMETERS **pbHighActiv**: TRUE = stop input is high active
FALSE = stop input is low active

EXAMPLE

```
LSX_GetStopPolarity(Tango1, &HighActiv);
```

SetStopPolarity

Sets the 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.

SYNTAX

```
int LSX_SetStopPolarity (int   lLSID,
                         BOOL  bHighActiv);
```

TANGO CMD !stoppol

PARAMETERS **bHighActiv**: TRUE = stop input high active
FALSE = stop input low active

EXAMPLE

```
LSX_SetStopPolarity(Tango1, FALSE);
// stop input is low active (e.g. normally open switch to ground)
```

GetVel

Read the velocity of all axes.

SYNTAX

```
int LSX_GetVel (int     lLSID,
                double *pdX,
                double *pdY,
                double *pdZ,
                double *pdA);
```

TANGO CMD ?vel

PARAMETERS **pdX...pdA**: Velocity values [depends on dimension: rev/sec or mm/s]

EXAMPLE

```
LSX_GetVel(Tango1, &X, &Y, &Z, &A);
```

SetVel

Sets the velocity of all axes.

SYNTAX

```
int LSX_SetVel (int     lLSID,
                double  dX,
                double  dY,
                double  dZ,
                double  dA);
```

TANGO CMD !vel

PARAMETERS **dX...dA**: > 0 ... max. speed [depends on dimension: rev/sec or mm/s]

EXAMPLE

```
LSX_SetVel(Tango1, 20.0, 15.0, 0.5, 10);
```

GetVelSingleAxis new in 1.533

Reads the velocity of a single axis. Single-axis variant of GetVel. Read resolution is firmware-dependent (neo = double64, up to 15 / old = float32, up to 7 significant digits).

SYNTAX

```
int LSX_GetVelSingleAxis (int     lLSID,
                          int     lAxis,
                          double *pdVel);
```

TANGO CMD ?vel x,y,z,a

PARAMETERS

lAxis: 1=X, 2=Y, 3=Z, 4=A

pdVel: velocity [mm/s]

EXAMPLE

```
LSX_GetVelSingleAxis(Tango1, 1, &dVel);
```

SetVelSingleAxis

Sets the velocity of a single axis.

SYNTAX

```
int LSX_SetVelSingleAxis (int     lLSID,
                          int     lAxis,
                          double  dVel);
```

TANGO CMD !vel x,y,z,a

PARAMETERS **lAxis**: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
dVel: >0 to max. speed [depends on dimension: rev/sec or mm/s]

EXAMPLE

```
LSX_SetVelSingleAxis(Tango1, 2, 10.0); // sets speed of Y-Axis to 10 [rev/s or mm/s]
```

GetSecVel

Reads the secure velocity of all axes.
The secure velocity limits the axis velocity to secvel
until the travel range of the axis is known (calibration and range measure executed).

SYNTAX

```
int LSX_GetSecVel (int     lLSID,
                   double *pdX,
                   double *pdY,
                   double *pdZ,
                   double *pdA);
```

TANGO CMD ?secvel

PARAMETERS **pdX...pdA**: Velocity values [mm/s]

EXAMPLE

```
LSX_GetSecVel(Tango1, &X, &Y, &Z, &A);
```

SetSecVel

Sets the secure velocity of all axes.
The secure velocity limits the axis velocity to secvel
until the travel range of the axis is known (calibration and range measure executed).
Default = 10mm/s, max. = 100mm/s.

SYNTAX

```
int LSX_SetSecVel (int     lLSID,
                   double  dX,
                   double  dY,
                   double  dZ,
                   double  dA);
```

TANGO CMD !secvel

PARAMETERS **dX...dA**: >0 ... max. speed [mm/s]

EXAMPLE

```
LSX_SetSecVel(Tango1, 20.0, 15.0, 0.5, 10);
```

GetSecVelSingleAxis new in 1.533

Reads the secure velocity of a single axis. Single-axis variant of GetSecVel. Read resolution is firmware-dependent (neo = double64, up to 15 / old = float32, up to 7 significant digits).

SYNTAX

```
int LSX_GetSecVelSingleAxis (int     lLSID,
                             int     lAxis,
                             double *pdSecVel);
```

TANGO CMD ?secvel x,y,z,a

PARAMETERS

lAxis: 1=X, 2=Y, 3=Z, 4=A
pdSecVel: secure velocity [mm/s]

EXAMPLE

```
LSX_GetSecVelSingleAxis(Tango1, 1, &dSecVel);
```

SetSecVelSingleAxis

Sets the secure velocity of a single axis.
The secure velocity limits the axis velocity to secvel
until the travel range of the axis is known (calibration and range measure executed).
Default = 10mm/s, max. = 100mm/s.

SYNTAX

```
int LSX_SetSecVelSingleAxis (int     lLSID,
                             int     lAxis,
                             double  dVel);
```

TANGO CMD !secvel x,y,z,a

PARAMETERS

lAxis: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
dVel: >0 ... max. speed [mm/s]

EXAMPLE

```
LSX_SetSecVelSingleAxis(Tango1, 2, 10.0); // sets secure speed of Y-Axis
to 10 mm/s
```

GetVelFac

Reads the velocity reduction factor of all axes.
A velocity factor which is multiplied to the velocity setting.
This factor should be left at the default of 1.0 and only used for applications where it is required.
Best case is trying to just set the velocity as required.

SYNTAX

```
int LSX_GetVelFac (int     lLSID,
                   double *pdX,
                   double *pdY,
                   double *pdZ,
                   double *pdA);
```

TANGO CMD ?velfac

PARAMETERS

pdX...pdA: Velocity factor, default = 1

EXAMPLE

```
LSX_GetVelFac(Tango1, &X, &Y, &Z, &A);
```

SetVelFac

Sets the velocity reduction factor.
A velocity factor which is multiplied to the velocity setting. Should be left at the default and only used for applications where it is required. Else just set !vel accordingly.

SYNTAX

```
int LSX_SetVelFac (int     lLSID,
                   double  dX,
                   double  dY,
                   double  dZ,
                   double  dA);
```

TANGO CMD !velfac

PARAMETERS **dX...dA**: Velocity reduction factor, within parameters 0.01 -- 1.00

EXAMPLE

```
LSX_SetVelFac(Tango1, 1, 1, 0.1, 0.1);
// reduces velocity of Z and A axes to 1/10 of nominal velocity
```

GetAccel

Reads the acceleration setting of all axes.

SYNTAX

```
int LSX_GetAccel (int     lLSID,
                  double *pdX,
                  double *pdY,
                  double *pdZ,
                  double *pdA);
```

TANGO CMD ?accel

PARAMETERS **pdX...pdA**: Acceleration values [m/s^2]

EXAMPLE

```
LSX_GetAccel(Tango1, &X, &Y, &Z, &A);
```

SetAccel

Set the acceleration of all axes.

SYNTAX

```
int LSX_SetAccel (int     lLSID,
                  double  dX,
                  double  dY,
                  double  dZ,
                  double  dA);
```

TANGO CMD !accel

PARAMETERS **dX...dA**: 0.01 - 20.00 [m/s^2]

EXAMPLE

```
LSX_SetAccel(Tango1, 1.0, 1.5, 0, 0);
```

GetAccelSingleAxis new in 1.533

Reads the acceleration of a single axis. Single-axis variant of GetAccel. Read resolution is firmware-dependent (neo = double64, up to 15 / old = float32, up to 7 significant digits).

SYNTAX

```
int LSX_GetAccelSingleAxis (int     lLSID,
                            int     lAxis,
                            double *pdAccel);
```

TANGO CMD ?accel x,y,z,a

PARAMETERS

lAxis: 1=X, 2=Y, 3=Z, 4=A
pdAccel: acceleration [m/s2]

EXAMPLE

```
LSX_GetAccelSingleAxis(Tango1, 1, &dAccel);
```

SetAccelSingleAxis

Sets the acceleration of a single axis.

SYNTAX

```
int LSX_SetAccelSingleAxis (int     lLSID,
                            int     lAxis,
                            double  dAccel);
```

TANGO CMD !accel x,y,z,a

PARAMETERS

lAxis: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
dAccel: Acceleration 0.01 - 20.00 [m/s²]

EXAMPLE

```
LSX_SetAccelSingleAxis(Tango1, 3, 1,0); // sets acceleration of Z-Axis
to 1.0 m/s²
```

GetAccelFunc

Reads the acceleration function setting of all axes.

SYNTAX

```
int LSX_GetAccelFunc (int  lLSID,
                      int *lX,
                      int *lY,
                      int *lZ,
                      int *lR);
```

TANGO CMD ?accelfunc

PARAMETERS

lX...lR: Acceleration function of the axes (pointer to variable)
0 = trapezoidal acceleration
1 = s-curve acceleration

EXAMPLE

```
LSX_GetAccelFunc(Tango1, &lX, &lY, &lZ, &lR);
```

SetAccelFunc

Sets the acceleration function of the acceleration ramps for move instructions (not for speed).

SYNTAX

```
int LSX_SetAccelFunc (int  lLSID,
                      int  lX,
                      int  lY,
                      int  lZ,
                      int  lR);
```

TANGO CMD !accelfunc

PARAMETERS **lX...lR**: Acceleration function for the axes
0 = trapezoidal acceleration
1 = s-curve acceleration
2, 3 for jerk control, refer to the Instruction Set Documentation

EXAMPLE

```
LSX_SetAccelFunc(Tango1, lX, lY, lZ, lR);
```

GetAccelFuncSingleAxis new in 1.533

Reads the acceleration ramp function of a single axis. Single-axis variant of GetAccelFunc.

SYNTAX

```
int LSX_GetAccelFuncSingleAxis (int  lLSID,
                                int  lAxis,
                                int *plAccelFunc);
```

TANGO CMD ?accelfunc x,y,z,a

PARAMETERS **lAxis:** 1=X, 2=Y, 3=Z, 4=A
plAccelFunc: 0=trapezoidal, 1=sinusoidal

EXAMPLE

```
LSX_GetAccelFuncSingleAxis(Tango1, 1, &lAccelFunc);
```

SetAccelFuncSingleAxis new in 1.533

Sets the acceleration ramp function of a single axis. Single-axis variant of SetAccelFunc.

SYNTAX

```
int LSX_SetAccelFuncSingleAxis (int  lLSID,
                                int  lAxis,
                                int  lAccelFunc);
```

TANGO CMD !accelfunc x,y,z,a

PARAMETERS **lAxis:** 1=X, 2=Y, 3=Z, 4=A
lAccelFunc: 0=trapezoidal, 1=sinusoidal

EXAMPLE

```
LSX_SetAccelFuncSingleAxis(Tango1, 1, 0);
```

GetStopAccel

Retrieves the deceleration for stop conditions, cal and rm.

SYNTAX
```
int LSX_GetStopAccel (int     lLSID,
                      double *pdXD,
                      double *pdYD,
                      double *pdZD,
                      double *pdAD);
```

TANGO CMD ?stopaccel

PARAMETERS **pdXD...pdAD**: Deceleration values [m/s²]

EXAMPLE
```
LSX_GetStopAccel(Tango1, &XD, &YD, &ZD, &AD);
```

SetStopAccel

Sets the deceleration used when moving into a limit switch or causing a stop condition.
If the axis acceleration (set with SetAccel) is higher, then this higher value will be used.

SYNTAX
```
int LSX_SetStopAccel (int     lLSID,
                      double  dX,
                      double  dY,
                      double  dZ,
                      double  dA);
```

TANGO CMD !stopaccel

PARAMETERS **dX...dA**: Brake acceleration, within parameters 0.01 to 20 [m/s²]

EXAMPLE
```
LSX_SetStopAccel(Tango1, 1.5, 1.5, 1.5, 1.5);
```

GetStopAccelSingleAxis new in 1.533

Reads the deceleration of a single axis. Single-axis variant of GetStopAccel. Read resolution is firmware-dependent (neo = double64, up to 15 / old = float32, up to 7 significant digits).

SYNTAX
```
int LSX_GetStopAccelSingleAxis (int     lLSID,
                                int     lAxis,
                                double *pdStopAccel);
```

TANGO CMD ?stopaccel x,y,z,a

PARAMETERS
lAxis: 1=X, 2=Y, 3=Z, 4=A
pdStopAccel: stop acceleration [m/s2]

EXAMPLE
```
LSX_GetStopAccelSingleAxis(Tango1, 1, &dStopAccel);
```

SetStopAccelSingleAxis new in 1.533

Sets the stop acceleration (deceleration) of a single axis. Single-axis variant of SetStopAccel.

SYNTAX

```
int LSX_SetStopAccelSingleAxis (int     lLSID,
                                int     lAxis,
                                double  dStopAccel);
```

TANGO CMD !stopaccel x,y,z,a

PARAMETERS

lAxis: 1=X, 2=Y, 3=Z, 4=A
dStopAccel: stop acceleration [m/s2]

EXAMPLE

```
LSX_SetStopAccelSingleAxis(Tango1, 1, 1.0);
```

GetBlSmoothSingleAxis

Reads the currently used backlash smoothing mode of an axis.
Backlash and backlash smoothing are used on open loop systems without encoders.
As the backlash individually compensates a deviation that occurs depending on travel direction, the blsmooth can be used to lower the impact on the compensation (like introduced shake).

SYNTAX

```
int LSX_GetBlSmoothSingleAxis (int  lLSID,
                               int  lAxis,
                               int *plBlSmooth);
```

TANGO CMD ?blsmooth x,y,z,a

PARAMETERS

lAxis 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
lBlSmooth Pointer to int for returning blsmooth mode of the specified axis

EXAMPLE

```
LSX_GetBlSmoothSingleAxis(Tango1, 2, &lBlSmooth); // Get blsmooth of Y
```

SetBlSmoothSingleAxis

Sets the currently used backlash smoothing mode of an axis.
Backlash and backlash smoothing are used on open loop systems without encoders.
As the backlash individually compensates a deviation that occurs depending on travel direction, the blsmooth can be used to lower the impact on the compensation (like introduced shake).

SYNTAX

```
int LSX_SetBlSmoothSingleAxis (int  lLSID,
                               int  lAxis,
                               int  lSmooth);
```

TANGO CMD !blsmooth x,y,z,a

PARAMETERS

lAxis: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
lBlSmooth New blsmooth mode for the axis

EXAMPLE

```
LSX_SetBlSmoothSingleAxis (Tango1, 2, 0); // Set blsmooth of Y to 0
```

LStepSave

Save the current configuration to TANGO EEPROM.
All settings made in the TANGO (velocity, limit switch etc.) will be stored permanently.
If the save command failed, the function returns error 4002.

SYNTAX `int LSX_LStepSave (int lLSID);`

TANGO CMD !save

PARAMETERS -

EXAMPLE `LSX_LStepSave(Tango1);`

SoftwareReset

TANGO will be reset and reboots.

SYNTAX `int LSX_SoftwareReset (int lLSID);`

TANGO CMD !reset

PARAMETERS -

EXAMPLE `LSX_SoftwareReset(Tango1);`

4.6. Move Functions and Position Management

Calibrate

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.
It is recommended to use the CalibrateEx function in case not all axes are allowed to travel at once. It is also possible to disable axes prior to calling Calibrate() and re-enable them afterwards by the SetActiveAxes() function.

SYNTAX
```
int LSX_Calibrate (int  lLSID);
```

TANGO CMD !cal

PARAMETERS -

EXAMPLE
```
LSX_Calibrate(Tango1);
```

CalibrateEx

Calibrates the by flag selected axis or axes.
Only calibrates axes with corresponding Bit set in lFlags.

SYNTAX
```
int LSX_CalibrateEx (int  lLSID,
                     int  lFlags);
```

TANGO CMD !cal

PARAMETERS

lFlags: Bit mask for the axes to be calibrated
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
```
LSX_CalibrateEx(Tango1, 6); // only calibrate Y- and Z-Axis (Bit 1 and 2 set = 2+4 = 6)
```

RMeasure

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.

SYNTAX
```
int LSX_RMeasure (int  lLSID);
```

TANGO CMD !rm

PARAMETERS -

EXAMPLE
```
LSX_RMeasure(Tango1);
```

RMeasureEx

Measure maximum position of the by flag selected axis or axes.
Only range measures axes with corresponding Bit set in lFlags.

SYNTAX
```
int LSX_RMeasureEx (int  lLSID,
                    int  lFlags);
```

TANGO CMD !rm

PARAMETERS

lFlags: Bit mask for the axes to be range measured
Bit 0=X, Bit 1=Y, Bit 2=Z, Bit 3=A
Bit 2 = 1 = range measure Z-Axis
Bit 2 = 0 = Do not range measure Z-Axis
...

EXAMPLE
```
LSX_RMeasureEx(Tango1, 3); // measure maximum position of X- and Y-Axis (1+2=3)
```

GetDelay

Retrieves time delay (wait time) until a commanded move is executed.

SYNTAX
```
int LSX_GetDelay (int  lLSID,
                  int *plDelay);
```

TANGO CMD ?delay

PARAMETERS **plDelay**: Delay [ms]

EXAMPLE
```
LSX_GetDelay(Tango1, &Delay);
```

SetDelay

Sets the time for which move commands are delayed.
Before each positioning the controller waits for this period of time delay, default = 0.

SYNTAX

```
int LSX_SetDelay (int  lLSID,
                  int  lDelay);
```

TANGO CMD !delay

PARAMETERS **lDelay**: 0 - 10000 [ms]

EXAMPLE

```
LSX_SetDelay(Tango1, 1000); // 1 Second delay until a move command is executed
```

MoveAbs

All axes are moved to the here specified absolute positions.

SYNTAX

```
int LSX_MoveAbs (int     lLSID,
                 double  dX,
                 double  dY,
                 double  dZ,
                 double  dA,
                 BOOL    bWait);
```

TANGO CMD !moa

PARAMETERS **dX...dA**: ± Travel range, command depends on measuring unit
bWait: Determines, whether function shall return after reaching position (= TRUE)
or directly after sending the command (= FALSE)

EXAMPLE

```
LSX_MoveAbs(Tango1, 10.0, 10.0, -10.0, 10.0, TRUE);
```

MoveAbsSingleAxis

Positions a single axis to the specified absolute position.
(Remarks: As autostatus here is always 0, this instruction will not generate a "trigger after position reached", which requires autostatus being set to 1 or 3)

SYNTAX

```
int LSX_MoveAbsSingleAxis (int     lLSID,
                           int     lAxis,
                           double  dValue,
                           BOOL    bWait);
```

TANGO CMD !moa x,y,z,a

PARAMETERS

lAxis: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
dValue: Position, command depends on measuring unit (dimension)
bWait: TRUE = Wait until the specified axis has reached its position
(The DLL will do by polling the axis **)
FALSE = No wait, function returns after sending the move
** The wait will only check for the specified axis. Therefore, autostatus will be set to 0
and the axis state is polled.

EXAMPLE

```
LSX_MoveAbsSingleAxis(Tango1, 2, 10.0, TRUE);
// Moves the Y-Axis to 10mm (mm if dim=2 or 9) and waits until reached
```

MoveEx

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.

SYNTAX

```
int LSX_MoveEx (int     lLSID,
                double  dX,
                double  dY,
                double  dZ,
                double  dA,
                BOOL    bRelative,
                BOOL    bWait,
                int     lAxisCount);
```

TANGO CMD

!mor

PARAMETERS

dX...dA: Position vectors

bRelative: FALSE = values of X, Y, Z and A are interpreted as absolute coordinates
TRUE = values are interpreted as relative coordinates to current position

bWait: TRUE = the function doesn't return before reaching the target position,
FALSE = function returns immediately after sending the command to the TANGO.

lAxisCount: 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

```
LSX_MoveEx(Tango1, 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
```

MoveRel

Move axes by relative positions.
All axes are moved by the transmitted distances. All axes reach their destinations simultaneously.

SYNTAX

```
int LSX_MoveRel (int     lLSID,
                 double  dX,
                 double  dY,
                 double  dZ,
                 double  dA,
                 BOOL    bWait);
```

TANGO CMD !mor

PARAMETERS

dX...dA: +/- Travel range, command depends on measuring unit (dimension)
bWait: TRUE = function waits until position is reached
FALSE = function does not wait

EXAMPLE

```
LSX_MoveRel(Tango1, 10.0, 10.0, -10.0, 10.0, TRUE);
```

MoveRelSingleAxis

Move single axis relative. Similar to MoveAbsSingleAxis, but for relative dist.

SYNTAX

```
int LSX_MoveRelSingleAxis (int     lLSID,
                           int     lAxis,
                           double  dValue,
                           BOOL    bWait);
```

TANGO CMD !mor x,y,z,a

PARAMETERS

lAxis: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
dValue: Distance, command depends on set measuring unit, **Wait:** TRUE or FALSE

EXAMPLE

```
LSX_MoveRelSingleAxis(Tango1, 3, 5,0, TRUE);
// Z-Axis is moved by 5mm in positive direction, the function waits
until reached
```

MoveRelShort

Relative positioning with short command (m).
This command may be used to execute several fast equal distance relative moves without communication overhead.
Distances must be pre-set once with LSX_SetDistance or are taken from the most recent LSX_MoveRel distances.

SYNTAX
```
int LSX_MoveRelShort (int  lLSID);
```

TANGO CMD m

PARAMETERS -

EXAMPLE
```
LSX_SetDistance(Tango1, 1.0, 1.0, 0, 0);
for (i = 0; i < 10; i++) LSX_MoveRelShort(Tango1);
// position X- and Y-Axis 10 times relatively by 1mm
```

GetDistance

Retrieves the distance values last used for LSX_MoveRelShort or specified by SetDistance.

SYNTAX
```
int LSX_GetDistance (int     lLSID,
                     double *pdX,
                     double *pdY,
                     double *pdZ,
                     double *pdA);
```

TANGO CMD ?distance

PARAMETERS **pdX...pdA**: Current distances of all axes, depending on corresponding measuring unit.

EXAMPLE
```
LSX_GetDistance(Tango1, &X, &Y, &Z, &A);
```

SetDistance

Sets the distance parameters for command LSX_MoveRelShort.
This enables very fast equal distance relative positioning without the need of communication overhead.

SYNTAX
```
int LSX_SetDistance (int     lLSID,
                     double  dX,
                     double  dY,
                     double  dZ,
                     double  dA);
```

TANGO CMD !distance

PARAMETERS **dX...dA**: Min-/max- travel range, values depend on measuring unit.

