CalibrationLibrary – Application Programming Interface

CalibrationLibrary V1.4.0

SCANLAB GmbH
Siemensstr. 2a
82178 Puchheim
Germany

Tel.+49 (89) 800 746-0
Fax+49 (89) 800 746-199

info@scanlab.de
www.scanlab.de

Contents

1 About this Manual

This manual describes the SCANLAB CalibrationLibrary DLL V1.4.0.

Notice!

Carefully read the document "Software License Agreement" before installing and using CalibrationLibrary. This agreement defines matters such as terms of usage, warranty information and liability disclaimers. If you have questions, simply contact SCANLAB.

Caution!

Read and observe all safety instructions in this manual!

SCANLAB accepts no liability for damages or consequential losses resulting from non-observance of this manual, in particular the safety instructions contained herein.

1.1 Related Documents

- RTC4 Manual
- RTC5 Manual
- RTC6 Manual
- Calibrating a 3-Axis Laser Scan System

1.2 Manufacturer

SCANLAB GmbH
Siemensstr. 2a
82178 Puchheim
Germany
Tel. +49 (89) 800 746-0
Fax +49 (89) 800 746-199
info@scanlab.de
www.scanlab.de

1.3 Intended Use

CalibrationLibrary DLL (32-bit version and 64-bit version) is part of Calibration Library software package #148051.

For developing user programs under MS Windows it provides a programming interface (API) in the form of functions. These allow:

- Loading, handling, and creating Correction files in SCANLAB format
- Offline transformation of bit coordinates using Correction files
- Simple access functions to set and retrieve certain parameters from Correction tables
- Improvement of Correction files based on user measurements

CalibrationLibrary DLL is a program library with a collection of functions. These are intended for 2D calibration and 3D calibration of SCANLAB scan systems. The functions can be implemented in existing program code and thus support the development of customized *automated calibration routines* for scan systems.

By calling functions in a dedicated manner, a SCANLAB standard correction file (`*.ct5` or `*.ctb`) loaded at the beginning can be converted step by step into a new, system-specific correction file. In this way, higher accuracies can be achieved with the scan system than with the initial correction file.

For the choice of the order of these functions, see the document Calibrating a 3-Axis Laser Scan System.

Furthermore, CalibrationLibrary DLL offers the following possibilities:

- Certain parameters of the loaded Correction file can be
  - queried
  - changed
- Offline transformations of bit coordinates
- Calculating (forward) transformations and inverse transformations that would otherwise (after corresponding RTC command calls) only take place on the RTC board itself

For access to all functionalities of CalibrationLibrary DLL are available:

- the “normal” C-API
- C# wrapper
- Python wrapper

1.4 Safety

Notice!

Carefully read the document "Software License Agreement" before installing and using CalibrationLibrary. This agreement defines matters such as terms of usage, warranty information and liability disclaimers. If you have questions, simply contact SCANLAB.

Caution!

Check your generated correction files before using them in production, for example, whether test markers have the expected quality.

1.5 Prerequisites

Precondition for the usage of the CalibrationLibrary DLL:

- Windows 10 PC

1.6 Glossary

2D-Correction file	Correction file with 2D-Correction table.
2D-Correction table	Correction table with 2D data. Corresponding to the file extension: • 2D-ct5-Correction table • 2D-ctb-Correction table
3D-Correction file	Correction file with 3D-Correction table.
3D-Correction table	Correction table with 3D data. Corresponding to the file extension: • 3D-ct5-Correction table • 3D-ctb-Correction table
ABC Calibration	The term refers to: **slcl_do_abc_calibration**
API	Abbreviation of Application Programming Interface. Program part (here: of the CalibrationLibrary DLL) which is available for other programs for connecting to the system (here: functions of the CalibrationLibrary DLL). See Chapter 3 "Functions Available in the API", page 11.
Beam Tilt Calibration	The term refers to: **slcl_do_beam_tilt_calibration**
Callback event	One of several CalibrationLibrary DLL-internal events. See also Callback function.
Callback function	Designates a user-supplied function that is to be executed when a certain "Callback event" occurs. A "Callback function" is registered to the CalibrationLibrary DLL via a Function for registering "Callback event". • **slcl_xy_calibration_bit_targets_callback** • **slcl_xy_calibration_mm_targets_callback** Therefore, it must comply to a dictated function signature: • `void ( const char*, int32_t, void* )`
Cone Calibration	The term refers to: **slcl_do_cone_calibration**
Correction file	File in SCANLAB format `*.ctb` or `*.ct5`. Contains the Correction table(s). For further information Calibrating a 3-Axis Laser Scan System.
Correction table	Relevant information inside the Correction file.
Correction table instance	Instance of a certain Correction table. Is a Correction table object which is created when **slcl_load_correction_table** is called. Every Correction table instance is represented by exactly one Handle and can be addressed by it.
ct5-Correction file	File in SCANLAB format `*.ct5`.
ct5-Correction table	Correction table of the ct5-Correction file: • 3D-ct5-Correction table • 2D-ct5-Correction table