EXAMPLE
```
LSX_SetDistance(Tango1, 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
```

Go

All axes are moved to the here specified absolute positions.
Go can be sent while preceding Go still is in progress.
It is designed to be called directly from mouse events, touchscreens, etc.

SYNTAX

```
int LSX_Go (int     lLSID,
            double  dX,
            double  dY,
            double  dZ,
            double  dA);
```

TANGO CMD go

PARAMETERS **dX...dA**: ± Travel range, command depends on measuring unit

EXAMPLE

```
LSX_Go(Tango1, 10.0, 10.0, -10.0, 10.0);
```

GoSingleAxis

Move one axis to the specified absolute position.
GoSingleAxis can be called even while a preceding Go or GoSingleAxis is in progress.
It is designed to be called directly from mouse events, touchscreens etc. to move axes.

SYNTAX

```
int LSX_GoSingleAxis (int     lLSID,
                      int     lAxis,
                      double  dValue);
```

TANGO CMD go x,y,z,a

PARAMETERS **lAxis**: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
dValue: ± Travel range, command depends on measuring unit

EXAMPLE

```
LSX_GoSingleAxis(Tango1, 2, 12.34); //move Y to target position 12.34
```

GoEx

Similar to Go() command, but with an additional parameter.
The number of axes, which are to be moved, can be specified by using AxisCount parameter.
For example this function can be used to move X and Y without Z.

SYNTAX

```
int LSX_GoEx (int     lLSID,
              double  dX,
              double  dY,
              double  dZ,
              double  dA,
              int     lAxisCount);
```

TANGO CMD go

PARAMETERS

dX...dA: Position vectors
lAxisCount: 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

```
LSX_GoEx(Tango1, 2.0, 3.0, 56.78, 67.89, 2);
// X and Y are moved relatively by 2 or 3 while Z and A will not move
```

GetDigJoySpeed

Retrieves the current travel speed as initiated by SetDigJoySpeed.

SYNTAX

```
int LSX_GetDigJoySpeed (int     lLSID,
                        double *pdX,
                        double *pdY,
                        double *pdZ,
                        double *pdA);
```

TANGO CMD ?speed

PARAMETERS **pdX...pdA**: Speed values [motor rev/s] or [mm/s in case of Dimension 9 and 10]

EXAMPLE

```
LSX_GetDigJoySpeed(Tango1, &X, &Y, &Z, &A);
```

SetDigJoySpeed

Moves axes at a constant speed.
The speed values can be overwritten without the need to stop.
To stop a speed move, the speed of the axes to stop can be set to 0.
Else the constant speed is maintained until approaching a limit switch
or StopAxes (abort, a) is executed.
The Speed / DigJoySpeed function can be disabled for individual axes
by setting **SetJoyDir** of the axes to 0. SetJoyDir does not change the DigJoySpeed direction. This must be done by inverting the speed values of SetDigJoySpeed.

SYNTAX

```
int LSX_SetDigJoySpeed (int     lLSID,
                        double  dX,
                        double  dY,
                        double  dZ,
                        double  dA);
```

TANGO CMD !speed

PARAMETERS

dX...dA: Speed, within parameter range +- max. speed
Depends on dimension in [motor revolutions/sec] or [mm/s] in Dimensions 9 and 10

EXAMPLE

```
LSX_SetDigJoySpeed(Tango1, 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“
```

GetDigJoySpeedSingleAxis new in 1.533

Read digital joystick speed of a single axis. Single-axis variant of GetDigJoySpeed. Read resolution is firmware-dependent (neo = double64, up to 15 / old = float32, up to 7 significant digits).

SYNTAX

```
int LSX_GetDigJoySpeedSingleAxis (int     lLSID,
                                  int     lAxis,
                                  double *pdDigJoySpeed);
```

TANGO CMD ?speed x,y,z,a

PARAMETERS

lAxis: 1=X, 2=Y, 3=Z, 4=A
pdDigJoySpeed: digital joystick speed [mm/s]

EXAMPLE

```
LSX_GetDigJoySpeedSingleAxis(Tango1, 1, &dDigJoySpeed);
```

SetDigJoySpeedSingleAxis new in 1.533

Sets the digital joystick speed of a single axis. Single-axis variant of SetDigJoySpeed.

SYNTAX

```
int LSX_SetDigJoySpeedSingleAxis (int     lLSID,
                                  int     lAxis,
                                  double  dDigJoySpeed);
```

TANGO CMD !speed x,y,z,a

PARAMETERS

lAxis: 1=X, 2=Y, 3=Z, 4=A

dDigJoySpeed: digital joystick speed [mm/s]

EXAMPLE

```
LSX_SetDigJoySpeedSingleAxis(Tango1, 1, 10.0);
```

StopAxes

Abort (a).
Stops all moving axes with their stop acceleration.

SYNTAX

```
int LSX_StopAxes (int  lLSID);
```

TANGO CMD a

PARAMETERS -

EXAMPLE

```
LSX_StopAxes(Tango1);
```

StopAxesEx

Abort (a).
Stops the specified moving axis/axes with their stop acceleration.
Axes are specified as integer bitmask (1=X, 2=Y, 4=Z, 8=A)

SYNTAX

```
int LSX_StopAxesEx (int  lLSID,
                    int  lFlags);
```

TANGO CMD a

PARAMETERS

lFlags: Axis bits of the axes to stop, 0 ... 15 (0...0x0F)

EXAMPLE

```
LSX_StopAxesEx(Tango1, 3); // 3: Stop X+Y axis
```

WaitForAxisStop

This function returns as soon as the axes selected by the bit mask “lAFlags” have reached their target positions or the timeout is exceeded (when the TANGO sends its completion reply). LSX_WaitForAxisStop uses '?statusaxis' to poll axis status, therefore it will not generate a “trigger after position reached”, if that trigger mode is required.

SYNTAX

```
int LSX_WaitForAxisStop (int   lLSID,
                         int   lAFlags,
                         int   lATimeoutValue,
                         BOOL *pbATimeout);
```

TANGO CMD

-

PARAMETERS

lAFlags: Bit mask
Bit 0 = X-Axis
Bit 1 = Y-Axis
Bit 2 = Z-Axis
Bit 3 = A-Axis

lATimeoutValue: 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 (TRUE) or not (FALSE)

EXAMPLE

```
LSX_WaitForAxisStop(Tango1, 3, 1000, pbATimeout);
// wait until X- and Y-Axes have stopped, wait will end (timeout) after
1 second
LSX_WaitForAxisStop(Tango1, 7, 20000, pbATimeout);
// wait until X-, Y- and Z-Axis have stopped, wait will end (timeout)
after 20 sec.
```

4.7. Joystick and Handwheel

SetJoystickOff

Switch the HDI off (Joystick, ERGODRIVE, etc.)

SYNTAX

```
int LSX_SetJoystickOff (int  lLSID);
```

TANGO CMD !joy

PARAMETERS -

EXAMPLE

```
LSX_SetJoystickOff(Tango1);
```

SetJoystickOn

Switch the HDI on (Joystick, ERGODRIVE, etc.)

SYNTAX

```
int LSX_SetJoystickOn (int   lLSID,
                       BOOL  bPositionCount,
                       BOOL  bEncoder);
```

TANGO CMD !joy

PARAMETERS

bPositionCount = TRUE = dummy (position count on)
= FALSE = dummy (position count off)
bEncoder = TRUE = dummy (encoder values, if encoders available)

EXAMPLE

```
LSX_SetJoystickOn(Tango1, TRUE, TRUE);
// switch on joystick with position count (encoder values)
```

GetJoystickDir

Retrieves the individual axis direction and enable state for HDI input devices.

SYNTAX

```
int LSX_GetJoystickDir (int  lLSID,
                        int *plXD,
                        int *plYD,
                        int *plZD,
                        int *plAD);
```

TANGO CMD ?joydir

PARAMETERS **plXD...plAD:**

0 = Axis disabled for Joystick (deflection ignored)
1 = positive axis direction, current reduction disabled (treated by TANGO as 2)
-1 = negative axis direction, current reduction disabled (treated by TANGO as -2)
2 = positive axis direction with current reduction (default)
-2 = negative axis direction with current reduction

EXAMPLE

```
LSX_GetJoystickDir(Tango1, &XD, &YD, &ZD, &AD);
```

SetJoystickDir

Sets the individual axis direction and enable state for Joystick and other HDI input devices. Individual axes can be reversed or disabled.
Setting JoystickDir to 0 also disables the "!speed" moves and the **SetDigJoySpeed** of the corresponding axes, but a negative value does not reverse them.

SYNTAX

```
int LSX_SetJoystickDir (int  lLSID,
                        int  lXD,
                        int  lYD,
                        int  lZD,
                        int  lAD);
```

TANGO CMD !joydir

PARAMETERS **lXD...lAD:**

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

```
LSX_SetJoystickDir(Tango1, 1, 1, -1, 0);
// X- and Y-Axis positive direction, Z-Axis negative direction, A-Axis blocked
```

GetJoystickDirSingleAxis new in 1.533

Read the joystick direction and enable state of a single axis. Single-axis variant of GetJoystickDir.

SYNTAX
```
int LSX_GetJoystickDirSingleAxis (int  lLSID,
                                  int  lAxis,
                                  int *plJoystickDir);
```

TANGO CMD ?joydir x,y,z,a

PARAMETERS
lAxis: 1=X, 2=Y, 3=Z, 4=A
plJoystickDir: Joystick direction 0, ±1, ±2

EXAMPLE
```
LSX_GetJoystickDirSingleAxis(Tango1, 1, &lJoystickDir);
```

SetJoystickDirSingleAxis new in 1.533

Set the joystick direction and enable state of a single axis. Single-axis variant of SetJoystickDir.

SYNTAX
```
int LSX_SetJoystickDirSingleAxis (int  lLSID,
                                  int  lAxis,
                                  int  lJoystickDir);
```

TANGO CMD !joydir x,y,z,a

PARAMETERS
lAxis: 1=X, 2=Y, 3=Z, 4=A
lJoystickDir: Joystick direction 0, ±1, ±2

EXAMPLE
```
LSX_SetJoystickDirSingleAxis(Tango1, 1, 1);
```

GetJoystick

Retrieves Joystick and in general the HDI status.

SYNTAX
```
int LSX_GetJoystick (int   lLSID,
                     BOOL *pbJoystickOn,
                     BOOL *pbManual,
                     BOOL *pbPositionCount,
                     BOOL *pbEncoder);
```

TANGO CMD ?joy

PARAMETERS
pbJoystickOn: TRUE = Joystick switched on
pbManual: FALSE = Joystick switch set on automatic
TRUE = Joystick is switched on manually via switch
pbPositionCount: TRUE = position count switched on
pbEncoder: TRUE = encoder values, if available

EXAMPLE
```
LSX_GetJoystick(Tango1, &JoystickOn, &Manual, &PositionCount,
&Encoder);
```

GetJoyChangeAxis

Retrieves Joystick X<->Y axis change state.

SYNTAX
```
int LSX_GetJoyChangeAxis (int   lLSID,
                          BOOL *pbChangedXY);
```

TANGO CMD ?joychangeaxis

PARAMETERS **bChangedXY**: TRUE = Joystick X and Y axes assignment swapped
FALSE = Normal operation: X controls X, Y controls Y (default)

EXAMPLE
```
LSX_GetJoyChangeAxis(Tango1, &bChangedXY);
```

JoyChangeAxis

Set Joystick X<->Y axis change state.

SYNTAX
```
int LSX_JoyChangeAxis (int   lLSID,
                       BOOL  bChangeXY);
```

TANGO CMD !joychangeaxis

PARAMETERS **bChangeXY**: TRUE = Joystick X and Y axes assignment swapped
FALSE = Normal operation: X controls X, Y controls Y (default)

EXAMPLE
```
LSX_JoyChangeAxis(Tango1, bChangeXY);
```

GetHandWheel

Retrieves hand wheel status.
Not supported by TANGO controllers! Use GetJoystick() instead.

SYNTAX
```
int LSX_GetHandWheel (int   lLSID,
                      BOOL *pbHandWheelOn,
                      BOOL *pbPositionCount,
                      BOOL *pbEncoder);
```

TANGO CMD ?hw

PARAMETERS
pbHandWheelOn: TRUE = hand wheel switched on
FALSE = hand wheel switched off
pbPositionCount: TRUE = position count switched on
FALSE = position count switched off
pbEncoder: TRUE = encoder values, if available

EXAMPLE
```
LSX_GetHandWheel(Tango1, &HandWheelOn, &PositionCount, &Encoder);
```

SetHandWheelOff

Switch hand wheel off.
Not supported by TANGO controllers! Use SetJoystickOff() instead.

SYNTAX
```
int LSX_SetHandWheelOff (int  lLSID);
```

TANGO CMD !hw

PARAMETERS -

EXAMPLE
```
LSX_SetHandWheelOff(Tango1);
```

SetHandWheelOn

Switch hand wheel on.
Not supported by TANGO controllers! Use SetJoystickOn() instead.

SYNTAX
```
int LSX_SetHandWheelOn (int   lLSID,
                        BOOL  bPositionCount,
                        BOOL  bEncoder);
```

TANGO CMD !hw

PARAMETERS

bPositionCount: TRUE = position counter on
FALSE = position counter off

bEncoder TRUE = encoder values, if encoders available

EXAMPLE
```
LSX_SetHandWheelOn(Tango1, TRUE, TRUE);
// switch on hand wheel with position count (encoder values)
```

GetJoystickWindow

Retrieves idle window for analog Joysticks.

SYNTAX
```
int LSX_GetJoystickWindow (int  lLSID,
                           int *plAValue);
```

TANGO CMD ?joywindow

PARAMETERS **plAValue**: Analog signal range (as digits) in which axes do not move.

EXAMPLE
```
LSX_GetJoystickWindow(Tango1, &AValue);
```

SetJoystickWindow

Set Joystick idle window for analog Joysticks.
A value in digits which configures an angle where an 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. Does not apply to TANGOs with digital HDI.

SYNTAX

```
int LSX_SetJoystickWindow (int  lLSID,
                           int  lAValue);
```

TANGO CMD !joywindow

PARAMETERS **lAValue**: Analog signal range (as digits) in which axes do not move. 0... 100

EXAMPLE

```
LSX_SetJoystickWindow(Tango1, 30);
```

GetHwFactor

Read the hand wheel factor of all axes, in [mm per knob rotation].

SYNTAX

```
int LSX_GetHwFactor (int     lLSID,
                     double *pdX,
                     double *pdY,
                     double *pdZ,
                     double *pdA);
```

TANGO CMD ?hwfactor

PARAMETERS **pdX...pdA:** Pointer to double

EXAMPLE

```
LSX_GetHwFactor(Tango1, &dX, &dY, &dZ, &dA);
```

SetHwFactor

Set the hand wheel factor for all axes, in [mm per knob rotation].

SYNTAX

```
int LSX_SetHwFactor (int     lLSID,
                     double  dX,
                     double  dY,
                     double  dZ,
                     double  dA);
```

TANGO CMD !hwfactor

PARAMETERS **dX...dA:** Floating point values (double) in mm/rev

EXAMPLE

```
LSX_SetHwFactor(Tango1, dX, dY, dZ, dA);
```

GetHwFactorSingleAxis

Read the hand wheel factor of the specified axis, in [mm per knob rotation].

SYNTAX

```
int LSX_GetHwFactorSingleAxis (int     lLSID,
                               int     lAxis,
                               double *pdFactor);
```

TANGO CMD ?hwfactor x,y,z,a

PARAMETERS

lAxis: Axis number 1, 2, 3, 4 (corresponding to axes X, Y, Z, A)
pdFactor: Pointer to double

EXAMPLE

```
LSX_GetHwFactorSingleAxis(Tango1, 2, &dFactor);
```

SetHwFactorSingleAxis

Set the hand wheel factor of the specified axis, in [mm per knob rotation].

SYNTAX

```
int LSX_SetHwFactorSingleAxis (int     lLSID,
                               int     lAxis,
                               double  dFactor);
```

TANGO CMD !hwfactor x,y,z,a

PARAMETERS

lAxis: Axis number 1, 2, 3, 4 (corresponding to axes X, Y, Z, A)
dFactor: Floating point value (double) in mm/rev

EXAMPLE

```
LSX_SetHwFactorSingleAxis (Tango1, 2, 14.4); // Set Factor of Y-axis
to 14.4mm/rev
```

GetHwFactorB

Read the second hand wheel factor of all axes, in [mm per knob rotation].

SYNTAX

```
int LSX_GetHwFactorB (int     lLSID,
                      double *pdX,
                      double *pdY,
                      double *pdZ,
                      double *pdA);
```

TANGO CMD ?hwfactorb

PARAMETERS

pdX...pdA: Pointer to double for all axes

EXAMPLE

```
LSX_GetHwFactorB(Tango1, &dX, &dY, &dZ, &dA);
```

SetHwFactorB

Set the second hand wheel factor for all axes, in [mm per knob rotation].

SYNTAX

```
int LSX_SetHwFactorB (int     lLSID,
                      double  dX,
                      double  dY,
                      double  dZ,
                      double  dA);
```

TANGO CMD !hwfactorb

PARAMETERS **dX...dA:** Double values for all axes

EXAMPLE

```
LSX_SetHwFactorB(Tango1, dX, dY, dZ, dA);
```

GetHwFactorBSingleAxis

Read the hand wheel factor B of the specified axis, in [mm per knob rotation].

SYNTAX

```
int LSX_GetHwFactorBSingleAxis (int     lLSID,
                                int     lAxis,
                                double *pdFactor);
```

TANGO CMD ?hwfactorb x,y,z,a

PARAMETERS **lAxis:** Axis number 1, 2, 3, 4 (corresponding to axes X, Y, Z, A)
pdFactor: Pointer to double

EXAMPLE

```
LSX_GetHwFactorBSingleAxis(Tango1, 2, &dFactorB);
```

SetHwFactorBSingleAxis

Set the hand wheel factor B of the specified axis, in [mm per knob rotation].

SYNTAX

```
int LSX_SetHwFactorBSingleAxis (int     lLSID,
                                int     lAxis,
                                double  dFactorB);
```

TANGO CMD !hwfactorb x,y,z,a

PARAMETERS **lAxis:** Axis number 1, 2, 3, 4 (corresponding to axes X, Y, Z, A)
Factor: Floating point value (double) in mm/rev

EXAMPLE

```
LSX_SetHwFactorBSingleAxis(Tango1, 2, 0.5); // Set Factor B of Y-axis
to 0.5mm/rev
```

GetZwTravel

Read the z-wheel travel distances, in [mm per knob rotation].

SYNTAX

```
int LSX_GetZwTravel (int     lLSID,
                     int     lIndex,
                     double *pdDistance);
```

TANGO CMD ?zwtravel

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

```
LSX_GetZwTravel(Tango1, lIndex, &dDistance);
```

SetZwTravel

Set the z-wheel travel distances, in [mm per knob rotation].

SYNTAX

```
int LSX_SetZwTravel (int     lLSID,
                     int     lIndex,
                     double  dDistance);
```

TANGO CMD !zwtravel

PARAMETERS

lIndex: 1 = Set standard distance
2 = Set slow distance
3 = Set fast distance

dDistance: Double value in mm/rev

EXAMPLE

```
LSX_SetZwTravel(Tango1, lIndex, dDistance);
```

GetHdiKeys

Get the HDI device key states, e.g. of Joystick or ERGODRIVE.

SYNTAX

```
int LSX_GetHdiKeys (int  lLSID,
                    int *plKey1,
                    int *plKey2,
                    int *plKey3,
                    int *plKey4);
```

TANGO CMD ?key

PARAMETERS

plKey1...4: Pointers to int, 1=Key is currently pressed, 0=key not pressed

EXAMPLE

```
LSX_GetHdiKeys(Tango1, &lKey[0], &lKey[1], &lKey[2], &lKey[3]);
```

GetKey

Get the HDI device key states, e.g. of Joystick or ERGODRIVE.
Same as GetHdiKey(), but uses BOOL pointers instead of int.

SYNTAX

```
int LSX_GetKey (int   lLSID,
                BOOL *pbKey1,
                BOOL *pbKey2,
                BOOL *pbKey3,
                BOOL *pbKey4);
```

TANGO CMD ?key

PARAMETERS **pbKey1...4:** Pointers to BOOL, TRUE=Key is currently pressed

EXAMPLE

```
LSX_GetKey(Tango1, &bKey[0], &bKey[1], &bKey[2], &bKey[3]);
```

GetKeyLatch

Get and clear HDI device key states.

SYNTAX

```
int LSX_GetKeyLatch (int   lLSID,
                     BOOL *pbKey1,
                     BOOL *pbKey2,
                     BOOL *pbKey3,
                     BOOL *pbKey4);
```

TANGO CMD ?keyl

PARAMETERS **pbKey1...4:** Pointers to BOOL, TRUE=Key was or is pressed

EXAMPLE

```
LSX_GetKeyLatch(Tango1, &bKey[0], &bKey[1], &bKey[2], &bKey[3]);
```

ClearKeyLatch

Clear latched key state(s) of one specified (1-4) or all (0) keys. No bitmask.