ctb-Correction file	• File in SCANLAB format `*.ctb`.
ctb-Correction table	Correction table of the ctb-Correction file: 2D-ctb-Correction table 3D-ctb-Correction table
Cylinder Calibration	The term refers to: **slcl_do_cylinder_calibration**
Focus Calibration	The term refers to: **slcl_do_focus_calibration**
Handle	Computer programming term: abstract reference to a resource. In this manual, this term refers to a certain Correction table instance. Its Handle value is assigned by **slcl_load_correction_table**. With the (most) CalibrationLibrary functions, the Handle value of the desired target-Correction table instance must be specified.
NULL	Means on the one hand the number 0, on the other hand a pointer with the value 0.
Plane Calibration	The term refers to: **slcl_do_plane_calibration**
Readme file	Text file supplied by SCANLAB together with the Correction file: • Exactly the same name as the Correction file • for ctb-Correction files "`_ReadMe`" is appended • For ct5-Correction files "`_ct5_ReadMe`" is appended • File extension is `*.txt` For more information, see Calibrating a 3-Axis Laser Scan System.
Scale Calibration	The term refers to: **slcl_do_scale_calibration**
Stretch Calibration	The term refers to: **slcl_do_stretch_calibration**
User	Designates a person (= "system programmer") who develops user programs using the CalibrationLibrary software package. Not meant is the "user or operator of a CalibrationLibrary system".
XY Calibration	The term refers to: **slcl_xy_calibration_bit_targets** or **slcl_xy_calibration_bit_targets_callback** or **slcl_xy_calibration_mm_targets** or **slcl_xy_calibration_mm_targets_callback**

2 Software Development with the CalibrationLibrary DLL

The first step when using the CalibrationLibrary is to activate its functions by calling **slcl_activate** with the correct password. If **slcl_activate** has not been successfully called, none of the other CalibrationLibrary functions can be used!

Notice!

The password for **slcl_activate** can be obtained from SCANLAB or is included in the Calibration Library software package.

To interface with the internal functionality of the CalibrationLibrary you need a reference pointer called Handle. Each Handle represents an Correction table instance. The Handle is generated by calling **slcl_load_correction_table** and needs to be handed over to the API with every function call.You can generate as many Handles as needed. These can be deleted again selectively by **slcl_delete_correction_table_handle**. See also Chapter 6 "Example.cpp", page 66.

2.1 Installation

There are different ways to integrate the CalibrationLibrary DLL into your user program. The following is an example of how to integrate CalibrationLibrary DLL into a 32-bit Microsoft Visual Studio project.

(1) Copy `CalibrationLibrary32.dll` to your `Binary` directory.

(2) Add the `CalibrationLibrary32.lib` as a dependency to your project.

(3) Place `CalibrationLibrary.h` in your `Include` directory.

(4) Include the `CalibrationLibrary.h` in your project by `#include "CalibrationLibrary.h"`.

(5) You can now use the CalibrationLibrary DLL functions. See also Chapter 6 "Example.cpp", page 66.

3 Functions Available in the API

3.1 Functional Overview

In this chapter:

3.1.1 Administration Functions

Functions for generating, deleting and saving Handles to interface with CalibrationLibrary.

3.1.2 Correction table Evaluation Functions

Functions for using a Correction table to transform working field bit values into control bit values and vice versa.

Correction table Evaluation Functions can be used to evaluate a Correction table. With the various transform functions, working field bit values can be transformed into control bit values while the inverse transform functions convert control bit values into working field bit values.