SYNTAX

```
int LSX_ClearKeyLatch (int  lLSID,
                       int  lKey);
```

TANGO CMD !keyl

PARAMETERS **lKey:**
0 = clear latched key state of all 4 keys
1 = clear latched key state of key 1 only
2 = clear latched key state of key 2 only
3 = clear latched key state of key 3 only
4 = clear latched key state of key 4 only

EXAMPLE

```
LSX_ClearKeyLatch(Tango1, 0); // Clear all
```

GetHdiSpeedIndex

Read the currently used HDI speed index of all axes.
The speed index is the currently (by HDI F-key) selected speed level index
e.g., from ERGODRIVE or the Multifunction Wheel

SYNTAX

```
int LSX_GetHdiSpeedIndex (int  lLSID,
                          int *plSi1,
                          int *plSi2,
                          int *plSi3,
                          int *plSi4);
```

TANGO CMD ?hdisi

PARAMETERS **plSi1...4:** Pointers to int for returning the speed index

EXAMPLE

```
LSX_GetHdiSpeedIndex(Tango1, &lSpeedIndex);
```

SetHdiSpeedIndex

Manipulate the currently used HDI speed index of all axes.
The speed index usually is the currently (by HDI F-key) selected speed level index
e.g., from ERGODRIVE or the Multifunction Wheel

SYNTAX

```
int LSX_SetHdiSpeedIndex (int  lLSID,
                          int  lSi1,
                          int  lSi2,
                          int  lSi3,
                          int  lSi4);
```

TANGO CMD !hdisi

PARAMETERS **lSi1...4:** New speed index for the HDI axes

EXAMPLE

```
LSX_SetHdiSpeedIndex(Tango1, 0, 0, 0, 0); // Set speed index to 0
```

GetHdiSpeedIndexSingleAxis

Read the currently used HDI speed index of an axes.
The speed index is the currently (by HDI F-key) selected speed level index
e.g., from ERGODRIVE or the Multifunction Wheel

SYNTAX

```
int LSX_GetHdiSpeedIndexSingleAxis (int  lLSID,
                                    int  lAxis,
                                    int *plSi);
```

TANGO CMD ?hdisi x,y,z,a

PARAMETERS **lAxis:** As number 1, 2, 3, 4 corresponding to X, Y, Z, A axes
lSi: Pointer to int for returning the speed index of the specified axis

EXAMPLE

```
LSX_GetHdiSpeedIndexSingleAxis(Tango1, 2, &lSpeedIndex); // Get Y
index
```

SetHdiSpeedIndexSingleAxis

Manipulate the currently used HDI speed index of all axes.
The speed index usually is the currently (by HDI F-key) selected speed level index
e.g., from ERGODRIVE or the Multifunction Wheel

SYNTAX

```
int LSX_SetHdiSpeedIndexSingleAxis (int  lLSID,
                                    int  lAxis,
                                    int  lSi);
```

TANGO CMD !hdisi x,y,z,a

PARAMETERS

lAxis: As number 1, 2, 3, 4 corresponding to X, Y, Z, A axes

lSi: New speed index for the HDI axis

EXAMPLE

```
LSX_SetHdiSpeedIndexSingleAxis(Tango1, 2, 0); // Set speed index of Y to 0
```

4.8. BPZ Console with Trackball and Joyspeed Keys

GetBPZ

Retrieves status of a custom-built control console with trackball.

SYNTAX

```
int LSX_GetBPZ (int  lLSID,
                int *plAValue);
```

TANGO CMD ?bpz

PARAMETERS

plAValue: 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

```
LSX_GetBPZ(Tango1, &AValue);
```

SetBPZ

Switches custom-built control console on / off.

SYNTAX

```
int LSX_SetBPZ (int  lLSID,
                int  lAValue);
```

TANGO CMD !bpz

PARAMETERS

lAValue: 0...2
0 = switch control console OFF
1 = activate control console and operate trackball at 0,1µm step resolution.
2 = activate control console and operate trackball with trackball factor.

EXAMPLE

```
LSX_SetBPZ(Tango1, 1);
```

GetBPZJoyspeed

Retrieves custom-built control console Joystick speed.

SYNTAX

```
int LSX_GetBPZJoyspeed (int     lLSID,
                        int     lAPar,
                        double *pdAValue);
```

TANGO CMD ?joyspeed

PARAMETERS

lAPar: 1, 2 or 3 (console keys for speed selection: slow, medium, fast)
pdAValue: max. speed [r/sec]

EXAMPLE

```
LSX_GetBPZJoyspeed(Tango1, &AValue); // retrieve set speed of key 1 (slow)
```

SetBPZJoyspeed

Set custom-built control console joystick speed.

SYNTAX

```
int LSX_SetBPZJoyspeed (int     lLSID,
                        int     lAPar,
                        double  dAValue);
```

TANGO CMD !joyspeed

PARAMETERS **lAPar**: 1, 2 or 3 (console keys for speed selection: slow, medium, fast)
dAValue: ±max. speed [r/sec]

EXAMPLE

```
LSX_SetBPZJoyspeed(Tango1, 1, 25); // Set key 1 parameter (slow) to speed 25
```

GetBPZTrackballBackLash

Retrieves custom-built control console trackball backlash.

SYNTAX

```
int LSX_GetBPZTrackballBackLash (int     lLSID,
                                 double *pdX,
                                 double *pdY,
                                 double *pdZ,
                                 double *pdA);
```

TANGO CMD ?bpzbl

PARAMETERS **pdX...pdA**: backlash [mm]

EXAMPLE

```
LSX_GetBPZTrackballBackLash(Tango1, &X, &Y, &Z, &A);
```

SetBPZTrackballBackLash

Set custom-built control console trackball backlash.

SYNTAX

```
int LSX_SetBPZTrackballBackLash (int     lLSID,
                                 double  dX,
                                 double  dY,
                                 double  dZ,
                                 double  dA);
```

TANGO CMD !bpzbl

PARAMETERS **dX...dA**: 0.001 to 0.15 mm

EXAMPLE

```
LSX_SetBPZTrackballBackLash(Tango1, 0.01, 0.01, 0.01, 0.01); // Set backlash for all axes to 10µm
```

GetBPZTrackballFactor

Retrieves control console trackball factor.

SYNTAX

```
int LSX_GetBPZTrackballFactor (int     lLSID,
                               double *pdAValue);
```

TANGO CMD ?bpztf

PARAMETERS **pdAValue**: Trackball factor
e.g. AValue of 3 means that one trackball pulse results in 3 motor increments.

EXAMPLE

```
LSX_GetBPZTrackballFactor(Tango1, &AValue);
```

SetBPZTrackballFactor

Set custom-built control console trackball factor.

SYNTAX

```
int LSX_SetBPZTrackballFactor (int     lLSID,
                               double  dAValue);
```

TANGO CMD !bpztf

PARAMETERS **dAValue**: 0.01... 100
AValue = 1 = Trackball factor = 1, i.e. one trackball impulse results in one motor increment

EXAMPLE

```
LSX_SetBPZTrackballFactor(Tango1, 1,0);
```

4.9. Limit Switches (Hardware and Software)

GetAutoLimitAfterCalibRM

Retrieves, whether internal software limits are set when calibrating 'cal' or measuring stage travel range 'rm'.

SYNTAX

```
int LSX_GetAutoLimitAfterCalibRM (int  lLSID,
                                  int *plFlags);
```

TANGO CMD ?nosetlimit

PARAMETERS

plFlags: 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

```
LSX_GetAutoLimitAfterCalibRM(Tango1, &Flags);
```

SetAutoLimitAfterCalibRM

Prevents setting of internal software limits when calibrating or measuring travel range.

SYNTAX

```
int LSX_SetAutoLimitAfterCalibRM (int  lLSID,
                                  int  lFlags);
```

TANGO CMD !nosetlimit

PARAMETERS

lFlags: 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

```
LSX_SetAutoLimitAfterCalibRM(Tango1, Flags);
```

GetLimit

Retrieves soft travel range limits, as set by SetLimit or cal+rm.

SYNTAX

```
int LSX_GetLimit (int     lLSID,
                  int     lAxis,
                  double *pdMinRange,
                  double *pdMaxRange);
```

TANGO CMD ?lim

PARAMETERS

lAxis: Axis from which travel range limits are to be retrieved (as number 1, 2, 3, 4 corresponding to X, Y, Z, A axes)
pdMinRange: lower travel range limit, unit depends on dimension
pdMaxRange: upper travel range limit, unit depends on dimension

EXAMPLE

```
LSX_GetLimit(Tango1, &MinRange, &MaxRange);
```

SetLimit

Set soft travel range limits.

SYNTAX

```
int LSX_SetLimit (int     lLSID,
                  int     lAxis,
                  double  dMinRange,
                  double  dMaxRange);
```

TANGO CMD !lim

PARAMETERS

lAxis: Axis from which travel range limits are to be retrieved (as number 1, 2, 3, 4 corresponding to X, Y, Z, A axes)
dMinRange: lower travel range limit, unit depends on dimension
dMaxRange: upper travel range limit, unit depends on dimension

EXAMPLE

```
LSX_SetLimit(Tango1, 1, -10.0, 20.0);
// assign X-Axis –10 as lower and 20 as upper travel range limits
```

GetLimitControl

Retrieves, whether area control (limits) is enabled or ignored.

SYNTAX

```
int LSX_GetLimitControl (int   lLSID,
                         int   lAxis,
                         BOOL *pbActive);
```

TANGO CMD ?limctr

PARAMETERS

lAxis: As number 1, 2, 3, 4 corresponding to X, Y, Z, A axes
pbActive: TRUE = area control of corresponding axis is active
FALSE = area control of corresponding axis is deactivated

EXAMPLE

```
LSX_GetLimitControl(Tango1, 3, &Active); // Read limit control enable
state of Z-Axis
```

SetLimitControl

Switches area control on / off.

SYNTAX

```
int LSX_SetLimitControl (int   lLSID,
                         int   lAxis,
                         BOOL  bActive);
```

TANGO CMD !limctr

PARAMETERS

lAxis: As number 1, 2, 3, 4 corresponding to X, Y, Z, A axes
bActive: TRUE = activate area control of corresponding axis
FALSE = disable area control of corresponding axis (no limits checked)

EXAMPLE

```
LSX_SetLimitControl(Tango1, 2, TRUE); // Enable area control of Y-Axis
is active
```

GetSwitchActive

Retrieves, whether hardware limit switches are enabled.

SYNTAX

```
int LSX_GetSwitchActive (int  lLSID,
                         int *plXA,
                         int *plYA,
                         int *plZA,
                         int *plAA);
```

TANGO CMD ?swact

PARAMETERS **plXA...plAA:** 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 its bit is set.

EXAMPLE

```
LSX_GetSwitchActive(Tango1, &XA, &YA, &ZA, &AA);
```

SetSwitchActive

Switches limit switches on / off.

SYNTAX

```
int LSX_SetSwitchActive (int  lLSID,
                         int  lXA,
                         int  lYA,
                         int  lZA,
                         int  lAA);
```

TANGO CMD !swact

PARAMETERS **lXA...lAA:** 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

```
LSX_SetSwitchActive(Tango1, 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
```

GetSwitchPolarity

Retrieves polarity of limit switches.

SYNTAX

```
int LSX_GetSwitchPolarity (int  lLSID,
                           int *plXP,
                           int *plYP,
                           int *plZP,
                           int *plAP);
```

TANGO CMD ?swpol

PARAMETERS **plXP...plAP:** 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

```
LSX_GetSwitchPolarity(Tango1, &XP, &YP, &ZP, &AP);
```

SetSwitchPolarity

Sets polarity of limit switches.

SYNTAX

```
int LSX_SetSwitchPolarity (int  lLSID,
                           int  lXP,
                           int  lYP,
                           int  lZP,
                           int  lAP);
```

TANGO CMD !swpol

PARAMETERS **lXP...lAP:** 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

```
LSX_SetSwitchPolarity(Tango1, 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
```

GetSwitchType

Retrieves type of limit switches.

SYNTAX

```
int LSX_GetSwitchType (int  lLSID,
                       int *plXP,
                       int *plYP,
                       int *plZP,
                       int *plAP);
```

TANGO CMD ?swtyp

PARAMETERS **plXP…plAP:** 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), input is for NPN type limit switch.
If bit is reset (0), input is for PNP type limit switch (default).

EXAMPLE

```
LSX_GetSwitchType(Tango1, &XP, &YP, &ZP, &RP);
```

SetSwitchType

Sets type of limit switches.

SYNTAX

```
int LSX_SetSwitchType (int  lLSID,
                       int  lXP,
                       int  lYP,
                       int  lZP,
                       int  lAP);
```

TANGO CMD !swtyp

PARAMETERS **lXP…lAP:** 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), input is configured for NPN type limit switch using pull-up resistor.
If bit is reset (0), input is configured for PNP type limit switch with pull down resistor (default).

EXAMPLE

```
LSX_SetSwitchType(Tango1, XP, YP, ZP, AP);
```

GetSwitches

Retrieves actuation status of all limit switches.
The 4 bits of the "Ref" switch are only used in systems with center reference.

SYNTAX

```
int LSX_GetSwitches (int  lLSID,
                     int *plFlags);
```

TANGO CMD ?readsw

PARAMETERS **plFlags**: 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**
Bits MSB→LSB:	0000	0000	0000
Example: 0x203 =	0010	0000	0011

= EE of Y-Axis is actuated, E0 of X- and Y-Axis are actuated

EXAMPLE

```
LSX_GetSwitches(Tango1, &Flags);
```

4.10. Digital and Analog Inputs and Outputs

GetAnalogInput

Retrieves current A/D conversion result of an analog channel.
Each TANGO controller might have its individual channels to read.
Check the TANGO Instruction Set for channel numbers and their assignment.

SYNTAX

```
int LSX_GetAnalogInput (int  lLSID,
                        int  lIndex,
                        int *plValue);
```

TANGO CMD ?anain

PARAMETERS

lIndex: 0...15 (analog channel),
0...9 = HDI connector, pins 1...10
10 = ANAIN0 of AUX-IO connector

plValue: Pointer to Integer value, to which the channel's A/D conversion result is written.
0...5V analog = 0...1023

EXAMPLE

```
LSX_GetAnalogInput(Tango1, 0, &Input); // Read channel 0
```

SetAnalogOutput

Set analog output signals.

SYNTAX

```
int LSX_SetAnalogOutput (int  lLSID,
                         int  lIndex,
                         int  lValue);
```

TANGO CMD !anaout

PARAMETERS

lIndex: 0, 1 (analog output index)

lValue: 0...100 [%]

EXAMPLE

```
LSX_SetAnalogOutput(Tango1, 0, 100);
// set analog output 0 to max. voltage (10V) (or 0 to 5V for TANGO
mini 3)
```

SetLedBright

Set the brightness of the LED100 illumination,
when connected in the default configuration (ANOUT0 and TAKT_OUT)
to the AUX I/O or AUX mini port.

The SetLedBright function also controls the TAKT_OUT digital pin
in order to entirely switch of LED100 with the LED-DR1 driver.

SYNTAX
```
int LSX_SetLedBright (int     lLSID,
                      double  dBright);
```

TANGO CMD !adigout, !anaout

PARAMETERS **dBright**: Brightness of the LED100
-1 = OFF A negative value switches the LED entirely off (IO pin)
0 ... 100 Brightness in %, up to 3 fractional digits supported

EXAMPLE
```
LSX_SetLedBright(Tango1, -1);    // set led off
LSX_SetLedBright(Tango1,  0);    // set led to lowest possible
brightness
LSX_SetLedBright(Tango1, 12.345);      // set led to 12.345%
brightness
LSX_SetLedBright(Tango1, 100);   // set led to max. brightness
```

GetAnalogOutputMode

Read the AUX-IO analog output mode.

SYNTAX
```
int LSX_GetAnalogOutputMode (int  lLSID,
                             int *plMode);
```

TANGO CMD ?anamode

PARAMETERS **plMode**: int pointer to return the anamode setting (0 ... 5)

EXAMPLE
```
LSX_GetAnalogOutputMode(Tango1, 0, &Mode); // Read AnaMode
```

SetAnalogOutputMode

Set the AUX-IO analog output mode.

SYNTAX
```
int LSX_SetAnalogOutputMode (int  lLSID,
                             int  lMode);
```

TANGO CMD !anamode

PARAMETERS **lMode**: Integer number of the desired AnaMode (0 ... 5)

EXAMPLE
```
LSX_SetAnalogOutputMode(Tango1, 5); // Activate AnaMode 5
```

SetAuxDigitalOutput

Set the specified digital output pin of the AUX-I/O port.

SYNTAX

```
int LSX_SetAuxDigitalOutput (int   lLSID,
                             int   lIndex,
                             BOOL  bValue);
```

TANGO CMD !adigout

PARAMETERS

lIndex: 0 to 3 for the output to set

TANGO 3 mini:
0 = Bit 0: AUX mini Pin 6 (TAKT_OUT, default LED100 on/off pin)
1 = Bit 1: AUX mini Pin 7 (VR_OUT)
2 = Bit 2: AUX mini Pin 8 (SHUTTER_OUT)
3 = Bit 3: AUX mini Pin 9 (TRIGGER_OUT)

Other TANGO controllers:
0 = Bit 0: AUX I/O Pin 5 (TAKT_OUT, default LED100 on/off pin)
1 = Bit 1: AUX I/O Pin 6 (VR_OUT)
2 = Bit 2: AUX I/O Pin 7 (SHUTTER_OUT)
3 = Bit 3: AUX I/O Pin 8 (TRIGGER_OUT)

bValue: FALSE = set pin to low
TRUE = set pin to high

EXAMPLE

```
LSX_SetAuxDigitalOutput(Tango1, 0, TRUE); // set output 0 to high
```

GetAuxDigitalInput

Get state of the specified digital input of the AUX-I/O port.

SYNTAX

```
int LSX_GetAuxDigitalInput (int   lLSID,
                            int   lIndex,
                            BOOL *bValue);
```

TANGO CMD ?adigin

PARAMETERS

lIndex: 0 to 3 for the digital input pin

TANGO 3 mini:
0 = Bit 0: AUX mini Pin 1 (TAKT_IN)
1 = Bit 1: Motor Connector Pin 7 (TRIN1)
2 = Bit 2: [not available]
3 = Bit 3: AUX mini Pin 2 (SNAPSHOT_IN)

Other TANGO controllers:
0 = Bit 0: AUX I/O Pin 1 (TAKT_IN)
1 = Bit 1: AUX I/O Pin 2 (VR_IN)
2 = Bit 2: AUX I/O Pin 3 (STOP)
3 = Bit 3: AUX I/O Pin 4 (SNAPSHOT2)

bValue: Pin level
FALSE = low
TRUE = high

EXAMPLE

```
LSX_GetAuxDigitalInput(Tango1, 3, &bState); // get input 3 state
```

GetDigitalInputs

Retrieve signal level of all 24 "I/O1 extension" digital input pins.

SYNTAX

```
int LSX_GetDigitalInputs (int  lLSID,
                          int *plValue);
```

TANGO CMD ?digin

PARAMETERS

plValue: Pointer to Integer value, to which the status of all inputs is written (as bit mask).
LSB = Digital input 0

EXAMPLE

```
int inputs;
LSX_GetDigitalInputs(Tango1, &inputs);
if (Inputs & 16)... // if input 4 is set...
```

GetDigitalInputsE

Retrieve signal level of all 12 "IO2 / Multi-IO" digital inputs.

SYNTAX

```
int LSX_GetDigitalInputsE (int  lLSID,
                           int *plValue);
```

TANGO CMD ?edigin

PARAMETERS

plValue: Pointer on a 32-Bit Integer, which returns the inputs 16...31 in the bits 0...15

EXAMPLE

```
int ext_inputs;
LSX_GetDigitalInputsE(Tango1, &ext_inputs);
```

SetDigitalOutput

Set individual digital output pin of IO1 extension.

SYNTAX

```
int LSX_SetDigitalOutput (int   lLSID,
                          int   lIndex,
                          BOOL  bValue);
```

TANGO CMD !digout

PARAMETERS

lIndex: 0 to 7

bValue: Set pin level to
FALSE = low
TRUE = high

EXAMPLE

```
LSX_SetDigitalOutput(Tango1, 2, TRUE); // set output pin 2 to´1´
```

SetDigitalOutputs

Set all digital output pins (0-7) of the I/O1 extension port.

SYNTAX

```
int LSX_SetDigitalOutputs (int  lLSID,
                           int  lValue);
```

TANGO CMD !digout

PARAMETERS

lValue: Bit mask, bits 0-7 determine value that is set for outputs 0-7

EXAMPLE

```
LSX_SetDigitalOutputs(Tango1, 3); // 3 = set outputs 0 and 1 to 1,
remaining pins to 0
```

SetDigitalOutputE

Set individual digital output pin of Multi I/O / IO2 port.

SYNTAX

```
int LSX_SetDigitalOutputE (int   lLSID,
                           int   lIndex,
                           BOOL  bValue);
```

TANGO CMD !edigout

PARAMETERS

lIndex: 0 to 7
bValue: Set pin level to
FALSE = low
TRUE = high

EXAMPLE

```
LSX_SetDigitalOutputE(Tango1, 2, TRUE); // set output pin 2 to´1´
```

SetDigitalOutputsE

Set digital outputs of the TANGO PCI-E or Desktop Multi I/O / IO2 port.

SYNTAX

```
int LSX_SetDigitalOutputsE (int  lLSID,
                            int  lValue);
```

TANGO CMD !edigout

PARAMETERS

lValue: Bit mask, bits 0-7 determine value that is set for outputs 0-7

EXAMPLE

```
LSX_SetDigitalOutputsE(Tango1, 5); // 5 = set outputs 0 and 2 to 1,
remaining pins=0
```

SetDigIO_Distance

NOT SUPPORTED BY TANGO Function of digital inputs / outputs.
Activate an output depending on preset distance before or after reaching designated position.

SYNTAX

```
int LSX_SetDigIO_Distance (int     lLSID,
                           int     lIndex,
                           BOOL    bFkt,
                           double  dDist,
                           int     lAxis);
```

TANGO CMD !digfkt

PARAMETERS

lIndex: 0 to 15 (output pin)
bFkt = 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
dDist: Distance, depends on selected dimension (unit)
lAxis: Axis number 1, 2, 3, 4 corresponding to X, Y, Z, A axes

EXAMPLE

```
LSX_SetDigIO_Distance(Tango1, 7, FALSE, 78.9, 3);
// output 7 is activated 78.9mm before reaching final position (Z-Axis)
```

SetDigIO_EmergencyStop

NOT SUPPORTED BY TANGO Function of digital inputs / outputs.
Assignment of Emergency-Stop pin functionality.

SYNTAX

```
int LSX_SetDigIO_EmergencyStop (int  lLSID,
                                int  lIndex);
```

TANGO CMD !digfkt

PARAMETERS

lIndex: 0 to 15 (input/output)

EXAMPLE

```
LSX_SetDigIO_EmergencyStop(Tango1, 15); // Pin 15 is used for Emergency-Stop
```

SetDigIO_Off

NOT SUPPORTED BY TANGO Switch off digital inputs / outputs function.
(Does not affect inputs / outputs states).

SYNTAX

```
int LSX_SetDigIO_Off (int  lLSID,
                      int  lIndex);
```

TANGO CMD !digfkt

PARAMETERS **lIndex:** 0 to 15 (individual Input/Output pins), 16 (all 16 port pins)

EXAMPLE

```
LSX_SetDigIO_Off(Tango1, 0); // Function of I/O pin 0 is switched ´Off“
```

SetDigIO_Polarity

NOT SUPPORTED BY TANGO Set polarity of digital inputs / outputs.

SYNTAX

```
int LSX_SetDigIO_Polarity (int   lLSID,
                           int   lIndex,
                           BOOL  bHigh);
```

TANGO CMD !digfkt

PARAMETERS

lIndex: 0 to 15 (individual I/O pin), 16 (all 16 port pins)
bHigh TRUE = high active
FALSE = low active

EXAMPLE

```
LSX_SetDigIO_Polarity(Tango1, 3, TRUE); // input pin / output pin 3 high active
```

4.11. TVR Clock & Direction IO

GetTVRMode

Retrieve the TVR mode of the clock & direction IO

SYNTAX

```
int LSX_GetTVRMode (int  lLSID,
                    int *plValue);
```

TANGO CMD ?tvrm

PARAMETERS **plValue**: Pointer to a 32-Bit Integer, which returns the TVR mode

EXAMPLE

```
int mode;
LSX_GetTVRMode(Tango1, &mode);
```

SetTVRMode

Set the TVR mode of the clock & direction IO

SYNTAX

```
int LSX_SetTVRMode (int  lLSID,
                    int  lMode1,
                    int  lMode2,
                    int  lMode3,
                    int  lMode4);
```

TANGO CMD !tvr

PARAMETERS **lMode1...4**: Required TVR mode of the axes

- 0 = disabled
- (1) = enabled without tvrf factor
- (2) = enabled with tvrf factor
- (3) = enabled without tvrf factor, requires ext. start/stop
- (4) = enabled with tvrf factor, requires ext. start/stop
- 5 = enabled with tvrjoyf factor

EXAMPLE

```
LSX_SetTVRMode(Tango1, 0, 0, 5, 0); // only set tvr mode 5 for Z
```

GetFactorTVR

Retrieve the TVR factor for TVR modes 1-4

SYNTAX

```
int LSX_GetFactorTVR (int     lLSID,
                      double *pdVal1,
                      double *pdVal2,
                      double *pdVal3,
                      double *pdVal4);
```

TANGO CMD ?tvrf

PARAMETERS **pdVal1...4**: Pointers to double, which hold the returned TVR factors

EXAMPLE

```
double tvrf[4];
LSX_GetTvrfTVR(Tango1, &tvrf[0], &tvrf[1], &tvrf[2], &tvrf[3]);
```

SetFactorTVR

Set the TVR factor for TVR modes 1-4

SYNTAX

```
int LSX_SetFactorTVR (int     lLSID,
                      double  dVal1,
                      double  dVal2,
                      double  dVal3,
                      double  dVal4);
```

TANGO CMD !tvrf

PARAMETERS **dVal1...4**: Required TVR factor for all axes

EXAMPLE

```
LSX_SetFactorTVR(Tango1, 0.2 0.2 0.05, 50);
```

4.12. Encoder Settings

ClearEncoder

Reset encoder TTL counter to zero.

SYNTAX

```
int LSX_ClearEncoder (int  lLSID,
                      int  lAxis);
```

TANGO CMD !clearhwcount

PARAMETERS **lAxis**: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)

EXAMPLE

```
LSX_ClearEncoder(Tango1, 2); // reset encoder counter of Y-Axis to zero
```

GetEncoder

Retrieves all encoder TTL counter positions.

SYNTAX

```
int LSX_GetEncoder (int     lLSID,
                    double *pdXP,
                    double *pdYP,
                    double *pdZP,
                    double *pdAP);
```

TANGO CMD ?hwcount

PARAMETERS **pdXP...pdAP**: Counter values, 4x interpolated

EXAMPLE

```
LSX_GetEncoder(Tango1, &XP, &YP, &ZP, &AP);
```

GetEncoderActive

Retrieves which encoder will be activated after calibration.
Please note: **This function corresponds to the „?encmask“** command! Not “?enc”.

SYNTAX

```
int LSX_GetEncoderActive (int  lLSID,
                          int *plFlags);
```

TANGO CMD ?encmask

PARAMETERS **plFlags**: 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

```
LSX_GetEncoderActive(Tango1, &Flags);
```

SetEncoderActive

Sets which encoder is activated after calibration
Please note: This function corresponds to „!encmask“ command, not “!enc”.

SYNTAX

```
int LSX_SetEncoderActive (int  lLSID,
                          int  lFlags);
```

TANGO CMD !encmask

PARAMETERS **lFlags**: 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

```
LSX_SetEncoderActive(Tango1, 0); // No encoder will be used
LSX_SetEncoderActive(Tango1, 2); // Y-encoder will be activated after
calibration
```

GetEncoderMask

Retrieve status of encoders.
Please note: This function corresponds to „?enc“ command, not “?encmask”!

SYNTAX

```
int LSX_GetEncoderMask (int  lLSID,
                        int *plFlags);
```

TANGO CMD ?enc

PARAMETERS **plFlags**: 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;
LSX_GetEncoderMask(Tango1, &EncMask);
if (EncMask & 2)...
// if encoder of Y-Axis connected + active...
```

SetEncoderMask

Activates / deactivates encoders manually.
Please note: This function corresponds to „!enc“ command, not “!encmask”!
Do not use in closed loop. Encoders should always be activated with Calibrate command.

SYNTAX

```
int LSX_SetEncoderMask (int  lLSID,
                        int  lValue);
```

TANGO CMD !enc

PARAMETERS

lValue: Active encoder mask (flags)
Bit 0 = (activate)/deactivate X encoder
Bit 1 = (activate)/deactivate Y encoder
Bit 2 = (activate)/deactivate Z encoder

EXAMPLE

```
LSX_SetEncoderMask(Tango1, 0); // deactivate all encoders
LSX_SetEncoderMask(Tango1, 2); // deactivate X and Z encoders, activate
Y encoder
```

GetEncoderSingleAxis

Read the encoder settings of Encoder Type, Signal Period and Reference Signal of the specified axis.

SYNTAX

```
int LSX_GetEncoderSingleAxis (int     lLSID,
                              int    *plAxis,
                              int    *plEncoderType,
                              double *pdPeriod,
                              int    *plReference);
```

TANGO CMD ?enctype, ?encref, ?encperiod x,y,z,a

PARAMETERS

lAxis: The axis number 1...4 (corresponding to X...A)
lEncoderType: Pointer to return the encoder type
dPeriod: Pointer to return the encoder signal period length [mm]
dReference: Pointer to return the usage of encoder reference (0 or 1)

EXAMPLE

```
LSX_GetEncoderSingleAxis(Tango1, 3, &lEncType, &dPeriod, &lReference);
// 3 = Z
```

SetEncoderSingleAxis

Set the Encoder Type, Signal Period and availability of Reference Signal of the specified axis.

SYNTAX

```
int LSX_SetEncoderSingleAxis (int     lLSID,
                              int     lAxis,
                              int     lEncoderType,
                              double  dPeriod,
                              int     lReference);
```

TANGO CMD !enctype, !encref, !encperiod x,y,z,a

PARAMETERS

lAxis: The axis number 1...4 (corresponding to X...A)
lEncoderType: The encoder type (MR, 1Vpp, TTL, ...) refer to the enctype description
dPeriod: The encoder signal period length [mm]
dReference: Usage of encoder reference signal (0=no signal or 1=has a ref signal)

EXAMPLE

```
LSX_SetEncoderSingleAxis(Tango1, 3, 2, 0.02, 0); // axis 3(Z) = Type 2, 20µm, no ref
```

GetEncoderPeriod

Retrieves encoder signal period length.
A NULL pointer can be used for not required axes.

SYNTAX

```
int LSX_GetEncoderPeriod (int     lLSID,
                          double *pdX,
                          double *pdY,
                          double *pdZ,
                          double *pdA);
```

TANGO CMD ?encperiod

PARAMETERS **pdX...pdA**: Period length [mm]

EXAMPLE

```
double X,Y,Z,A;
LSX_GetEncoderPeriod(Tango1, &X, &Y, &Z, &A);
LSX_GetEncoderPeriod(Tango1, &X, &Y, NULL, NULL); // Ignore Z+A axes
```

SetEncoderPeriod

Set encoder signal period length.

SYNTAX

```
int LSX_SetEncoderPeriod (int     lLSID,
                          double  dX,
                          double  dY,
                          double  dZ,
                          double  dA);
```

TANGO CMD !encperiod

PARAMETERS **dX...dA**: 0.0001 – 4 mm

EXAMPLE

```
LSX_SetEncoderPeriod(Tango1, 0.5, 0.5, 0.5, 0.5);
// set encoder signal period of all axes to 0.5mm
```

GetEncoderPeriodSingleAxis new in 1.533

Read encoder period of a single axis. Single-axis variant of GetEncoderPeriod. EncoderPeriod is double64 on all controllers (full 15 significant-digit resolution).

SYNTAX

```
int LSX_GetEncoderPeriodSingleAxis (int     lLSID,
                                    int     lAxis,
                                    double *pdEncoderPeriod);
```

TANGO CMD ?encperiod x,y,z,a

PARAMETERS **lAxis:** 1=X, 2=Y, 3=Z, 4=A
pdEncoderPeriod: encoder period [mm]

EXAMPLE

```
LSX_GetEncoderPeriodSingleAxis(Tango1, 1, &dEncoderPeriod);
```

SetEncoderPeriodSingleAxis new in 1.533

Set encoder period of a single axis. Single-axis variant of SetEncoderPeriod.

SYNTAX

```
int LSX_SetEncoderPeriodSingleAxis (int     lLSID,
                                    int     lAxis,
                                    double  dEncoderPeriod);
```

TANGO CMD !encperiod x,y,z,a

PARAMETERS **lAxis:** 1=X, 2=Y, 3=Z, 4=A
dEncoderPeriod: encoder period [mm]

EXAMPLE

```
LSX_SetEncoderPeriodSingleAxis(Tango1, 1, 0.0001);
```

GetEncoderPosition

Retrieves position response type.

SYNTAX

```
int LSX_GetEncoderPosition (int   lLSID,
                            BOOL *pbValue);
```

TANGO CMD ?encpos

PARAMETERS **pbValue**: TRUE = axis position values will be read from the encoder, if activated.
(If no encoders are active, the position is taken from the motor)
FALSE = Position will be taken from the motor position only.

EXAMPLE

```
LSX_GetEncoderPosition(Tango1, &Value);
```

SetEncoderPosition

Select readout of encoder- or motor-related position values.

SYNTAX

```
int LSX_SetEncoderPosition (int   lLSID,
                            BOOL  bValue);
```

TANGO CMD !encpos

PARAMETERS **bValue**: TRUE = axis position values will be read from the encoder, if activated.
(If no encoders are active, the position is taken from the motor)
FALSE = Position will be taken from the motor position.

EXAMPLE

```
LSX_SetEncoderPosition(Tango1, TRUE);
```

GetEncoderRefSignal

Retrieves whether the encoder reference signal is used when calibrating.

SYNTAX

```
int LSX_GetEncoderRefSignal (int  lLSID,
                             int *plXR,
                             int *plYR,
                             int *plZR,
                             int *plAR);
```

TANGO CMD ?encref

PARAMETERS **plXR...plAR**: 0 = encoder reference signal is evaluated while calibrating
1 = reference signal is not evaluated, zero position is set at the CAL end switch

EXAMPLE

```
LSX_GetEncoderRefSignal(Tango1, &X, &Y, &Z, &A);
```

SetEncoderRefSignal

Use reference signal from encoder when calibrating.

SYNTAX

```
int LSX_SetEncoderRefSignal (int  lLSID,
                             int  lXR,
                             int  lYR,
                             int  lZR,
                             int  lAR);
```

TANGO CMD !encref

PARAMETERS **lXR...lAR**: 0 = encoder reference signal is evaluated while calibrating
1 = reference signal is not evaluated, zero position is set at the CAL end switch

EXAMPLE

```
LSX_SetEncoderRefSignal(Tango1, 1, 1, 0, 0);
// when calibrating, reference signals of encoders X and Y are evaluated
```

GetRefSpeed

Old method for reading the velocity for calibrating to the encoder reference mark or to the reference switch. It is recommended to use ?encrefvel.

SYNTAX

```
int LSX_GetRefSpeed (int  lLSID,
                     int *plSpeed);
```

TANGO CMD ?calrefspeed

PARAMETERS **plSpeed:** Pointer to int for returning the velocity value (in 1/100 rev/s, one for all axes)

EXAMPLE

```
LSX_GetRefSpeed (Tango1, &Speed);
```

SetRefSpeed

Old method of setting the velocity for calibrating to the encoder reference mark or to the reference switch. It is recommended to use !encrefvel.

SYNTAX

```
int LSX_SetRefSpeed (int  lLSID,
                     int  lSpeed);
```

TANGO CMD !calrefspeed

PARAMETERS **lSpeed:** Velocity in 1/100 motor revolutions/s, one velocity applies to all axes

EXAMPLE

```
LSX_SetRefSpeed (Tango1, 50); // search the reference switch (or the reference mark) search velocity of all axes to 50/100 motor revolutions/s (=0.5 rev/s)
```

4.13. Closed Loop Settings

GetController

Retrieve Closed Loop mode.

SYNTAX

```
int LSX_GetController (int  lLSID,
                       int *plXC,
                       int *plYC,
                       int *plZC,
                       int *plRC);
```

TANGO CMD ?ctr

PARAMETERS **plXC...plRC**: Closed Loop Mode
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

```
LSX_GetController(Tango1, &X, &Y, &Z, &A);
```

SetController

Set Closed Loop mode.

SYNTAX

```
int LSX_SetController (int  lLSID,
                       int  lXC,
                       int  lYC,
                       int  lZC,
                       int  lAC);
```

TANGO CMD !ctr

PARAMETERS **lXC...lAC**: Closed Loop Mode
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

```
LSX_SetController(Tango1, 2, 2, 0, 0); // Enable permanent closed loop for X and Y
```

GetControllerCall

Read Closed Loop interval time.
Should remain at the default setting (3 or 5 ms).

SYNTAX
```
int LSX_GetControllerCall (int  lLSID,
                           int *plCtrCall);
```

TANGO CMD ?ctrc

PARAMETER: ***CtrCall***: Controller call time [ms]
(default 3 or 5ms, depending on the TANGO type)

EXAMPLE
```
LSX_GetControllerCall(Tango1, &CtrCall);
```

SetControllerCall

Set Closed Loop interval time.
Applies to all axes. The interval, in which the closed loop processes the deviation.
Should remain at the controller default setting (3 or 5 ms).

SYNTAX
```
int LSX_SetControllerCall (int  lLSID,
                           int  lCtrCall);
```

TANGO CMD !ctrc

PARAMETERS **lCtrCall**: Controller call time [ms]

EXAMPLE
```
LSX_SetControllerCall(Tango1, 5);
// CtrCall = 5 means: Closed Loop controller is called every 5
milliseconds
```

GetControllerFactor

FOR COMPATIBILITY ONLY! Retrieve Closed Loop controller factors.
Note: The TANGO supports 2 factors per axis (idle & move),
therefore it is highly recommended to **use GetControllerFactorSingleAxis()!**

SYNTAX
```
int LSX_GetControllerFactor (int     lLSID,
                             double *pdX,
                             double *pdY,
                             double *pdZ,
                             double *pdA);
```

TANGO CMD ?ctrf

PARAMETERS **pdX...pdA**: Closed Loop factors

EXAMPLE
```
LSX_GetControllerFactor(Tango1, &X, &Y, &Z, &A);
```

SetControllerFactor

FOR COMPATIBILITY ONLY!

Set Closed Loop controller factor.
Note: The TANGO supports 2 factors per axis (idle & move),
therefore it is highly recommended to **use SetControllerFactorSingleAxis()!**

SYNTAX

```
int LSX_SetControllerFactor (int     lLSID,
                             double  dX,
                             double  dY,
                             double  dZ,
                             double  dA);
```

TANGO CMD !ctrf

PARAMETERS **dX...dA**: Position difference amplification factor 1 – 64

EXAMPLE

```
LSX_SetControllerFactor(Tango1, 2, 2, 2, 0);
// Closed Loop amplification is set to 2 for X, Y and Z axes
```

GetControllerFactorSingleAxis

Retrieve Closed Loop controller factors of the specified axis.
Note: The TANGO supports 2 factors per axis (idle & move)

SYNTAX

```
int LSX_GetControllerFactorSingleAxis (int     lLSID,
                                       int     lAxis,
                                       double *pdFidle,
                                       double *dFmove);
```

TANGO CMD ?ctrff x,y,z,a

PARAMETERS

lAxis = *Axis number 1,2,3,4 (for axes X,Y,Z,A)*
dFidle = *returns 0.0 ...25.4*
dFmove = *returns 0.0 ...25.4*

EXAMPLE

```
LSX_GetControllerFactorSingleAxis(Tango1, 2, &idlefactor,
&movefactor); // Read Y
```

SetControllerFactorSingleAxis

Set Closed Loop controller factor.
The TANGO supports 2 factors per axis (idle & move).

SYNTAX

```
int LSX_SetControllerFactorSingleAxis (int     lLSID,
                                       int     lAxis,
                                       double  dFidle,
                                       double  dFmove);
```

TANGO CMD !ctrff x,y,z,a

PARAMETERS

lAxis = *Axis number 1,2,3,4 (for axes X,Y,Z,A)*
dFidle = *0.0 ...25.4*
dFmove = *0.0 ...25.4*

EXAMPLE

```
LSX_SetControllerFactorSingleAxis(Tango1, 3, 8, 2.5, 0);
// Set Closed Loop amplification for Z to 8 at idle and 2.5 at move
```

GetControllerSteps

Retrieves length of controller steps.
The TANGO uses ctrs as the “Lock-In Range”, where after exceeding a certain behavior can be specified with ctrfm.

SYNTAX
```
int LSX_GetControllerSteps (int     lLSID,
                            double *pdX,
                            double *pdY,
                            double *pdZ,
                            double *pdA);
```

TANGO CMD ?ctrs

PARAMETERS **pdX...pdA**: Lock-In Range [mm]

EXAMPLE
```
LSX_GetControllerSteps(Tango1, &X, &Y, &Z, &A);
```

SetControllerSteps

Set controller steps.
The TANGO uses ctrs as the “Lock-In Range”, where after exceeding a certain behavior can be specified with ctrfm.

SYNTAX
```
int LSX_SetControllerSteps (int     lLSID,
                            double  dX,
                            double  dY,
                            double  dZ,
                            double  dA);
```

TANGO CMD !ctrs

PARAMETERS **dX...dA**: Lock-In Range, >= 0.01 [mm]

EXAMPLE
```
LSX_SetControllerSteps(Tango1, 0.2, 0.2, 0.05, 1);
```

GetControllerTimeout

Read the closed loop control timeout. The maximum wait for TWI timeout.

SYNTAX
```
int LSX_GetControllerTimeout (int  lLSID,
                              int *plACtrTimeout);
```

TANGO CMD ?ctrt

PARAMETERS **plACtrTimeout:** 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
```
LSX_GetControllerTimeout(Tango1, &ActrTimeout);
```

SetControllerTimeout

Set the closed loop control timeout. The maximum wait for TWI timeout.

SYNTAX

```
int LSX_SetControllerTimeout (int  lLSID,
                              int  lACtrTimeout);
```

TANGO CMD *!ctrt*

PARAMETERS ***lACtrTimeout***: 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

```
LSX_SetControllerTimeout(Tango1, 500);
// Abort after trying to settle in the target window for 500ms
```

GetControllerTWDelaySingleAxis

Read controller delay. Time to remain in TWI until “reached” state is assigned.
For individual axes.

SYNTAX

```
int LSX_GetControllerTWDelaySingleAxis (int  lLSID,
                                        int  lAxis,
                                        int *plCtrTWDelay);
```

TANGO CMD ?ctrd x,y,z,a

PARAMETERS

Axis: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
CtrTWDelay: Controller delay [ms]

EXAMPLE

```
LSX_GetControllerTWDelaySingleAxis (Tango1, 3, &CtrTWDelay); // Read Z
```

SetControllerTWDelaySingleAxis

Set controller delay. Time to remain in TWI until “reached” state is assigned.
For individual axes.

SYNTAX

```
int LSX_SetControllerTWDelaySingleAxis (int  lLSID,
                                        int  lAxis,
                                        int  lCtrTWDelay);
```

TANGO CMD !ctrd x,y,z,a

PARAMETERS

lAxis: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
lCtrTWDelay: Controller delay 0 – 250 ms
Time for which the axis must remain in the target window.
Moves are delayed by at least this time.

EXAMPLE

```
LSX_SetControllerTWDelaySingleAxis(Tango1, 3, 0);
// controller delay in Z switched off, closed loop end position will be
inaccurate
```

GetControllerTWDelay

FOR COMPATIBILITY ONLY!

Read controller delay.
Time to remain in TWI until "reached" state is assigned.
OLD Function! As the TANGO allows to set ctrd for individual axes,
use GetControllerTWDelaySingleAxis() or ctrd by SendString.

SYNTAX
```
int LSX_GetControllerTWDelay (int  lLSID,
                               int *plCtrTWDelay);
```

TANGO CMD ?ctrd

PARAMETERS **plCtrTWDelay**: Controller delay [ms]

EXAMPLE
```
LSX_GetControllerTWDelay(Tango1, &CtrTWDelay);
```

SetControllerTWDelay

FOR COMPATIBILITY ONLY!

Set controller delay.
Time to remain in TWI until "reached" state is assigned.
OLD Function! As the TANGO allows to set ctrd for individual axes,
use SetControllerTWDelaySingleAxis() or ctrd by SendString.

SYNTAX
```
int LSX_SetControllerTWDelay (int  lLSID,
                               int  lCtrTWDelay);
```

TANGO CMD !ctrd

PARAMETERS **lCtrTWDelay**: 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
```
LSX_SetControllerTWDelay(Tango1, 0);
// controller delay switched off, closed loop end position will be
inaccurate
```

GetTargetWindow

Retrieves closed loop target windows of all axes.

SYNTAX
```
int LSX_GetTargetWindow (int     lLSID,
                         double *pdX,
                         double *pdY,
                         double *pdZ,
                         double *pdA);
```

TANGO CMD ?twi

PARAMETERS **pdX...pdA**: Target window, depends on selected dimension

EXAMPLE
```
LSX_GetTargetWindow(Tango1, &X, &Y, &Z, &A);
```

SetTargetWindow

Set closed loop controller target windows.
The closed loop controller has to settle within ± this window size for the specified delay time.

SYNTAX