Notes

- Inverse transformations are only available for ct5-Correction tables:
  - All calculations can be done with just a single Correction file
  - An RTC board is not required

- **slcl_get_z_distance**
- **slcl_inverse_transform_points_2d**, Page 32
- **slcl_inverse_transform_points_2d_io**, Page 33
- **slcl_inverse_transform_points_3d**, Page 34
- **slcl_inverse_transform_points_3d_fixed_z**, Page 35
- **slcl_inverse_transform_points_3d_fixed_z_io**, Page 36
- **slcl_inverse_transform_points_3d_io**, Page 37
- **slcl_inverse_transform_points_3d_io_with_trafos**, Page 38
- **slcl_inverse_transform_points_3d_with_trafos**, age 39
- **slcl_transform_points_2d**, Page 43
- **slcl_transform_points_2d_io**, Page 44
- **slcl_transform_points_3d**, Page 45
- **slcl_transform_points_3d_io**, Page 46

3.1.3 Correction table Parameter Access Functions

Functions for retrieving or setting certain parameters for a given Correction table.

Correction table Parameter Access Functions may be used to retrieve certain parameters from a Correction table and to make simple alterations to other Correction tables. If a parameter is set by Correction table Parameter Access Functions, no calculations are performed and the parameter is simply set to the new value.

- **slcl_get_current_abc_coeffs**, Page 28
- **slcl_get_current_calibration_factor**, Page 29
- **slcl_get_current_stretch_factors**, Page 29
- **slcl_set_abc_manually**, Page 41
- **slcl_set_stretch_factors_manually**, Page 42

3.1.4 Calibration Optimization Functions

Functions for improving a given Correction table based on user input.

Calibration Optimization Functions perform internal calculations to improve the Correction file based on the given input.

When improving Correction file it is critical to adhere to a logical order. Otherwise, one method of improvement may invalidate another.

Any of these steps may be skipped, but the sequence must never be altered:

(1) Converting ctb-Correction files to ct5-Correction files

(2) Plane Calibration OR Cylinder Calibration OR Cone Calibration

(3) Scale Calibration OR Cone Calibration

(4) Beam Tilt Calibration

(5) XY Calibration

(6) Focus Calibration

(7) ABC Calibration AND/OR Stretch Calibration - any or all may be done in any order

3.2 Function Reference

In this chapter:

3.2.1 General Structure of the Reference Tables

Name of the function	**prefix_name** CalibrationLibrary functions have the prefix "slcl_".
Purpose	Short description describing the purpose of the function.

Function signature	`datatype prefix name(datatype A, datatype* B, datatype C)`
	`| | | | > Line Argument(s) C`
	`| | | > Line Argument(s) B(a)`
	`| | > Line Argument(s) A`
	`| > Lines Name of the function, Purpose`
	`> Line Return value`
	Example: `uint32_t slcl_load_correction_table( size_t* Handle, const char* CorrFilename, const char* ReadMeFilename )`

Argument(s)	`A` Data type. Short text.
	`B` Data type. Short text.
	`C` Data type. Short text.
Return value	Reference to a description of the return value, for example, "See Standard return values of CalibrationLibrary functions, page 62".
Comment(s)	• Additional information on this and similar functions. • References to other chapters and publications.
Code example	Example.cpp
Version info	States the CalibrationLibrary DLL version in which the function has been published for the first time and, if applicable, further information on changes.
References	Links to related functions: **prefix_name_2**

(a) 'datatype*' (address operator) indicates a pointer.

3.2.2 Data Types of the CalibrationLibrary DLL Functions

C programming language	Data format
`char`	A presentable character of 1 byte = 8 bit.
`char*`	Pointer to a \0-terminated ANSI string, 1 byte per char. 4 Byte for Win32 executables. 8 Byte for Win64 executables. Synonym: char array, C-string.
`double`	64-bit IEEE floating point format. See https://de.wikipedia.org/wiki/IEEE_754.
`double*`	Pointer to a `double` value. `double*` can be an array also.
`int32_t`	Signed 32-bit value: $[-2^{31}\ldots+(2^{31}-1)]$.
`int32_t*`	Pointer to a signed 32-bit value: $[-2^{31}\ldots+(2^{31}-1)]$.
`size_t`	As defined in `stddef.h`. In general, `uint32_t` for Win32 Executables.
`size_t*`	Pointer to a `size_t` value.
`uint16_t`	Unsigned 16-bit value: $[0\ldots+(2^{16}-1)]$. Synonym: `unsigned short`.
`uint32_t`	Unsigned 32-bit value: $[0\ldots+(2^{32}-1)]$. Synonym: `unsigned int`.
`uint32_t*`	Pointer to a unsigned 32-bit value: $[0\ldots+(2^{32}-1)]$.
`uint64_t`	Unsigned 64-bit value: $[0\ldots+(2^{64}-1)]$. Synonym: `unsigned long long`.
`uint64_t*`	Pointer to a unsigned 64-bit value: $[0\ldots+(2^{64}-1)]$.

Notes

- `**`
  means pointer to a pointer, for example, with **slcl_load_correction_table**.
- `const`
  (for example, with `const uint64_t Password`) means that the value that follows is not changeable. That is, after the function call the value is the same as before the function call (unlike `size_t*`). `const` is used to differentiate these values from returned parameter values.
- `void`
  means that the function does not deliver a return value.
- `void*`
  means a pointer to a generic data type.

3.2.3 Reference Tables

The sequence of the reference tables in this chapter is alphabetically.

Name of the function	slcl_activate
Purpose	Category: Administration Functions. Activates CalibrationLibrary by password.
Function signature	`uint32_t slcl_activate( uint64_t Password )`
Argument(s)	`Password` 64-bit value for activation of CalibrationLibrary.
Return value	See Standard return values of CalibrationLibrary functions, page 62.
Comment(s)	• The first step when using the CalibrationLibrary is to activate its functions by calling **slcl_activate** with the correct `Password` successfully. If **slcl_activate** has not been successfully called, none of the other functions can be used!
Code example	See Example.cpp.
Version info	Available as of CalibrationLibrary DLL V1.0.0.
References	See Example.cpp.

Name of the function	slcl_convert_ctb_to_ct5	
Purpose	Category: Calibration Optimization Functions. Converts an existing ctb-Correction file into a fully valid ct5-Correction file.	
Function signature	`uint32_t slcl_convert_ctb_to_ct5( size_t CtbHandle, size_t* Ct5HandleNew, const char* NewCt5Filename )`	
Argument(s)	`CtbHandle`	Handle of the ctb-Correction table that is to be converted.
	`Ct5HandleNew`	Memory address of the newly constructed ct5-Correction table that is generated by **slcl_convert_ctb_to_ct5**.
	`NewCt5Filename`	Name of the ct5-Correction file that the resulting ct5-Correction table shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• **slcl_convert_ctb_to_ct5** requires parameters from the Readme file of the ctb-Correction file!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_delete_correction_table_handle
Purpose	Category: Administration Functions. Release a specific Correction table object.
Function signature	`uint32_t slcl_delete_correction_table_handle( size_t Handle )`
Argument(s)	`Handle` Handle to be released.
Return value	See Standard return values of CalibrationLibrary functions, page 62.
Comment(s)	• –
Code example	See Example.cpp.
Version info	Available as of CalibrationLibrary DLL V1.0.0.
References	–

Name of the function	slcl_disable_logging
Purpose	Category: Administration Functions. Disables CalibrationLibrary logging.
Function signature	`uint32_t slcl_disable_logging(void)`
Argument(s)	-
Return value	See Standard return values of CalibrationLibrary functions, page 62.
Comment(s)	• Logging is by default enabled after activation.
Code example	–
Version info	Available as of CalibrationLibrary DLL V1.0.0.
References	**slcl_enable_logging**

Name of the function	slcl_do_abc_calibration	
Purpose	Category: Calibration Optimization Functions. Calculates the ABC coefficients for the selected table for a given set of measurements for the focal length and the Z-control bit.	
Function signature	`uint32_t slcl_do_abc_calibration( size_t Handle, uint16_t NPoints, const int32_t* FocalLengthsBit, const int32_t* ZControlBit, const char* NewFilename, slcl_abc_pol_coefficients* CalculatedCoeffs )`	
Argument(s)	`Handle`	Handle for the Correction table to be used. Its ABC coefficients will be modified by the function.
	`NPoints`	Number of measured points.
	`FocalLengthsBit`	Pointer to an array of dimension `NPoints`. This array must contain the measured focal lengths: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`ZControlBit`	Pointer to an array of dimension `NPoints`. This array must contain the measured z control values: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
	`CalculatedCoeffs`	Pointer to the struct `slcl_abc_pol_coefficients` containing the new coefficients.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• –	
Code example	See Example.cpp.	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	**slcl_do_beam_tilt_calibration**	
Purpose	Category: Calibration Optimization Functions. Recalculates the Correction table using offset values measured for galvanometer scanner angles of 0.	
Function signature	`uint32_t slcl_do_beam_tilt_calibration( size_t Handle, double OffsetXMM, double OffsetYMM, const char* NewFilename )`	
Argument(s)	`Handle`	Handle of a 3D-Correction table that is to be improved. Must not be a 2D-Correction table.
	`OffsetXMM`	Measured x offset value for galvanometer scanner angle of 0. In mm.
	`OffsetYMM`	Measured y offset value for galvanometer scanner angle of 0. In mm.
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• **slcl_do_beam_tilt_calibration** parameters from the Readme file! • If you do not want to calculate the offset values yourself, use **slcl_do_beam_tilt_calibration_measurement_data**. • **slcl_do_beam_tilt_calibration** requires the previous calculation of `OffsetXMM` and `OffsetYMM`. Alternatively, you can use **slcl_do_beam_tilt_calibration_measurement_data**, which additionally takes over the calculation of the offset values.	
Code example	See Example.cpp.	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	**slcl_do_beam_tilt_calibration_measurement_data**	