```
int LSX_SetTargetWindow (int     lLSID,
                         double  dX,
                         double  dY,
                         double  dZ,
                         double  dA);
```

TANGO CMD !twi

PARAMETERS

dX...dA: closed loop target windows
1 – 25000 (motor increments)
0.1 – 1000 (µm)
0.0001 – 1 (mm)
(values depend on dimension)

EXAMPLE

```
LSX_SetTargetWindow(Tango1, 1.0, 0.001, 0.001, 0.0005);
```

GetTargetWindowSingleAxis new in 1.533

Read target window of a single axis. Single-axis variant of GetTargetWindow.

SYNTAX

```
int LSX_GetTargetWindowSingleAxis (int     lLSID,
                                   int     lAxis,
                                   double *pdTargetWindow);
```

TANGO CMD ?twi x,y,z,a

PARAMETERS

lAxis: 1=X, 2=Y, 3=Z, 4=A
pdTargetWindow: target window, unit depends on dimension

EXAMPLE

```
LSX_GetTargetWindowSingleAxis(Tango1, 1, &dTargetWindow);
```

SetTargetWindowSingleAxis new in 1.533

Set target window of a single axis. Single-axis variant of SetTargetWindow.

SYNTAX

```
int LSX_SetTargetWindowSingleAxis (int     lLSID,
                                   int     lAxis,
                                   double  dTargetWindow);
```

TANGO CMD !twi x,y,z,a

PARAMETERS

lAxis: 1=X, 2=Y, 3=Z, 4=A
dTargetWindow: target window, unit depends on dimension

EXAMPLE

```
LSX_SetTargetWindowSingleAxis(Tango1, 1, 0.001);
```

SetCtrFastMoveOff

Not supported by TANGO.
FastMove function deactivated.

SYNTAX
```
int LSX_SetCtrFastMoveOff (int  lLSID);
```

TANGO CMD !ctrfm

PARAMETERS -

EXAMPLE
```
LSX_SetCtrFastMoveOff(Tango1);
```

SetCtrFastMoveOn

Not supported by TANGO.
Activate FastMove function, meaning a new vector is started if controller position difference is larger than the lock-in range.

SYNTAX
```
int LSX_SetCtrFastMoveOn (int  lLSID);
```

TANGO CMD !ctrfm

PARAMETERS -

EXAMPLE
```
LSX_SetCtrFastMoveOn(Tango1);
```

GetCtrFastMove

Not supported by TANGO.
Retrieves setting of FastMove function.

SYNTAX
```
int LSX_GetCtrFastMove (int   lLSID,
                        BOOL *pbActive);
```

TANGO CMD ?ctrfm

PARAMETERS **pbActive**: TRUE = FastMove function active

EXAMPLE
```
LSX_GetCtrFastMove(Tango1, &Active);
```

GetCtrFastMoveCounter

Not supported by TANGO.

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.

SYNTAX

```
int LSX_GetCtrFastMoveCounter (int  lLSID,
                               int *plXC,
                               int *plYC,
                               int *plZC,
                               int *plAC);
```

TANGO CMD ?ctrfmc

PARAMETERS **plXC...plAC**: Number of carried out Fast Move functions

EXAMPLE

```
LSX_GetCtrFastMoveCounter(Tango1, &XC, &YC,&ZC,&AC);
```

ClearCtrFastMoveCounter

Not supported by TANGO.

If position difference is larger than lock-in range, a new vector will be started and corresponding counter will be increased by one.

SYNTAX

```
int LSX_ClearCtrFastMoveCounter (int  lLSID);
```

TANGO CMD !ctrfmc

PARAMETERS -

EXAMPLE

```
LSX_ClearCtrFastMoveCounter(Tango1);
```

4.14. Trigger Output

GetTrigger

Retrieve trigger globally enable setting.

SYNTAX
```
int LSX_GetTrigger (int   lLSID,
                    BOOL *pbATrigger);
```

TANGO CMD ?trig

PARAMETERS **pbATrigger:** TRUE = trigger is on (enabled)
FALSE = trigger is off (disabled)

EXAMPLE
```
LSX_GetTrigger(Tango1, &Atrigger);
```

SetTrigger

Globally enable or disable the trigger (switch the trigger functionality ON / OFF).

SYNTAX
```
int LSX_SetTrigger (int   lLSID,
                    BOOL  bATrigger);
```

TANGO CMD !trig

PARAMETERS **bATrigger** TRUE = switch trigger on (enabled)
FALSE = switch trigger off (disabled)

EXAMPLE
```
LSX_SetTrigger(Tango1, TRUE); // Globally enable the trigger (also
defines start pos.)
```

GetTriggerMode

Retrieve trigger mode.

SYNTAX
```
int LSX_GetTriggerMode (int  lLSID,
                        int *plMode);
```

TANGO CMD ?trigm

PARAMETERS **lMode:** Pointer to variable where the mode should be returned to

EXAMPLE
```
LSX_GetTriggerMode(Tango1, &Mode);
```

SetTriggerMode

Select trigger mode.

Refer to the TANGO Instruction Set documentation for available modes.

SYNTAX

```
int LSX_SetTriggerMode (int  lLSID,
                        int  lMode);
```

TANGO CMD !trigm

PARAMETERS **lMode** Desired Trigger Mode

EXAMPLE

```
LSX_SetTriggerMode(Tango1, 5); // Set Trigger Mode to 5
```

GetTriggerPar

Retrieves trigger parameters.

SYNTAX

```
int LSX_GetTriggerPar (int     lLSID,
                       int    *plAxis,
                       int    *plMode,
                       int    *plSignal,
                       double *pdDistance);
```

TANGO CMD ?triga

PARAMETERS

plAxis: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
plMode: Trigger mode (see command !trigm)
plSignal: aTrigger signal (see command !trigs)
pdDistance: Trigger distance (see command !trigd)

EXAMPLE

```
LSX_GetTriggerPar(Tango1, &Axis, &Mode, & Signal, &Distance);
```

SetTriggerPar

Set trigger parameters. Sets several parameters at once.

SYNTAX

```
int LSX_SetTriggerPar (int     lLSID,
                       int     lAxis,
                       int     lMode,
                       int     lSignal,
                       double  dDistance);
```

TANGO CMD !triga

PARAMETERS

lAxis: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
lMode: Trigger mode (see command !trigm)
lSignal: Trigger signal (see command !trigs)
dDistance: Trigger distance (see command !trigd)

EXAMPLE

```
LSX_SetTriggerPar(Tango1, 1, 3, 2, 5.0);
```

GetTrigCount

Retrieve trigger counter value.

SYNTAX
```
int LSX_GetTrigCount (int  lLSID,
                      int *plValue);
```

TANGO CMD ?trigcount

PARAMETERS **plValue**: Number of executed triggers

EXAMPLE
```
LSX_GetTrigCount(Tango1, &Value);
```

SetTrigCount

Set trigger counter value.

SYNTAX
```
int LSX_SetTrigCount (int  lLSID,
                      int  lValue);
```

TANGO CMD !trigcount

PARAMETERS **lValue**: 0 to 2147483647

EXAMPLE
```
LSX_SetTrigCount(Tango1, 0); // Clear trigger counter
```

GetTriggerAxis

Read the selected axis for position trigger.

SYNTAX
```
int LSX_GetTriggerAxis (int  lLSID,
                        int *plAxis);
```

TANGO CMD ?triga

PARAMETERS **plAxis**: Axis for position-dependent trigger modes
1, 2, 3, 4 (corresponding to X, Y, Z, A axes)

EXAMPLE
```
LSX_GetTriggerAxis(Tango1, &Axis);
```

SetTriggerAxis

Set the axis for position-dependent triggering.

SYNTAX
```
int LSX_SetTriggerAxis (int  lLSID,
                        int  lAxis);
```

TANGO CMD !triga

PARAMETERS **lAxis**: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)

EXAMPLE
```
LSX_SetTriggerAxis(Tango1, 2); // Trigger uses position of Y-axis (2)
```

GetTriggerSignalLength

Read the trigger output signal length / pulse width.

SYNTAX
```
int LSX_GetTriggerSignalLength (int  lLSID,
                                int *plLength);
```

TANGO CMD ?trigs

PARAMETERS **plLength**: Trigger pulse width 0 to 2500000 [µs]

EXAMPLE
```
LSX_GetTriggerSignalLength(Tango1, &Length);
```

SetTriggerSignalLength

Set the trigger output signal length / pulse width.

SYNTAX
```
int LSX_SetTriggerSignalLength (int  lLSID,
                                int  lLength);
```

TANGO CMD !trigs

PARAMETERS **lLength**: Trigger pulse width 0 to 2500000 [µs] in increments of 40 µs (0,40,80,...)

EXAMPLE
```
LSX_SetTriggerSignalLength Tango1, 120); // Trigger signal is 120µs
long
```

GetTriggerDistance

Read the position interval for trigger pulses.

SYNTAX
```
int LSX_GetTriggerDistance (int     lLSID,
                            double *pdDistance);
```

TANGO CMD ?trigd

PARAMETERS **pdDistance**: >0.0 to 5000000 [position unit depends on Dimension]

EXAMPLE
```
LSX_GetTriggerDistance(Tango1, &Distance);
```

SetTriggerDistance

Set the position interval for trigger pulses.

SYNTAX
```
int LSX_SetTriggerDistance (int     lLSID,
                            double  dDistance);
```

TANGO CMD !trigd

PARAMETERS **dDistance**: >0.0 to 5000000 [position unit depends on Dimension]

EXAMPLE
```
LSX_SetTriggerDistance(Tango1, 12.5); // Set the position interval to
12.5
```

GetTriggerCompensation

Read the trigger look-ahead timing compensation.
Compensates for delays within the trigger chain by looking ahead.
Useful for bidirectional scanning in order to compensate comb effect of the lines.

SYNTAX
```
int LSX_GetTriggerCompensation (int  lLSID,
                                int *plTime);
```

TANGO CMD ?trigcomp

PARAMETERS **plTime**: -10000... +10000 [µs] in increments of 10µs (0,10,20,...)

EXAMPLE
```
LSX_GetTriggerCompensation(Tango1, &Time);
```

SetTriggerCompensation

Set the trigger look-ahead timing compensation.
Compensates for delays within the trigger chain by looking ahead.
Useful for bidirectional scanning in order to compensate comb effect of the lines.

SYNTAX
```
int LSX_SetTriggerCompensation (int  lLSID,
                                int  lTime);
```

TANGO CMD !trigcomp

PARAMETERS **lTime**: -10000... +10000 [µs] in increments of 10µs (0,10,20,...)

EXAMPLE
```
LSX_SetTriggerCompensation(Tango1, 40); // Trigger compensation looks 40µs ahead
```

GetTriggerEncoder

Read if the trigger unit uses the encoder position.

SYNTAX
```
int LSX_GetTriggerEncoder (int  lLSID,
                           int *plEncoder);
```

TANGO CMD ?trigenc

PARAMETERS **plEncoder**: 0 = trigger on motor position, 1 = trigger on encoder position

EXAMPLE
```
LSX_GetTriggerEncoder(Tango1, &Encoder);
```

SetTriggerEncoder

Select trigger on encoder (1) or motor (0) position.

SYNTAX
```
int LSX_SetTriggerEncoder (int  lLSID,
                           int  lEncoder);
```

TANGO CMD !trigenc

PARAMETERS **lEncoder**: 0 = trigger on motor position, 1 = trigger on encoder position

EXAMPLE
```
LSX_SetTriggerEncoder(Tango1, 0); // Trigger on motor position
```

GetTriggerFrequency

Read the trigger output frequency for trigger modes 100, 101.

SYNTAX
```
int LSX_GetTriggerFrequency (int     lLSID,
                             double *pdFrequency);
```

TANGO CMD ?trigf

PARAMETERS **pdFrequency**: 0.01 to 25000 Hz

EXAMPLE
```
LSX_GetTriggerFrequency(Tango1, &Frequency);
```

SetTriggerFrequency

Set the trigger output frequency for trigger modes 100, 101.

SYNTAX
```
int LSX_SetTriggerFrequency (int     lLSID,
                             double  dFrequency);
```

TANGO CMD !trigf

PARAMETERS **dFrequency**: 0.01 to 25000 Hz

EXAMPLE
```
LSX_SetTriggerFrequency(Tango1, 1000); // Set the trigger frequency to
1kHz
```

GetTriggerOutput

Read the trigger output setting.

SYNTAX
```
int LSX_GetTriggerOutput (int  lLSID,
                          int *plValue);
```

TANGO CMD ?trigo

PARAMETERS **plValue**: 0 = no output used, 1=TRIGGER_OUT, and further combinations (refer to trigo)

EXAMPLE
```
LSX_GetTriggerOutput(Tango1, &Value);
```

SetTriggerOutput

Select trigger output setting. Select output 1 and/or 2 and output mode.

SYNTAX

```
int LSX_SetTriggerOutput (int  lLSID,
                          int  lValue);
```

TANGO CMD !trigo

PARAMETERS **lValue**: 0 = no output used, 1=TRIGGER_OUT,
and further combinations (refer to trigo)

Output / Mode	STANDARD	PREC.WIDTH2	PREC.DELAY2	PREC.FREQUENCY2
No output No signal out	**0**	(4)	(8 PCI-E)	(12)
Primary TRIGGER OUT	**1**	(5)	(9 PCI-E)	(13)
Secondary ** TAKT OUT	**2**	**6**	**10** (PCI-E)	**14**
Both, P&S ** TRIGGER+TAKT	**3**	**7**	**11** (PCI-E)	**15**

EXAMPLE

```
LSX_SetTriggerOutput(Tango1, 1); // Use trigger output 1 (TRIGGER_OUT) only
```

Get2ndTriggerDelay

Read the precise edge delay of the 2nd output related to the trigger output.

SYNTAX

```
int LSX_Get2ndTriggerDelay (int     lLSID,
                            double *pdValue);
```

TANGO CMD ?trigbdelay

PARAMETERS **pdValue**: 0.00 to 32500000 [µs], internal resolution is 1/132µs

EXAMPLE

```
LSX_Get2ndTriggerDelay(Tango1, &Value);
```

Set2ndTriggerDelay

Set the precise edge delay of the 2nd output related to the trigger output.

SYNTAX

```
int LSX_Set2ndTriggerDelay (int     lLSID,
                            double  dValue);
```

TANGO CMD !trigbdelay

PARAMETERS **dValue**: 0.00 to 32500000 [µs], internal resolution is 1/132µs

EXAMPLE

```
LSX_Set2ndTriggerDelay(Tango1, 0.05); // Set the 2nd output delay to 50ns
```

Get2ndTriggerWidth

Read the precise pulse width of the 2nd trigger output signal.

SYNTAX
```
int LSX_Get2ndTriggerWidth (int     lLSID,
                            double *pdValue);
```

TANGO CMD ?trigbwidth

PARAMETERS **pdValue**: 0.00 to 32500000 [µs], internal resolution is 1/132µs

EXAMPLE
```
LSX_Get2ndTriggerWidth(Tango1, &Value);
```

Set2ndTriggerWidth

Set the precise pulse width of the 2nd trigger output signal.

SYNTAX
```
int LSX_Set2ndTriggerWidth (int     lLSID,
                            double  dValue);
```

TANGO CMD !trigbwidth

PARAMETERS **dValue**: 0.00 to 32500000 [µs], internal resolution is 1/132µs

EXAMPLE
```
LSX_Set2ndTriggerWidth(Tango1, 1.05); // Set the 2nd output pulse width
to 1.05µs
```

Get2ndTriggerFrequency

Read the precise frequency of the 2nd trigger output signal.

SYNTAX
```
int LSX_Get2ndTriggerFrequency (int     lLSID,
                                double *pdValue);
```

TANGO CMD ?trigbf

PARAMETERS **pdValue**: 0.010... 66000000 Hz

EXAMPLE
```
LSX_Get2ndTriggerFrequency(Tango1, &Value);
```

Set2ndTriggerFrequency

Set the precise frequency of the 2nd trigger output signal.

SYNTAX
```
int LSX_Set2ndTriggerFrequency (int     lLSID,
                                double  dValue);
```

TANGO CMD !trigbf

PARAMETERS **dValue**: 0.010... 66000000 Hz

EXAMPLE
```
LSX_Set2ndTriggerFrequency(Tango1, 1000000); // Set the 2nd out
frequency to 1MHz
```

GetTriggerRange

Read back the trigger range settings which were set by LSX_SetTriggerRange.

SYNTAX

```
int LSX_GetTriggerRange (int     lLSID,
                         double *pdStartPos,
                         double *pdEndPos,
                         int    *plNumberOfTriggerPulses);
```

TANGO CMD ?trigr

PARAMETERS

Axis: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
pdStartPos: Position where the triggering starts (first pulse)
pdEndPos: Position where the trigger ends (last pulse)
plNumberOfTriggerPulses:
To achieve the required trigger distance (interval).
It must be considered that the first pulse is before the first interval, so N+1 pulses must be set

EXAMPLE

```
LSX_GetTriggerRange(Tango1, &StartPos, &EndPos,
&NumberOfTriggerPulses);
```

SetTriggerRange

Set the trigger range, which defines a trigger start position, end position and the number of triggers from start to end. The unit (µm,mm,...) depends on dimension.
Info: SetTriggerRange creates an internal equidistant position list, so the TriggerPositionList functions can also be used with TriggerRange, if required.
To define the triggered axis, use SetTriggerAxis once before using TriggerRange(s).

SYNTAX

```
int LSX_SetTriggerRange (int     lLSID,
                         double  dStartPos,
                         double  dEndPos,
                         int     lNumberOfTriggerPulses);
```

TANGO CMD !trigr

PARAMETERS

Axis: 1, 2, 3, 4 (corresponding to X, Y, Z, A axes)
dStartPos: Position where the triggering starts (first pulse)
dEndPos: Position where the trigger ends (last pulse)
lNumberOfTriggerPulses:
To achieve the required trigger distance (interval).
It must be considered that the first pulse is before the first interval, so N+1 pulses must be set

EXAMPLE

```
LSX_SetTriggerRange(Tango1, 10.5, 20.5 101); // Set 101 pulses from
10.5 to 20.5mm
// which is the 1st pulse at 10.5mm plus 100 pulses until and including
20.5mm that lead to a (20.5-10.5)/100 = 0.1mm distance between the
pulses
```

GetTriggerPositionList

Read back the trigger position list from SetTriggerPositionList.
This can also read back the internal list set by SetTriggerRange.

SYNTAX

```
int LSX_GetTriggerPositionList (int     lLSID,
                                int     lIndex,
                                double *pdPos);
```

TANGO CMD ?trigp

PARAMETERS

lIndex: 1... GetTriggerPositionListEntries
pdPos: Position list entry of the selected trigger axis (depends on Dimension)

EXAMPLE

```
LSX_GetTriggerPositionList(Tango1, &Index, &Pos);
```

SetTriggerPositionList

Create a trigger position list with individual positions, e.g. not equidistant.
The positions can be individual but must be either constantly increasing or decreasing.

SYNTAX

```
int LSX_SetTriggerPositionList (int     lLSID,
                                int     lIndex,
                                double  dPosition);
```

TANGO CMD !trigp

PARAMETERS

lIndex: 1...MAX_ENTRIES
Pos: Position list entry for the selected trigger axis (depends on Dimension)

EXAMPLE

```
LSX_SetTriggerPositionList(Tango1, 1, 10.5);    // Set first pulse at
10.5mm
LSX_SetTriggerPositionList(Tango1, 2, 17.5031); // Set 2nd pulse at
17.5031mm
LSX_SetTriggerPositionList(Tango1, 3, 51.8774); // Set 3rd pulse at
51.8774mm
```

GetTriggerPositionListIndex

Read at which entry (index) the position list currently is.
This also reads back the current index in case of SetTriggerRange.

SYNTAX

```
int LSX_GetTriggerPositionListIndex (int  lLSID,
                                     int *plIndex);
```

TANGO CMD ?trigi

PARAMETERS

plIndex: 1...N

EXAMPLE