Name of the function	**slcl_do_beam_tilt_calibration_measurement_data**	
Purpose	Category: Calibration Optimization Functions. Recalculates the Correction table using the measured difference values (First, calculates the offset values for galvanometer scanner angle of 0 (= `OffsetXMM`, `OffsetYMM`) from `DeltaXMM`, `DeltaYMM`, `HeightMM`. Then calls **slcl_do_beam_tilt_calibration**).	
Function signature	`uint32_t slcl_do_beam_tilt_calibration_measurement_data( size_t Handle, double DeltaXMM, double DeltaYMM, double HeightMM, const char* NewFilename )`	
Argument(s)	`Handle`	Handle of a 3D-Correction table that is to be improved. Must not be a 2D-Correction table.
	`DeltaXMM`	Measured difference in x direction. In mm.
	`DeltaYMM`	Measured difference in y direction. In mm.
	`HeightMM`	Distance between the 2 measuring planes in z direction. In mm.
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• **slcl_do_beam_tilt_calibration_measurement_data** parameters from the Readme file!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.4.0.	
References	**slcl_do_beam_tilt_calibration**	

Name of the function	slcl_do_cone_calibration	
Purpose	Category: Calibration Optimization Functions. Recalculates a 3D-Correction file for a user-defined conic target surface that can be shifted, tilted, and rotated.	
Function signature	`uint32_t slcl_do_cone_calibration( size_t Handle, const double* ReferencePointMM, const double* ConeDirection, const double ConeRadiusAtOriginMM, const double ConeInclinationAngleRad, const char* NewFilename )`	
Argument(s)	`Handle`	Handle of the Correction table that is to be changed. Must be 3d.
	`ReferencePointMM`	Pointer to an array of dimension 3 containing a reference point on the cone.
	`ConeDirection`	Pointer to an array of dimension 3 containing the direction of the cone axis.
	`ConeRadiusAtOriginMM`	Radius of the cone at the reference point. In mm.
	`ConeInclinationAngleRad`	Inclination angle of the cone. In rad. Must be between ±π/3. Must not be NULL.
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• The target surface must be provided as a reference point, a direction vector, the radius at the reference point and the inclination angle of the cone. The reference point is a point on the lateral surface of the cone and represents the new coordinate origin of the target surface. In the cross-section of the cone that is perpendicular to the direction of the cone it is the point with the largest z coordinate. The given direction is the direction of the cone axis, which will represent the x coordinate in the new coordinate system. The new y coordinate is the angular offset from the surface line going through the reference point multiplied by the radius of the cone at the reference point. This means that the same y coordinate describes a shorter distance at a more narrow part of the cone and a larger distance at a wider part of the cone. The inclination angle is provided in radians and must positive if the cone widens for larger values of x, and negative if it becomes more narrow. • The new file should not be used for points outside the provided target surface. • **slcl_do_cone_calibration** requires parameters from the Readme file!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_do_cylinder_calibration	
Purpose	Category: Calibration Optimization Functions. Recalculates a 3D-Correction file for a user-defined cylindrical target surface that can be shifted, tilted, and rotated.	
Function signature	`uint32_t slcl_do_cylinder_calibration( size_t Handle, const double* ReferencePointMM, const double* CylinderDirection, const double CylinderRadiusMM, const char* NewFilename )`	
Argument(s)	`Handle`	Handle of the 3D-Correction table that is to be changed. Must not be a 2D-Correction table.
	`ReferencePointMM`	Pointer to an array of dimension 3 containing a reference point on the cylinder.
	`CylinderDirection`	Pointer to an array of dimension 3 containing the direction of the cylinder axis.
	`CylinderRadiusMM`	Radius of the cylinder.
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• The target surface must be provided as a reference point, a direction vector, and a radius. The reference point is a point on the cylinder mantle and represents the new coordinate origin of the target surface. In the cylinder cross-section it is the point with the largest z coordinate. The given direction is the direction of the cylinder axis, which will represent the x coordinate in the new coordinate system. The new y coordinate is the arc length distance on the cylinder mantle from the axis parallel to the cylinder direction going through the reference point. • The new file should not be used for points outside the provided target surface. • **slcl_do_cylinder_calibration** requires parameters from the Readme file!	