```
LSX_GetTriggerPositionListIndex(Tango1, &Index);
```

SetTriggerPositionListIndex

Manipulate the position list index for the next trigger pulse.
Usually, the trigger goes forward through the position list, but by manipulating the current list index the trigger can skip some positions by manipulating the index forward.

SYNTAX
```
int LSX_SetTriggerPositionListIndex (int  lLSID,
                                     int  lIndex);
```

TANGO CMD !trigi

PARAMETERS **lIndex**: 1...N

EXAMPLE
```
LSX_SetTriggerPositionListIndex(Tango1, 17);  // Set the next trigger
position at list entry number 17 (to skip the trigger positions in
between)
```

GetTriggerPositionListEntries

Read the number of entries in the trigger position list.
This also is the index of the last entry in the trigger position list: 1...Entries
And can be used e.g. to append positions at Index "entries+1" etc.

SYNTAX
```
int LSX_GetTriggerPositionListEntries (int  lLSID,
                                       int *plNumberOfEntries);
```

TANGO CMD ?trigc

PARAMETERS **plNumberOfEntries**: 0...N

EXAMPLE
```
LSX_GetTriggerPositionListEntries(Tango1, &NumberOfEntries);
```

SetTriggerPositionListEntries

Manipulate the amount of position list entries.
The number 0 can be used to clear the entire position list (to allow entering a new list).

SYNTAX
```
int LSX_SetTriggerPositionListEntries (int  lLSID,
                                       int  lNumberOfEntries);
```

TANGO CMD !trigc

PARAMETERS **lNumberOfEntries**: 0, 1...N

EXAMPLE
```
LSX_SetTriggerPositionListEntries(Tango1, 5); // Reduce the number of
entries to 5
LSX_SetTriggerPositionListEntries(Tango1, 0); // Delete the entire
trigger position list
```

GetTriggerLevel

Read the trigger level used exclusively by trigger modes 20 and 21.
All other modes specify their level (pulse polarity) by the mode itself, e.g. 100, 101.

SYNTAX
```
int LSX_GetTriggerLevel (int  lLSID,
                         int *plLevel);
```

TANGO CMD ?trigl

PARAMETERS **plLevel**: 0 = active low, 1 = active high trigger pulse

EXAMPLE
```
LSX_GetTriggerLevel(Tango1, &Level);
```

SetTriggerLevel

Set the trigger level used exclusively by TriggerRange and PositionList.
All other modes specify their level (pulse polarity) by the mode itself, e.g. 100, 101.

SYNTAX
```
int LSX_SetTriggerLevel (int  lLSID,
                         int  lLevel);
```

TANGO CMD !trigl

PARAMETERS **lLevel**: 0 = active low, 1 = active high trigger pulse

EXAMPLE
```
LSX_SetTriggerLevel(Tango1, 0); // Set trigger to active low (0)
```

ForceTriggerPulse

Fire the trigger signal manually. The trigger must be enabled and set to manual mode.

The pulse length is set by SetTriggerSignalLength and:

- A call with 0 forces a single trigger pulse, returns immediately
- A call with 1 forces a single trigger pulse, returns with the frequency (period) of SetTriggerFrequency
- 2 to 127 generate pulse sequences at the trigger frequency of SetTriggerFrequency and returns after all pulses have been sent

SYNTAX
```
int LSX_ForceTriggerPulse (int  lLSID,
                           int  lNumberOfPulses);
```

TANGO CMD !trigger

PARAMETERS **lNumberOfPulses**: 0 = one single pulse, returns immediately.
1 = one single pulse as well, but the call returns only after one period of the trigger frequency of SetTriggerFrequency.
2 ... 127 = that number of pulses at the trigger frequency of SetTriggerFrequency; the call returns when all pulses have been sent.
Any other value returns 4002.

EXAMPLE
```
LSX_ForceTriggerPulse(Tango1, 5); // Fire 5 pulses at the configured
frequency
```

4.15. Snapshot Input

GetSnapshot

Retrieves current Snapshot state, if it is ON/enabled or OFF/disabled.

SYNTAX
```
int LSX_GetSnapshot (int   lLSID,
                     BOOL *pbASnapshot);
```

TANGO CMD ?sns

PARAMETERS **pbASnapshot:** TRUE = Snapshot is “On” (enabled)
FALSE = Snapshot is “Off” (disabled)

EXAMPLE
```
LSX_GetSnapshot(Tango1, &Asnapshot);
```

SetSnapshot

Switch Snapshot functionality ON or OFF.

SYNTAX
```
int LSX_SetSnapshot (int   lLSID,
                     BOOL  bASnapshot);
```

TANGO CMD !sns

PARAMETERS **bASnapshot**: TRUE = switch Snapshot “On” (enable)
FALSE = switch Snapshot “Off” (disable)

EXAMPLE
```
LSX_SetSnapshot(Tango1, TRUE); // Globally enable the snapshot
functionality
```

GetSnapshotMode

Retrieves the current Snapshot mode.

SYNTAX
```
int LSX_GetSnapshotMode (int  lLSID,
                         int *plMode);
```

TANGO CMD ?snsm

PARAMETERS **plMode:** 0-12 (refer to snsm documentation in TANGO Instruction Set)

EXAMPLE
```
LSX_GetSnapshotMode(Tango1, &Mode);
```

SetSnapshotMode

Sets the Snapshot mode/functionality.

SYNTAX
```
int LSX_SetSnapshotMode (int  lLSID,
                         int  lMode);
```

TANGO CMD !snsm

PARAMETERS **lMode**: 0-12 (refer to snsm documentation in TANGO Instruction Set)

EXAMPLE
```
LSX_SetSnapshotMode(Tango1, 0); // Set mode to 0 = capture positions @
HDI F2 key
```

GetSnapshotCount

Snapshot counter. It counts the snapshot events
= number of captured positions / entries in the position array (see SnapshotPosArray).

SYNTAX
```
int LSX_GetSnapshotCount (int  lLSID,
                          int *plSnsCount);
```

TANGO CMD ?snsc

PARAMETERS **plSnsCount:** Amount of captured Snapshots (= available position array entries)

EXAMPLE
```
LSX_GetSnapshotCount(Tango1, &SnsCount);
```

SetSnapshotCount

Manipulate Snapshot counter (captured positions), truncate position array entries.

SYNTAX
```
int LSX_SetSnapshotCount (int  lLSID,
                          int  lSnsCount);
```

TANGO CMD !snsc

PARAMETERS **lSnsCount**: Amount of available position array entries

EXAMPLE
```
LSX_SetSnapshotCount(Tango1, 5); // Truncate position array to 5
entries.
```

GetSnapshotFilter

Retrieve input filter times for signal chatter.

SYNTAX
```
int LSX_GetSnapshotFilter (int  lLSID,
                           int *plTime);
```

TANGO CMD ?snsf

PARAMETERS **plTime**: Filter time [ms]

EXAMPLE
```
LSX_GetSnapshotFilter(Tango1, &Time);
```

SetSnapshotFilter

Set input debounce filter.
If a mechanical switch is connected, a typical debounce time is around 10ms.
As the debounce filter slows down the processing speed (interval), for a true digital signal (which does not bounce), the filter might be set to 0 ms.

SYNTAX

```
int LSX_SetSnapshotFilter (int  lLSID,
                           int  lTime);
```

TANGO CMD !snsf

PARAMETERS **lTime**: Filter time, within 0-100 ms

EXAMPLE

```
LSX_SetSnapshotFilter(Tango1, 0); // no filter, fast response (e.g. for TTL signals)
```

GetSnapshotPar

Retrieve Snapshot parameters.
Does not support reading of higher snapshot modes! Only 0 or "not 0".
→ Use GetSnapShotMode instead.

SYNTAX

```
int LSX_GetSnapshotPar (int   lLSID,
                        BOOL *pbHigh,
                        BOOL *pbAutoMode);
```

TANGO CMD ?snsl

PARAMETERS

pbHigh: TRUE = snapshot is high active
FALSE = snapshot is low active

pbAutoMode: TRUE = snapshot „Automatic": Position is automatically moved to after first snapshot pulse (corresponds to SnapshotMode 1)
FALSE = snapshot capture (corresponds to SnapshotMode 0)

EXAMPLE

```
LSX_GetSnapshotPar(Tango1, &High, &AutoMode);
```

SetSnapshotPar

Set Snapshot parameters (polarity and only snapshot mode 0 or 1: !snsl + !snsm 0/1).
The AutoMode might interfere with a previously set SnapshotMode, if that was set to a mode higher than 1) Does not support setting of higher snapshot modes! Only 0 or 1.
→ Use SetSnapShotMode instead.

SYNTAX

```
int LSX_SetSnapshotPar (int   lLSID,
                        BOOL  bHigh,
                        BOOL  bAutoMode);
```

TANGO CMD !snsl

PARAMETERS

bHigh: TRUE = snapshot is high active
FALSE = snapshot is low active

bAutoMode: TRUE = snapshot „Automatic“: Position is automatically moved to after first snapshot pulse (corresponds to SnapshotMode 1)
FALSE = snapshot capture (corresponds to SnapshotMode 0)

EXAMPLE

```
LSX_SetSnapshotPar(Tango1, TRUE, FALSE);
```

GetSnapshotPos

Retrieve position that was captured on the most recent Snapshot event.

SYNTAX

```
int LSX_GetSnapshotPos (int    lLSID,
                        double *pdX,
                        double *pdY,
                        double *pdZ,
                        double *pdA);
```

TANGO CMD ?snsp

PARAMETERS **pdX...pdA**: Position values

EXAMPLE

```
LSX_GetSnapshotPos(Tango1, &X, &Y, &Z, &A);
```

GetSnapshotPosArray

Retrieve Snapshot position from Array.

SYNTAX

```
int LSX_GetSnapshotPosArray (int     lLSID,
                             int     lIndex,
                             double *pdX,
                             double *pdY,
                             double *pdZ,
                             double *pdA);
```

TANGO CMD ?snsa

PARAMETERS

lIndex: Index of snapshot positions (from =1 to SnapshotCount, max. entries is 1024)

X, Y, Z, A: Position values

EXAMPLE

```
LSX_GetSnapshotPosArray(Tango1, 2, &X, &Y, &Z, &A);
// 2 = Read positions captured on the second snapshot event (second array entry)
```

SetSnapshotPosArray

Set, append or change entries of the position array.

SYNTAX

```
int LSX_SetSnapshotPosArray (int     lLSID,
                             int     lIndex,
                             double  dX,
                             double  dY,
                             double  dZ,
                             double  dA);
```

TANGO CMD !snsa

PARAMETERS

lIndex: Index of snapshot positions (1-1024)
Index must be within the number of existing entries (or one above to append)
appending is also possible by using Index = -1, which is easier to handle

X, Y, Z, A: Position values

EXAMPLE

```
LSX_SetSnapshotPosArray(Tango1, -1, 0.55, 2.4, 0.0, 0.0);
// Append a position array entry by software
```

ClearSnapshotPosArray

Deletes the entire position array, clears all entries
and checks if the array was cleared by ?snsc == 0.

SYNTAX
```
int LSX_ClearSnapshotPosArray (int  lLSID);
```

TANGO CMD !snsa

PARAMETERS -

EXAMPLE
```
LSX_ClearSnapshotPosArray(Tango1); // Delete the entire PosArray
```

GetSnapshotIndex

Read the current Snapshot index, e.g. to identify where it is in “Automatic” mode.
Remarks: The index goes from 0 to SnapshotCount-1, so index “0” is PosArray(1).

SYNTAX
```
int LSX_GetSnapshotIndex (int  lLSID,
                          int *plSnsIndex);
```

TANGO CMD ?snsi

PARAMETERS **plSnsIndex:** Current position of the index pointer within the position array

EXAMPLE
```
LSX_GetSnapshotIndex(Tango1, &SnsIndex);
```

SetSnapshotIndex

Manipulate Snapshot index: set index to a different position array entry
Remarks: The index goes from 0 to SnapshotCount-1, so index “0” is PosArray(1).

SYNTAX
```
int LSX_SetSnapshotIndex (int  lLSID,
                          int  lSnsIndex);
```

TANGO CMD !snsi

PARAMETERS **lSnsIndex**: Required position of the index pointer within the PosArray,
e.g. for SnapshotMode “Automatic”

EXAMPLE
```
LSX_SetSnapshotIndex(Tango1, 5); // Set pointer to Index 5
```

5. SlideExpress Functions

This chapter describes additional DLL functions for the SlideExpress. From application point of view there are only few differences between previous top loader and new front loader systems SlideExpress (1) and SlideExpress 2.

Constant Name	Meaning	Top Loader	Front Loader
MAXMAGA	number of magazines	4	3
MAXROW	number of rows	50	30
MAXCOL	Number of columns	4	4

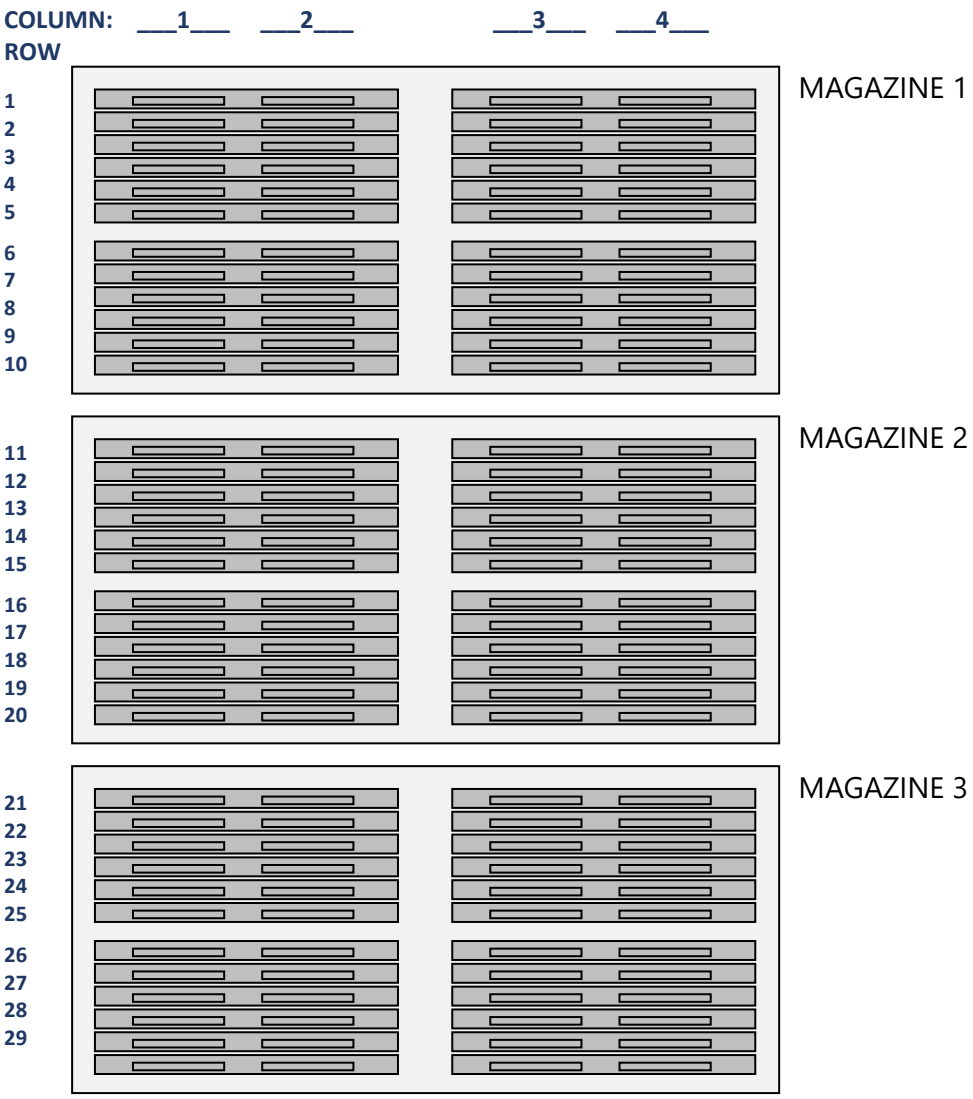
SlideExpress 2: Organization of slides in rows and columns

Despite the organization of **clips and magazines**, the SlideExpress instructions still handle slides as **rows and columns**.

For the new SlideExpress 2, there are 3 magazines 1,2,3 stacked over each other.

Each magazine has 10 rows with two clip columns, where a clip usually contains 2 slides.

This leads to the following organization:



Eject

Move magazine(s) to a position that allows user access.

SYNTAX

```
int LSX_Eject (int  lLSID,
               int  maga,
               int  keep);
```

TANGO CMD !eject

PARAMETERS

maga = magazine number [1..MAXMAGA]
keep = 0 to empty gripper before eject magazine(s) or
1 to keep slide(s) in gripper

EXAMPLE

```
LSX_Eject(Tango1, 1, 0);
```

Insert

Magazine(s) are inserted and tested if seated and which slides are present.
This function is precondition to use SlideSeated() and MagazinSeated().

SYNTAX

```
int LSX_Insert (int  lLSID);
```

TANGO CMD !insert

PARAMETERS -

EXAMPLE

```
LSX_Insert(Tango1);
```

SlideSeated

Query if slide is present or not or unknown.

SYNTAX

```
int LSX_SlideSeated (int  lLSID,
                     int  col,
                     int  row,
                     int *status);
```

TANGO CMD ?insert

PARAMETERS

col = col number [1..MAXCOL]
row = row number [1..MAXROW]
status = returns slide status (-1 = unknown, 0 = empty, 1 = seated)

EXAMPLE

```
LSX_SlideSeated(Tango1, 4, 30, &status);
```

MagazinSeated

Query if magazine is present or not or unknown.

SYNTAX

```
int LSX_MagazinSeated (int  lLSID,
                       int  maga,
                       int *status);
```

TANGO CMD ?insert

PARAMETERS

maga = magazine number [1..MAXMAGA]
status = returns magazine status (-1 = unknown, 0 = empty, 1 = seated)

EXAMPLE

```
LSX_MagazinSeated(Tango1, 1, &status); // check if magazine 1 is seated
```

GetGripper

Query gripper status information. Returns status of gripper 1 and 2.
Status information like unknown, empty or origin of slide in gripper.

SYNTAX

```
int LSX_GetGripper (int  lLSID,
                    int *c1,
                    int *r1,
                    int *c2,
                    int *r2);
```

TANGO CMD ?gripper

PARAMETERS

c1 = column number [-1, 0, 1..MAXCOL] of slide 1 in gripper
r1 = row number [-1, 0, 1..MAXROW] of slide 1 in gripper
c2 = column number [-1, 0, 1..MAXCOL] of slide 2 in gripper
r2 = row number [-1, 0, 1..MAXROW] of slide 2 in gripper

EXAMPLE

```
LSX_GetGripper(Tango1, &c1, &r1, &c2, &r2); // check status of gripper
1 and 2
c1, c2     =  -1 = unknown, 0 = empty or 1 to 4 for magazine number
r1, r2     =  -1 = unknown, 0 = empty or 1 to 50 for slot number
c1=1, r1=0        indicates priority slide 1 in gripper
                  (obsolete for front loader)
c2=1, r2=0        indicates priority slide 2 in gripper
                  (obsolete for front loader)
```

SetGripper

Set gripper forces an overwrite of the gripper status.
Usually the status represents empty, unknown or slide/tray origin state.
The status is managed internally and shall only be manipulated to solve unknown gripper state after power loss or to manipulate the origin where it comes from and so where it will be returned to, e.g. for custom sorting.

SYNTAX

```
int LSX_SetGripper (int  lLSID,
                    int  c1,
                    int  r1,
                    int  c2,
                    int  r2);
```

TANGO CMD !gripper

PARAMETERS

c1 = column number [-1, 0, 1..MAXCOL] of slide 1 in gripper
r1 = row number [-1, 0, 1..MAXROW] of slide 1 in gripper
c2 = column number [-1, 0, 1..MAXCOL] of slide 2 in gripper
r2 = row number [-1, 0, 1..MAXROW] of slide 2 in gripper

EXAMPLE

```
LSX_SetGripper(Tango1, 0, 0, 0, 0); // set gripper to “empty”
```

GetClipType

GetClipType returns the clip type that is currently in the gripper.
For SlideExpress handling system.

SYNTAX

```
int LSX_GetClipType (int  lLSID,
                     int *plClipType);
```

TANGO CMD ?cliptype

PARAMETERS **plClipType** = pointer to int, returns the clip type

EXAMPLE

```
LSX_GetClipType(Tango1, plClipType);
```

GetSlide

Get slide(s) from addressed position in magazine.

SYNTAX

```
int LSX_GetSlide (int  lLSID,
                  int  col,
                  int  row,
                  int  mode);
```

TANGO CMD !getslide

PARAMETERS

col = column number [1..MAXCOL]
row = row number [1..MAXROW]
mode = (0 = inspection, 1 = bar code reader, 2 = liquid dispenser “oiler”)

EXAMPLE

```
LSX_GetSlide(Tango1, 1, 1, 0);
```

PutSlide

Put slide(s) back to addressed position in magazine ~~or priority handler~~.
To return the slide to its original position (where it was taken from by GetSlide),
set both col and row parameters to 0.

SYNTAX
```
int LSX_PutSlide (int  lLSID,
                  int  col,
                  int  row);
```

TANGO CMD !putslide

PARAMETERS
col = column [1..MAXCOL]
row = slot number [1..MAXROW] (obsolete: ~~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 (if any).

EXAMPLE
```
LSX_PutSlide(Tango1, 4, 20); // put slide to magazine 4 slot 20.
LSX_PutSlide(Tango1, 0, 0);  // put slide back to where it was taken
from.
```

Obsolete:

GetPrioHandlerPos

Read actual priority handler position.

SYNTAX
```
int LSX_GetPrioHandlerPos (int  lLSID,
                           int *php);
```

TANGO CMD ?priohand

PARAMETERS
php = return value of actual priority handler position
(55 = unknown, 0 = middle, -1 = shift in, 1 = pulled out)

EXAMPLE
```
LSX_GetPrioHandlerPos(Tango1, &php);
```

Obsolete:

SetPrioHandlerPos

Enables user to shift priority handler to required position.
Handler is locked at destination or after 30s timeout.

SYNTAX
```
int LSX_SetPrioHandlerPos (int  lLSID,
                           int  php);
```

TANGO CMD !priohand

PARAMETERS
php = specify destination 0 = middle, -1 = shift in, 1 = pulled out

EXAMPLE
```
LSX_SetPrioHandlerPos (Tango1, 1); //enable user to pull out priority
handler
```

6. TrayExpress Functions

This chapter describes optional DLL functions to be used in conjunction for TrayExpress.

Eject

Eject magazine.
The TrayExpress moves magazine downwards and opens front cover to allow user operations like removing trays or loading trays.

SYNTAX

```
int LSX_Eject (int  lLSID,
               int  maga,
               int  keep);
```

TANGO CMD !eject

PARAMETERS

maga = magazine number [1] (currently only 1 allowed)
keep = 0 to empty gripper before eject magazine, or
1 to keep tray in gripper

EXAMPLE

```
LSX_Eject(Tango1, 1, 0);
```

Insert

Front Cover is closed and magazine is inserted and tested if seated and which trays are present. This function is precondition to use SlideSeated() and MagazinSeated().

SYNTAX

```
int LSX_Insert (int  lLSID);
```

TANGO CMD !insert

PARAMETERS -

EXAMPLE

```
LSX_Insert(Tango1);
```

SlideSeated

Query if tray is present or not or unknown.

SYNTAX
```
int LSX_SlideSeated (int  lLSID,
                     int  maga,
                     int  slot,
                     int *status);
```

TANGO CMD ?insert

PARAMETERS
maga = magazine number [1]
slot = slot number [1..50]
status = returns slide status (-1 = unknown, 0 = empty, 1 = seated)

EXAMPLE
```
LSX_SlideSeated(Tango1, 1, 1, &status);
```

MagazinSeated

Query if magazine is present or not or unknown.

SYNTAX
```
int LSX_MagazinSeated (int  lLSID,
                       int  maga,
                       int *status);
```

TANGO CMD ?insert

PARAMETERS
maga = magazine number [1]
status = returns magazine status (-1 = unknown, 0 = empty, 1 = seated)

EXAMPLE
```
LSX_MagazinSeated Tango1, 1, &status); // check if magazine 1 is seated
```

GetGripper

Query gripper status information. Returns status of gripper.

SYNTAX
```
int LSX_GetGripper (int  lLSID,
                    int *c1,
                    int *s1,
                    int *c2,
                    int *s2);
```

TANGO CMD ?gripper

PARAMETERS
c1 = magazine number [-1, 0, 1..4] of slide in gripper
s1 = slot number [-1, 0, 1..24] of slide in gripper
c2 = dummy for compatibility with slide express
s2 = dummy for compatibility with slide express

EXAMPLE
```
LSX_GetGripper(Tango1, &c1, &s1, &c2, &s2); //check status of gripper 1
and 2
c1 =  -1 = unknown, 0 = empty or 1 (magazine number)
s1 =  -1 = unknown, 0 = empty or 1 to 24 for slot number
```

SetGripper

Set gripper status information.
The status is managed internally and shall only be manipulated in case of solving gripper states after power loss or for tray sorting tasks.

SYNTAX

```
int LSX_SetGripper (int  lLSID,
                    int  c1,
                    int  s1,
                    int  c2,
                    int  s2);
```

TANGO CMD !gripper

PARAMETERS

c1 = magazine number [-1, 0, 1..4] of slide in gripper
s1 = slot number [-1, 0, 1..50] of slide in gripper
c2 = dummy for compatibility with slide express
s2 = dummy for compatibility with slide express

EXAMPLE

```
LSX_SetGripper(Tango1, 0, 0, 0, 0); // set gripper to “empty”
```

GetTray

Get tray from the specified magazine position.

SYNTAX

```
int LSX_GetTray (int  lLSID,
                 int  slot,
                 int  mode);
```

TANGO CMD !gettray

PARAMETERS

slot = slot number [1..35*]
mode = place tray for [0=microscope, 1=liquid dispenser, 2=bar code reader *]
* The maximum slot number and availability of modes depends on hardware

EXAMPLE

```
LSX_GetTray(Tango1, 2, 0); // get tray from slot 2 and put it under the
microscope
```

PutTray

Put tray back to the specified magazine position.
or to the position where it came from = by setting slot = 0.

SYNTAX

```
int LSX_PutTray (int  lLSID,
                 int  slot);
```

TANGO CMD !puttray

PARAMETERS

slot = slot number [1..35*] or 0 to return it back to the original position of GetTray

* The maximum slot number (24, 35, ...) depends on hardware

EXAMPLE

```
LSX_PutTray(Tango1, 10); // put tray to magazine slot 10
LSX_PutTray(Tango1, 0);   // put tray back to where it was taken from by GetTray
```

GetRFID

Get RFID of addressed tray, when tray is properly seated in magazine

SYNTAX

```
int LSX_GetRFID (int  lLSID,
                 int  slot,
                 int  bank,
                 int *plRFID);
```

TANGO CMD ?rfid

PARAMETERS

slot = slot number [1..MAXSLOT]
bank = bank number [0 to 64]
plRFID = pointer to int returns data stored in RFID transponder device

EXAMPLE

```
LSX_GetRFID(Tango1, 1, 0, plRFID);
```

SetRFID

Store RFID data into addressed tray

SYNTAX

```
int LSX_SetRFID (int  lLSID,
                 int  slot,
                 int  bank,
                 int  rfdata);
```

TANGO CMD !rfid

PARAMETERS

slot = slot number [1..MAXSLOT]
bank = bank number [2 to 64] (bank 0 and 1 are not writeable)
rfdata = int contains customer data to be coded into RFID transponder device

EXAMPLE

```
LSX_SetRFID(Tango1, 1, 0, rfdata);
```

GetNumberOfSlots

Get number of available slots per magazine

SYNTAX
```
int LSX_GetNumberOfSlots (int  lLSID,
                          int *plSlots);
```

TANGO CMD ?separ 15

PARAMETERS **plSlots** = returns number of slots per magazine

EXAMPLE
```
LSX_GetNumberOfSlots(Tango1, plSlots);
```

GetNumberOfMagazins

Get number of available magazines.
Returns always 1 and is available for compatibility to SlideExpress only.

SYNTAX
```
int LSX_GetNumberOfMagazins (int  lLSID,
                             int *plMagazines);
```

TANGO CMD ?maxmaga

PARAMETERS **plMagazines** = pointer to int returns number [1]

EXAMPLE
```
LSX_GetNumberOfMagazins(Tango1, plMagazines);
```

7. Additional Handling System Functions

Additional commands for SlideExpress, TrayExpress and other handling systems.

GetLoaderType

Get loader type
Response depends on system configuration.

SYNTAX
```
int LSX_GetLoaderType (int  lLSID,
                       int *plLoaderType);
```

TANGO CMD ?loadertype

PARAMETERS **plLoaderType** = pointer to int returns loader type
1 = SlideExpress
2 = Custom handling system: standalone base unit
3 = Custom handling system: loader master base unit
4 = Custom handling system: loader slave (magazine)

EXAMPLE
```
LSX_GetLoaderType(Tango1, plLoaderType);
```

GetNumberOfRows

Get number of magazine rows, e.g. max. number of slots to insert trays.
Response is number of magazine rows.

SYNTAX
```
int LSX_GetNumberOfRows (int  lLSID,
                         int *plRows);
```

TANGO CMD ?separ 15

PARAMETERS **plRows** = pointer to int returns number of available slots or magazine rows

EXAMPLE
```
LSX_GetNumberOfRows(Tango1, plRows);
```

GetNumberOfColumns

Get number of magazine columns, e.g. max number of slide sensors per slot/tray
.
Response is number of magazine columns.

SYNTAX
```
int LSX_GetNumberOfColumns (int  lLSID,
                            int *plCols);
```

TANGO CMD ?separ 16

PARAMETERS **plCols** = pointer to int returns number of magazine column
(6 manual, 6 for loader system)

EXAMPLE
```
LSX_GetNumberOfColumns(Tango1, plCols);
```

GetTraySN

Get tray SN returns unique tray RFID serial number of addressed slot / tray.
For TrayExpress and custom handling system.

SYNTAX
```
int LSX_GetTraySN (int  lLSID,
                   int  slot,
                   int *plTraySN);
```

TANGO CMD ?traysn

PARAMETERS **plTraySN** = pointer to int returns unique tray RFID serial number

EXAMPLE
```
LSX_GetTraySN(Tango1, 1, plTraySN);
```

GetTrayType

GetTrayType returns tray type of addressed tray.
For TrayExpress and custom handling system.

SYNTAX
```
int LSX_GetTrayType (int  lLSID,
                     int  slot,
                     int *plTrayType);
```

TANGO CMD ?traytype

PARAMETERS **plTrayType** = pointer to int returns tray type (user coded data)

EXAMPLE
```
LSX_GetTrayType(Tango1, 1, plTrayType);
```

SetTrayType

SetTrayType stores tray type into RFID transponder of addressed slot / tray.
For TrayExpress and custom handling system, **only used for factory programming**.

SYNTAX
```
int LSX_SetTrayType (int  lLSID,
                     int  slot,
                     int  aTrayType);
```

TANGO CMD !traytype

PARAMETERS **aTrayType** = int data contains information of required tray type

EXAMPLE
```
int aTrayType = 0x0100010a; //see customer specification requirements
for explanation
LSX_SetTrayType(Tango1, 1, aTrayType);
```

SetCabinLED

SetCabinLED on or off.
For handling systems with internal illumination.

SYNTAX
```
int LSX_SetCabinLED (int  lLSID,
                     int  lOn);
```

TANGO CMD !cabinled

PARAMETERS **lOn** = 0 to switch OFF or 1 to switch ON the loader illumination

EXAMPLE
```
LSX_SetCabinLED(Tango1, 1); // switch ON illumination
LSX_SetCabinLED(Tango1, 0); // switch OFF
```

GetCabinLED

GetCabinLED returns actual state of cabin illumination, on or off.
For handling systems with internal illumination.

SYNTAX
```
int LSX_GetCabinLED (int  lLSID,
                     int *plState);
```

TANGO CMD ?cabinled

PARAMETERS **plState** = Pointer to int returns illumination state 0 (off) or 1 (on)

EXAMPLE
```
LSX_GetCabinLED(Tango1, plState);
```

SetLabelLED

SetLabelLED on or off.
For handling systems with barcode illumination.

SYNTAX
```
int LSX_SetLabelLED (int  lLSID,
                     int  lOn);
```

TANGO CMD !labelled

PARAMETERS **lOn** = 0 to switch OFF or 1 to switch ON the label illumination

EXAMPLE
```
LSX_SetLabelLED(Tango1, 1); // switch ON illumination
LSX_SetLabelLED(Tango1, 0); // switch OFF
```

GetLabelLED

GetLabelLED returns actual state of label illumination.
For handling systems with barcode illumination.

SYNTAX

```
int LSX_GetLabelLED (int  lLSID,
                     int *plState);
```

TANGO CMD ?labelled

PARAMETERS **plState** = pointer to int returns illumination state (0=off, 1=on)

EXAMPLE

```
LSX_GetLabelLED(Tango1, plState);
```

8. POS3 Auxiliary Axes

Additional commands for the 3 auxiliary axes of the optional xPos / POS3 module.

Xpos3_GetPosSingleAxis

Read axis position from an xPos axis

SYNTAX

```
int LSX_Xpos3_GetPosSingleAxis (int     lLSID,
                                int     lAxis,
                                double *plPos);
```

TANGO CMD ?xp

PARAMETERS

lAxis = xPos axis 1, 2 or 3
lPos = variable to return the position to

EXAMPLE

```
LSX_Xpos3_GetPosSingleAxis(Tango1, 2, &Pos); // Read Position of the 2nd xPos axis
```

Xpos3_SetPosSingleAxis

Set/Change axis position of an xPos axis

SYNTAX

```
int LSX_Xpos3_SetPosSingleAxis (int     lLSID,
                                int     lAxis,
                                double  lPos);
```

TANGO CMD !xp

PARAMETERS

lAxis = xPos axis 1, 2 or 3
lPos = desired position value

EXAMPLE

```
LSX_Xpos3_SetPosSingleAxis(Tango1, 3, 12.345); // Set xPos 3rd axis position to 12.345
```

Xpos3_MoveAbsSingleAxis

Absolute Move for an xPos axis

SYNTAX

```
int LSX_Xpos3_MoveAbsSingleAxis (int                    lLSID,
                                 int                    lAxis,
                                 double lTargetPos BOOL bWait);
```

TANGO CMD !xma

PARAMETERS

lAxis = xPos axis 1, 2 or 3
lTargetPos = Absolute Position value to move to
bWait = function shall return after reaching position (= TRUE)
or directly after sending the command (= FALSE)

EXAMPLE

```
LSX_Xpos3_MoveAbsSingleAxis (Tango1, 2, 10.5, FALSE); // Move 2nd xPos axis
```

Xpos3_MoveRelSingleAxis

Relative Move for an xPos axis

SYNTAX

```
int LSX_Xpos3_MoveRelSingleAxis (int    lLSID,
                                 int    lAxis,
                                 double lDistance BOOL bWait);
```

TANGO CMD !xmr

PARAMETERS

lAxis = xPos axis 1, 2 or 3
lPos = desired position value
bWait = function shall return after reaching position (= TRUE)
or directly after sending the command (= FALSE)

EXAMPLE

```
LSX_Xpos3_MoveRelSingleAxis(Tango1, 3, -0.5, FALSE); // Move 3rd xPos axis 0.5mm backwards
```

Xpos3_GetVelSingleAxis new in 1.533

Read the move velocity of an xPos axis. Velocity used for all xPos move commands [mm/s].

SYNTAX

```
int LSX_Xpos3_GetVelSingleAxis (int    lLSID,
                                int    lAxis,
                                double *pdVel);
```

TANGO CMD ?xv

PARAMETERS

lAxis = xPos axis (1, 2 or 3)
pdVel = move velocity [mm/s]

EXAMPLE

```
LSX_Xpos3_GetVelSingleAxis(Tango1, 1, &dVel);
```

Xpos3_SetVelSingleAxis new in 1.533

Set the move velocity of an xPos axis. Velocity used for all xPos move commands [mm/s].

SYNTAX

```
int LSX_Xpos3_SetVelSingleAxis (int    lLSID,
                                int    lAxis,
                                double  dVel);
```

TANGO CMD !xv

PARAMETERS

lAxis = xPos axis (1, 2 or 3)
dVel = move velocity [mm/s]

EXAMPLE

```
LSX_Xpos3_SetVelSingleAxis(Tango1, 1, 2.5);
```

Xpos3_GetSpeedSingleAxis new in 1.533

Read the current speed-move velocity of an xPos axis [mm/s].

SYNTAX

```
int LSX_Xpos3_GetSpeedSingleAxis (int     lLSID,
                                  int     lAxis,
                                  double *pdSpeed);
```

TANGO CMD ?xsp

PARAMETERS

lAxis = xPos axis (1, 2 or 3)
pdSpeed = speed-move velocity [mm/s]

EXAMPLE

```
LSX_Xpos3_GetSpeedSingleAxis(Tango1, 1, &dSpeed);
```

Xpos3_SetSpeedSingleAxis new in 1.533

Start a constant-velocity (speed) move of an xPos axis. A value of 0 stops the speed move [mm/s].

SYNTAX

```
int LSX_Xpos3_SetSpeedSingleAxis (int     lLSID,
                                  int     lAxis,
                                  double  dSpeed);
```

TANGO CMD !xsp

PARAMETERS

lAxis = xPos axis (1, 2 or 3)
dSpeed = speed-move velocity [mm/s] (0 = stop)

EXAMPLE

```
LSX_Xpos3_SetSpeedSingleAxis(Tango1, 1, 0.05);
```

Xpos3_GetAccelSingleAxis new in 1.533

Read the acceleration of an xPos axis. Used for all move, speed, cal and rm commands [m/s2].

SYNTAX

```
int LSX_Xpos3_GetAccelSingleAxis (int     lLSID,
                                  int     lAxis,
                                  double *pdAccel);
```

TANGO CMD ?xacc

PARAMETERS

lAxis = xPos axis (1, 2 or 3)
pdAccel = acceleration [m/s2]

EXAMPLE

```
LSX_Xpos3_GetAccelSingleAxis(Tango1, 1, &dAccel);
```

Xpos3_SetAccelSingleAxis new in 1.533

Set the acceleration of an xPos axis. Used for all move, speed, cal and rm commands [m/s2].

SYNTAX

```
int LSX_Xpos3_SetAccelSingleAxis (int     lLSID,
                                  int     lAxis,
                                  double  dAccel);
```

TANGO CMD !xacc

PARAMETERS

lAxis = xPos axis (1, 2 or 3)
dAccel = acceleration [m/s2]

EXAMPLE

```
LSX_Xpos3_SetAccelSingleAxis(Tango1, 1, 0.5);
```

Xpos3_GetStopAccelSingleAxis new in 1.533

Read the emergency-stop deceleration of an xPos axis [m/s2].

SYNTAX

```
int LSX_Xpos3_GetStopAccelSingleAxis (int     lLSID,
                                      int     lAxis,
                                      double *pdStopAccel);
```

TANGO CMD ?xstopacc

PARAMETERS

lAxis = xPos axis (1, 2 or 3)
pdStopAccel = emergency-stop deceleration [m/s2]

EXAMPLE

```
LSX_Xpos3_GetStopAccelSingleAxis(Tango1, 1, &dStopAccel);
```

Xpos3_SetStopAccelSingleAxis new in 1.533

Set the emergency-stop deceleration of an xPos axis. Used on stop events and limit-switch runs [m/s2].

SYNTAX

```
int LSX_Xpos3_SetStopAccelSingleAxis (int     lLSID,
                                      int     lAxis,
                                      double  dStopAccel);
```

TANGO CMD !xstopacc

PARAMETERS

lAxis = xPos axis (1, 2 or 3)
dStopAccel = emergency-stop deceleration [m/s2]

EXAMPLE

```
LSX_Xpos3_SetStopAccelSingleAxis(Tango1, 1, 1.5);
```

Xpos3_GetPitchSingleAxis new in 1.533

Read the lead-screw pitch of an xPos axis [mm/rev].

SYNTAX

```
int LSX_Xpos3_GetPitchSingleAxis (int     lLSID,
                                  int     lAxis,
                                  double *pdPitch);
```

TANGO CMD ?xpitch

PARAMETERS

lAxis = xPos axis (1, 2 or 3)
pdPitch = lead-screw pitch [mm/rev]

EXAMPLE

```
LSX_Xpos3_GetPitchSingleAxis(Tango1, 1, &dPitch);
```

Xpos3_SetPitchSingleAxis new in 1.533

Set the lead-screw pitch of an xPos axis [mm/rev].

SYNTAX

```
int LSX_Xpos3_SetPitchSingleAxis (int     lLSID,
                                  int     lAxis,
                                  double  dPitch);
```

TANGO CMD !xpitch

PARAMETERS

lAxis = xPos axis (1, 2 or 3)
dPitch = lead-screw pitch [mm/rev]

EXAMPLE

```
LSX_Xpos3_SetPitchSingleAxis(Tango1, 1, 4.0);
```

Xpos3_GetGearSingleAxis new in 1.533

Read the gear ratio of an xPos axis.

SYNTAX

```
int LSX_Xpos3_GetGearSingleAxis (int     lLSID,
                                 int     lAxis,
                                 double *pdGear);
```

TANGO CMD ?xgear

PARAMETERS

lAxis = xPos axis (1, 2 or 3)
pdGear = gear ratio

EXAMPLE

```
LSX_Xpos3_GetGearSingleAxis(Tango1, 1, &dGear);
```

Xpos3_SetGearSingleAxis new in 1.533

Set the gear ratio of an xPos axis.

SYNTAX

```
int LSX_Xpos3_SetGearSingleAxis (int     lLSID,
                                 int     lAxis,
                                 double  dGear);
```

TANGO CMD !xgear

PARAMETERS

lAxis = xPos axis (1, 2 or 3)
dGear = gear ratio

EXAMPLE

```
LSX_Xpos3_SetGearSingleAxis(Tango1, 1, 1.0);
```

Xpos3_GetAxisDirSingleAxis new in 1.533

Read the axis direction of an xPos axis. 0 = normal, 1 = reversed.

SYNTAX

```
int LSX_Xpos3_GetAxisDirSingleAxis (int  lLSID,
                                    int  lAxis,
                                    int *plAxisDir);
```

TANGO CMD ?xaxdir

PARAMETERS

lAxis = xPos axis (1, 2 or 3)
plAxisDir = axis direction (0 = normal, 1 = reversed)

EXAMPLE

```
LSX_Xpos3_GetAxisDirSingleAxis(Tango1, 1, &lAxisDir);
```

Xpos3_SetAxisDirSingleAxis new in 1.533

Set the axis direction of an xPos axis. 0 = normal, 1 = reversed.

SYNTAX

```
int LSX_Xpos3_SetAxisDirSingleAxis (int  lLSID,
                                    int  lAxis,
                                    int  lAxisDir);
```

TANGO CMD !xaxdir

PARAMETERS

lAxis = xPos axis (1, 2 or 3)
lAxisDir = axis direction (0 = normal, 1 = reversed)

EXAMPLE

```
LSX_Xpos3_SetAxisDirSingleAxis(Tango1, 1, 1);
```

Xpos3_AbortSingleAxis new in 1.533

Abort a running move on one xPos axis, or on all axes when called without an axis. Deceleration uses the xstopacc setting.

SYNTAX
```
int LSX_Xpos3_AbortSingleAxis (int lLSID, int lAxis);
```

TANGO CMD !xa

PARAMETERS
lAxis = xPos axis (1, 2 or 3)

EXAMPLE
```
LSX_Xpos3_AbortSingleAxis(Tango1, 2);
```

Xpos3_GetStatusAxisSingleAxis new in 1.533

Read the state of a single xPos axis. Returns the ASCII state character as a code (64='@' idle/ready, 77='M' moving, 83='S' limit switch or stop active, 69='E' error, 45='-' axis not available).

SYNTAX
```
int LSX_Xpos3_GetStatusAxisSingleAxis (int  lLSID,
                                       int  lAxis,
                                       int *plState);
```

TANGO CMD ?xsa

PARAMETERS
lAxis = xPos axis (1, 2 or 3)
plState = axis state code: 64='@' idle, 77='M' moving, 83='S' switch/stop, 69='E' error, 45='-' n/a

EXAMPLE
```
LSX_Xpos3_GetStatusAxisSingleAxis(Tango1, 1, &lState);
```

Xpos3_GetStatusBits new in 1.533

Read the 32 internal status bits of the xPos module, packed MSB-first as byte1<<24 | byte2<<16 | byte3<<8 | byte4 (temperature / supply-voltage / amplifier errors, axis-busy, cal/rm-done, limit switches).

SYNTAX
```
int LSX_Xpos3_GetStatusBits (int lLSID, int *plValue);
```

TANGO CMD ?xstb

PARAMETERS
plValue = variable to return the 32 packed status bits to

EXAMPLE
```
LSX_Xpos3_GetStatusBits(Tango1, &Bits);
```

Xpos3_GetTemp new in 1.533

Read the xPos module board temperature in degrees Celsius.

SYNTAX `int LSX_Xpos3_GetTemp (int lLSID, double *pdValue);`

TANGO CMD ?xtemp

PARAMETERS **pdValue** = variable to return the board temperature to [degC]

EXAMPLE
```
LSX_Xpos3_GetTemp(Tango1, &Temp);
```

Xpos3_GetTempMCU new in 1.533

Read the xPos module CPU temperature in degrees Celsius.

SYNTAX `int LSX_Xpos3_GetTempMCU (int lLSID, double *pdValue);`

TANGO CMD ?xtempmcu

PARAMETERS **pdValue** = variable to return the CPU temperature to [degC]

EXAMPLE
```
LSX_Xpos3_GetTempMCU(Tango1, &Temp);
```

Xpos3_GetError new in 1.533

Read the xPos module error number; 0 means no error. Refer to the xPos error-number list.

SYNTAX `int LSX_Xpos3_GetError (int lLSID, int *plValue);`

TANGO CMD ?xerr

PARAMETERS **plValue** = variable to return the xPos error number to (0 = no error)

EXAMPLE
```
LSX_Xpos3_GetError(Tango1, &Err);
```

Xpos3_ClearError new in 1.533

Clear a non-permanent xPos module error state.

SYNTAX `int LSX_Xpos3_ClearError (int lLSID);`

TANGO CMD !xerr

PARAMETERS

EXAMPLE
```
LSX_Xpos3_ClearError(Tango1);
```

9. Error Codes

9.1. TANGO Errors

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	! or ? is missing or not allowed
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 joystick enabled
12	limit switch actuated
13	function not executable, because encoder detected
14	error during calibration (limit switch not released)
15	error during calibration (opposing limit switch actuated)
21	multiple axis moves are forbidden (e.g. during initialization)
22	automatic or manual move is not allowed (e.g. door open or initialization)
27	emergency STOP is active
29	servo amplifiers are disabled (switched OFF)
30	safety circuit out of order
32	move discarded target outside limit
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, deviation or enc. Error
75	could not enable axis correction, or axis correction was disabled
76	io extension error (output overload on IO1 or Multi-IO connector)
77	io/xPos internal bus communication error
78	HDI input device error
79	xPos module error
80	internal error: HDI ISR not running (neo TANGOs: internal DMA error)
81	internal error: Encoder ISR not running
82	overload on motor connector +5V (PCI-E/DT-E: also on +5V of AUX I/O)
83	overload on AUX I/O +5V supply
84	overload on encoder +5V supply
85	overload on AUX I/O +12V supply or AUX mini +24V supply
86	low brake output voltage
87	overload on motor 4 connector +5V
88	overload on a supply output pin (latched overload state), clear by “!err”
89	not executable while in standby mode
90	temperature error
91	encoder error
92	overload on HDI +5V supply
93	overload on CAN 24V supply

9.2. SlideExpress and TrayExpress Errors

Error	Meaning	Explanation / Solution
100	hardware missing (IO1)	PCB option IO1 not installed inside TANGO
101	magazine not correctly seated	proof if magazine is seated correctly
102	magazine slot is empty	the slide index points to an empty slot
103	magazine slot is occupied	the slide index points to an occupied slot
104	sensor reports get failure	the slides are still visible from glass sensor
105	sensor reports put failure	the glass sensor did not detect the slide
106	sensor overmodulation	
107	magazine unknown	
108	ejector timeout	proof if ejector is mechanically blocked
109	priority handler is rear	
110	priority handler is in front	
111	priority handler is not locked	
112	priority handler position not clear	
113	priority handler timeout (front)	
114	priority handler timeout (middle)	
115	priority handler timeout (rear)	
116	front door is open	proof if front door is completely closed
117	timeout close door	
118	no priority handler available	slide index for row value must not be zero
119	gripper is not empty	proof signal from clip detector inside gripper
120	gripper contains unknown clip or slide(s)	
121	system not yet 184nitialized	calibrate at first
122	clip not correctly seated in gripper	proof signal from clip detector inside gripper
123	clamp not open	
124	tray on stage	
125	no tray in gripper	
126	step mode finished	
127	POS3 stop input	
128	no tray on stage	
129	amplifier OFF from crash detection	

9.3. RFID Interface Errors

130	RF connect
131	RF timeout
132	RF address
133	RF NAK
134	RF sync
135	RF cancel
136	RF not OK
137	RF length
138	RF chksum

9.4. Piezo Z-Stage Errors

140	Piezo connect
141	Piezo timeout
142	Piezo address

9.5. Custom Handling System Errors

Error	Meaning	Explanation / Solution
150	HL connect	the master is not yet configured
151	HL timeout	none or too slow response from slave
152	HL cal X	slave scara arm calibration problem
153	HL cal Y	slave magazine calibration problem
154	HL cal Z	slave vertical axis calibration problem
155	HL insert	
156	HL eject	
157	HL get tray	
158	HL put tray	
159	HL protocol	
160	HL hall tray detection	none of the 2 outer tray hall sensors actuated (tray not present?)
161	clamp electronic	no response from stage ETS
162	no tray on stage but RFID read	inconsistent hardware information clamp vs RFID
163	escape position Z	The Z axis is not in the safe escape position
164	HL Tray alignment	At least one tray is misaligned in the magazine
165	tray on stage but no RFID	inconsistent hardware information clamp vs RFID
166	HM motor flap timeout	The motorized flap is stuck
167	HM door open	The loading door is open
168	HM door just opened	movement of the motorized flap was interrupted from opening the loading door
169	HL laser alignment	The position of comb tines is unexpected
170	HL laser count	The number of detected comb tines is not correct
171	HL slot alignment	At least one comb tine is misaligned
172	gripper not open	Gripper did not open completely
173	cal error pos3	Error while calibrating pos3 module axis
174	loader at barcode positions	Some instructions are not possible in this position and so report error 174
175	limit switch not reached	Axis should be moved into a limit switch but did not reach it
176	IO2 hardware missing	The required IO2 module is not present (not installed or defective)
177	gripping timeout	Timeout while trying to close the gripper
178	unable to free gripper	Unable to free gripper in magazine while cal
179	error stop latch	Stop- was detected -> Cal required
180	magazine not empty	Magazine must be empty, e.g. for the “!mol” instruction

9.6. DLL Errors

As returned from DLL function calls.

0	no error	
4001	A passed pointer is NULL, a file error or failed save	
4002	A parameter value or string length is out of range	(in the function call or in a controller reply)
4003	Function call with wrong LSID or wrong axis	(Axis number or LSID out of range)
4004	Unknown interface type	(parameter of Connect/ConnectEx/ConnectSimple)
4005	Error while initializing interface	(connecting to the interface failed)
4006	No connection to the controller	(e.g., function used before connecting to controller)
4007	Timeout while reading from interface	(no reply from the controller)
4008	Error during command transmission to the controller	(send or receive error, received data error)
4009	Command aborted by SetAbortFlag call	
4010	Command is not supported by the controller	(due to firmware version or controller type)
4011	Manual Joystick mode switched on	(can occur with SetJoystickOn/Off function)
4012	No move command possible, because manual joystick enabled	
4013	Closed Loop Controller Timeout	(could not settle within target window)
4014	Error while calibrating	(Limit switch not released, timeout or cal/rm error)
4015	Limit switch actuated in travel direction	(prevents or stops axis travel)
4016	Repeated vector start	(here: if the axis is already traveling)
4031	Not possible to switch on joystick, because move active	
4032	Software limits undefined	

Errors from 4100 are used to forward error numbers from the controller.
Then, the DLL adds +4100 to the controller's error number (e.g., 5 → 4105).
Please refer to the TANGO Error Messages above, chapters 9.1 to 9.5.

4100	no error
4101	No valid axis name
4102	No executable instruction
4103	Too many characters in command line
4104	Invalid instruction
4105	Number is not inside allowed range
4106	Wrong number of parameters
4107	Either ! or ? is missing
4108	No TVR possible, while axis active
4109	No ON or OFF of axis possible while TVR active
4110	Function not configured
4111	No move instruction possible while joystick enabled
4112	Limit switch actuated
4113	Function not executable, because encoder detected
4114	Error while calibrating (limit switch not released)
4115	Opposing limit switch actuated
4120	Driver relais out of order
4121	Only single vectors allowed
4122	Door open or setup-operation
4123	SECURITY Error X-axis
4124	SECURITY Error Y-axis
4125	SECURITY Error Z-axis
4126	SECURITY Error A-axis
4127	Stop active
4128	Error in door safety circuit
4129	Amplifiers off
4130	GAL safety error
4131	Joystick cannot be switched on because move active
4132	Move discarded, target outside limit

4170	Wrong CPLD data
4171	ETS error
4172	Parameter is write protected (check lock bits)
4173	Internal error, e.g. EEPROM data corruption
4174	Closed loop switched off (parameter change, deviation or encoder error)
4175	Could not enable axis correction, or axis correction was disabled
4176	IO extension error (output overload on IO1 or Multi-IO connector)
4177	IO/xPos internal bus communication error
4178	HDI input device error
4179	xPos module error
4180	Internal error: HDI ISR not running (neo TANGOs: internal DMA error)
4181	Internal error: Encoder ISR not running
4182	Overload on motor connector +5V (PCI-E/DT-E: also on +5V of AUX I/O)
4183	Overload on AUX I/O +5V supply
4184	Overload on encoder +5V supply
4185	Overload on AUX I/O +12V supply or AUX mini +24V supply
4186	Low brake output voltage
4187	Overload on motor 4 connector +5V
4188	Overload on a supply output pin (latched overload state, clear by !err)
4189	Not executable while in standby mode
4190	Temperature error
4191	Encoder error
4192	Overload on HDI +5V supply
4193	Overload on CAN 24V supply

10. 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	
05	E	24. Mar. 2014	Chapter 2.4 reformatted to Arial text	
06	F	18. Sep. 2014	Added: GetCommandTimeout SetCommandTimeout	
07	G	11. Jul. 2016	general review Chapter 6: TrayExpress interface	
08	H	04. Jul. 2017	Added: GetSnapshotMode SetSnapshotMode SetSnapshotCount SetSnapshotPosArray ClearSnapshotPosArray GetSnapshotIndex SetSnapshotIndex Updated Error Codes Added ConnectSimple Interface Type -1	Based on TANGO_DLL 1.384 (ML)
09	I	16. Aug. 2017	Added: SetAuxDigitalOutput Corrected IO descriptions	Based on TANGO_DLL 1.385 (ML)
10	J	19. Oct. 2017	Added: SetLedBright	Based on TANGO_DLL 1.387 (ML)
11	K	01. Nov. 2017	Added: Chapter 3.3 API State Diagram	
12	L	22. Jan. 2018	new: Chapter 7 Express IFC Extensions	Implemented since version 1.388 (FD)
13	M	28. Aug. 2018	Update of Chapter 8	
14	N	18. Dec. 2018	new: SetCabinLED / GetCabinLED SetLabelLED / GetLabelLED	Implemented since version 1.397 (FD)
15	O	19. Feb. 2019	Calibrate returns more specific error code GetLoaderType return value expanded prevent endless loop at removed USB	Implemented since version 1.398 (FD)
16	P	8. Mar. 2019	Update list of TANGO error messages	
17	Q	2. Nov. 2020	Added: GetResolution / SetResolution Bugfix: USB endless wait after loosing USB connection (e.g. unplug or TANGO power down or TANGO reset)	implemented since version 1.399 (FD)
18	R	13. Apr. 2021	Added: GetAutoStatus Bugfix: LSX_Connect, LSX_LoadConfig correct description LSX_SetActiveAxes()	implemented since version 1.401 (FD)
19		06.Jan. 2022	Document type changed from.odt to.docx, new Table of Contents	(ML)

No.	Revision	Date	Changes	Remarks
20	S	18. Jan. 2022	Entirely revised DLL documentation, added connecting over Ethernet.	Based on TANGO_DLL 1.403 (ML)
21		19. Jan. 2022	Added GetTangoVersion, GetErrorString, SetControllerFactorSingleAxis, GetControllerFactorSingleAxis Improved explanations	(ML)
22		20. Jan. 2022	Added JoyChangeAxis, GetJoyChangeAxis, GetHdiKeys, ConnectEx, SetAnalogOutputMode, SetAnalogOutputMode, StopAxesEx	(ML)
23		21. Jan. 2022	Added additional handling system functions (10) and xPos functions (4) Listed all available trigger functions	(ML)
24		25. Jan. 2022	Added further functions and TVR section	Based on TANGO_DLL 1.403 (ML)
25		31. Jan. 2022	Documented HdiSpeedIndex and TVR	Based on TANGO_DLL 1.403 (ML)
26	T	31. Jan. 2022	Release for DLL 1.403 and 1.404	Based on TANGO_DLL 1.403/1.404 (ML)
27	U	10. Mar. 2022	Added GetControllerTWDelaySingleAxis, SetControllerTWDelaySingleAxis, GetBlSmoothSingleAxis, SetBlSmoothSingleAxis, GetStA Switched IsVel descriptions to the Status Request chapter	Based on TANGO_DLL 1.405 (ML)
28		28. Mar. 2022	Added SetWriteLogText Release for DLL 1.406	Based on TANGO_DLL 1.406 (ML)
29		29. Mar. 2022	Added description for Get / Set TriggerAxis, Get / Set TriggerSignalLength, Get / Set TriggerDistance, Get / Set TriggerCompensation, Get / Set TriggerEncoder, Get / Set TriggerFrequency, Get / Set TriggerOutput, Get / Set 2ndTriggerDelay, Get / Set 2ndTriggerWidth, Get / Set 2ndTriggerFrequency	(ML)
30	V	30. Mar. 2022	Added Get / Set TriggerRange, Get / Set TriggerPositionList, Get / Set TriggerPositionListIndex, Get / Set TriggerPositionListEntries, Get / Set TriggerLevel Error Messages 172 to 180 Corrected the description of GetRefSpeed, SetRefSpeed Updated, corrected several descriptions	Release for TANGO_DLL 1.406 (ML)
31		27. Jun. 2022	Added GetSecVel, SetSecVel, SetSecVelSingleAxis Corrected vel units (to not only r/sec)	Based on TANGO_DLL 1.409 (ML)
32		22. Aug. 2022	Corrected GetSlide “mode” parameter	

No.	Revision	Date	Changes	Remarks
33	W	25. Nov. 2022	Added GetAuxDigitalInput	Based on TANGO_DLL 1.410 (ML)
34		21. Dec. 2022	Added SetDigitalOutputE	Based on TANGO_DLL 1.412 (ML)
35		17. Jan. 2023	Added ClearProtocolWindow	Based on TANGO_DLL 1.413 (ML)
36		26. Jan. 2023	Changed GetTray/Puttray number of slots	(ML)
37	X	01. Feb. 2023	Changed DLL Version to 1.414	Release for TANGO_DLL 1.414 (ML)
38		15. Feb 2023	Corrected error messages description, according to new DLL version 1.415 Corrected and more clearer function descriptions	Based on TANGO_DLL 1.415 (ML)
39	Y	21. Feb. 2023	Added Get / Set EncoderSingleAxis	Based on TANGO_DLL 1.415 (ML)
40	Z	20. Nov. 2023	Updated DLL Interface Integration info	Release for TANGO_DLL 1.418 (ML)
41	ZA	27. Nov. 2023	Added GetClipType for SlideExpress	Release for TANGO_DLL 1.419 (ML)
42		28. Mar. 2024	Improved introduction (Chapters 1 and 2)	(ML)
43	ZB	02. April 2024	Released	(ML)
44	ZC	16. April 2024	Updated information of MSVC runtime, changed 2010 to 2017, and their download	Release for TANGO_DLL 1.500 (ML)
45	ZD	16. May 2024	Improved and corrected description of DLL usage and initialization, improved pitch and gear documentation	(ML)
46	ZE	13. Aug. 2024	Corrected description of SetTriggerRange / GetTriggerRange Added MATLAB support	Based on TANGO DLL 1.520 (ML)
47	ZF	19. Sep. 2024	Improved SendStringPosCmd description Improved descriptions of several functions	Based on TANGO DLL 1.520 (ML)
48		13. Feb. 2025	Improved description of SetDigJoySpeed and SetJoystickDir	(ML)
49		27. Mar. 2025	Corrected parameter description for Set/GetEncoderSingleAxis	(ML)
50	ZG	03. Jun. 2025	Corrected all function examples to LSX_	(ML)
51		11. Mar. 2026	Improved SendString description	(ML)

No.	Revision	Date	Changes	Remarks
52	ZH	17. Jul. 2026	Added single-axis functions: GetPitchSingleAxis, SetPitchSingleAxis, GetGearSingleAxis, SetGearSingleAxis, GetMotorCurrentSingleAxis, SetMotorCurrentSingleAxis, GetReductionSingleAxis, SetReductionSingleAxis, GetTargetWindowSingleAxis, SetTargetWindowSingleAxis, GetStopAccelSingleAxis, SetStopAccelSingleAxis, GetEncoderPeriodSingleAxis, SetEncoderPeriodSingleAxis, GetVelSingleAxis, GetSecVelSingleAxis, GetAccelSingleAxis, GetMotorStepsSingleAxis, SetMotorStepsSingleAxis, GetCurrentDelaySingleAxis, SetCurrentDelaySingleAxis, GetDimensionsSingleAxis, SetDimensionsSingleAxis, GetAccelFuncSingleAxis, SetAccelFuncSingleAxis, GetJoystickDirSingleAxis, SetJoystickDirSingleAxis, GetDigJoySpeedSingleAxis, SetDigJoySpeedSingleAxis. Added xPos module functions: Xpos3_Get/SetVel, Xpos3_Get/SetSpeed, Xpos3_Get/SetAccel, Xpos3_Get/SetStopAccel, Xpos3_Get/SetPitch, Xpos3_Get/SetGear, Xpos3_Get/SetAxisDir, Xpos3_Abort, Xpos3_GetStatusAxis, Xpos3_GetStatusBits, Xpos3_GetTemp, Xpos3_GetTempMCU, Xpos3_GetError, Xpos3_ClearError	Based on TANGO DLL 1.533 (ML)
53	ZI	29. Jul. 2026	New lean layout and design; improved readability; added C# and Python integration examples.	Based on TANGO DLL 1.533 (ML)
54	ZJ	30. Jul. 2026	Reworked all function reference tables for consistent column alignment (parameter names, value lists and descriptions); centered flow-chart terminators.	Based on TANGO DLL 1.533 (ML)
55	ZK	04. Aug. 2026	Completed the DLL error-message list (4114-4193) and corrected the STA status-flag list. Reworked the Supported Development Environments chapter (standard C interface; Visual Studio 2019/2022; extended environment list). Kept error sub-chapters (9.x) together per page. Removed redundant lLSID from parameter lists; unified getter wording and parameter descriptions.	Based on TANGO DLL 1.533 (ML)

No.	Revision	Date	Changes	Remarks
56	ZL	18. Aug. 2026	Reworked the protocol window chapter: an open protocol window no longer slows down the communication, because the lines are now collected and displayed in blocks; added the description of the new context menu and of the search function, with new screenshots. Added the section Reconnecting after a Reset or Power Cycle, which describes how the DLL derives its waiting time from the boot time reported by the controller. Reworked the search of the protocol window: the option for case sensitivity was replaced by Whole word only, the dialog got a Find Next button, and the screenshot was renewed. Documented the keyboard shortcuts of the protocol window. Added the new function SetPosSingleAxis. Added recognition of the TANGO-CZ-DAVINCI controller type. Refined the reconnect timing: the waiting time now allows for the repeated alignment that some controllers perform during boot, the reported boot time is checked for plausibility, and the total waiting time is limited to 60 seconds. Removed the descriptions of functions that are not implemented in the DLL and marked the remaining ones as DUMMY or NOT SUPPORTED. Corrected justified text inside tables and removed empty pages.	Based on TANGO DLL 1.540 (ML)
57	ZM	20. Aug. 2026	Implemented SetWriteLogText and SetWriteLogTextFN: the interface protocol can now be written to a file -- TANGO.log, or a file of your choice, with the Desktop as fallback -- and the path actually used is reported once in the protocol window.	Based on TANGO DLL 1.541 (ML)
58	ZN	21. Aug. 2026	Added the keyboard shortcuts HOME and END to the protocol window chapter and renewed the screenshot of its context menu, which now shows the shortcuts. Changed DLL version number to 1.544.	Based on TANGO DLL 1.544 (ML)
59	ZO	24. Aug. 2026	Documented the dark color scheme of the protocol window: negative ShowProt values open the window in dark colors, described for ConnectSimple, for SetShowProt and in the protocol window chapter. Renewed the screenshot of the protocol window, which now shows the new colors and font size. Added the missing TANGO instructions for GetStA, GetNumberOfRows and SetTrayType, and completed the parameter index of the separ instruction for GetNumberOfSlots and GetNumberOfColumns. Changed DLL version number to 1.545.	Based on TANGO DLL 1.545 (ML)

No.	Revision	Date	Changes	Remarks
60	ZP	25. Aug. 2026	Corrected the value range for the dark color scheme of the protocol window: 11 to 13 instead of negative values, and documented that any negative value is treated as 1 (show with timestamp). Reason: TRUE is 0xFFFFFFFF in Delphi LongBool, in Java via JNA and in Visual Basic, so those callers received a dark window although they only asked for the window to be shown. Added Ctrl+D, which switches the color scheme directly in the window, to the shortcut list and to the protocol window chapter, and renewed the screenshot of the context menu, which now shows that entry. Added the interface types 100 and 101 to the parameter list of ConnectSimple, together with a new section "Sending Macros" describing the pure text mode that is used to transfer macro text. Stated at SendString that the closing carriage return must come from the caller and that the function never appends one.	Based on TANGO DLL 1.545 (ML)
61		04. Sep. 2026	Added the new function ForceTriggerPulse, which fires the trigger signal manually: one single pulse, or up to 127 pulses at the configured trigger frequency. Stated at the automatic reconnect that it only works while the connection is still open; if the application calls Disconnect itself, the instance is torn down and no reconnect follows. Stated in the chapter on the "LS_" instructions that instance 1 is created when connecting only, so an "LS_" read instruction issued before the first connect returns 4003, and that the two instruction sets should not be mixed in one program. Changed DLL version number to 1.546.	Based on TANGO DLL 1.546 and 1.547 (ML)
62		09. Sep. 2026	Corrected the font size of the two notes added on 04. Sep. 2026 to 10 pt, matching the surrounding body text. Set the automatic line breaks of the ForceTriggerPulse entry in the quick reference at readable places. Made the description of ForceTriggerPulse precise: 0 emits a single pulse and returns immediately, 1 emits a single pulse as well but returns only after one period of the trigger frequency, and 2 to 127 emit that number of pulses at that frequency and return when all of them have been sent. Corrected the frequency function named there from Set2ndTriggerFrequency to SetTriggerFrequency: the second one drives the high resolution output on TAKT_OUT, not the trigger. Brought the parameter list in line with the description.	