Code example	See Example.cpp.	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_do_focus_calibration	
Purpose	Category: Calibration Optimization Functions. Re-adjusts the 3d calibration based on measurements in the Z = 0 plane.	
Function signature	`uint32_t slcl_do_focus_calibration( size_t Handle, uint16_t NPoints, const int32_t* XMeasurementBit, const int32_t* YMeasurementBit, int32_t* ZControlBit, const char* NewFilename )`	
Argument(s)	`Handle`	Handle of the Correction table that is to be changed.
	`NPoints`	Number of measured points.
	`XMeasurementBit`	Pointer to an array of dimension `NPoints`. This array must contain the measured x values: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`YMeasurementBit`	Pointer to an array of dimension `NPoints`. This array must contain the measured y values: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`ZControlBit`	Pointer to an array of dimension `NPoints`. This array must contain the measured Z-control values: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• –	
Code example	See Example.cpp.	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_do_freeform_pointcloud_calibration	
Purpose	Category: Calibration Optimization Functions. Recalculates a 3D-Correction file for a user-defined free-form target surface that is defined by a point cloud.	
Function signature	`uint32_t slcl_do_freeform_pointcloud_calibration( size_t Handle, uint32_t NPoints, const double* XValuesMM, const double* YValuesMM, const double* ZValuesMM, const char* NewFilename )`	
Argument(s)	`Handle`	Handle of the 3D-Correction table that is to be changed.
	`NPoints`	Number of points in the provided point cloud.
	`XValuesMM`	Pointer to an array of dimension `NPoints`. This array must contain the x coordinates of all the points in the point cloud.
	`YValuesMM`	Pointer to an array of dimension `NPoints`. This array must contain the y coordinates of all the points in the point cloud.
	`ZValuesMM`	Pointer to an array of dimension `NPoints`. This array must contain the z coordinates of all the points in the point cloud.
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• The target surface must be provided as a a series of points in 3d space. **slcl_do_freeform_pointcloud_calibration** interpolates between these points and adjust the Correction table so that the laser spot is always in focus on the given surface. Later, users need to command 2d positions to the RTC only – the focal shift is automatically adjusted. The coordinate system after the correction uses top view projection, meaning that X and Y are exactly the same as before. • **slcl_do_freeform_pointcloud_calibration** requires parameters from the Readme file!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_do_plane_calibration	
Purpose	Category: Calibration Optimization Functions. Recalculates a 3D-Correction table for a new user-defined target plane that may be shifted, tilted, and rotated.	
Function signature	`uint32_t slcl_do_plane_calibration( size_t Handle, const double* ReferencePointMM, const double* NewXDirection, const double* NewYDirection, const char* NewFilename )`	
Argument(s)	`Handle`	Handle of the 3D-Correction table that is to be changed. Must not be a 2D-Correction table.
	`ReferencePointMM`	Pointer to an array of dimension 3. This array must contain a reference point for the new target plane. This point will be treated as the coordinate origin for the target plane.
	`NewXDirection`	Pointer to an array of dimension 3. This array must contain the new x direction in the target plane.
	`NewYDirection`	Pointer to an array of dimension 3. This array must contain the new y direction in the target plane. Must be perpendicular to `NewXDirection`.
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• The new target plane must be provided in parameter form with a reference point and two direction vectors. The reference point represents the new origin of the target plane, while the direction vectors represent the new x direction and y direction in the target plane. The direction vectors must be perpendicular. • Depending on how much shifted and tilted the new target plane is compared to the original target plane, the Correction file may not be useful anymore for 3d-points that are not in the target plane. This cannot be remedied through further 3d-correction. • **slcl_do_plane_calibration** requires parameters from the Readme file!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	**slcl_do_scale_calibration**	
Purpose	Category: Calibration Optimization Functions. Recalculates the Correction table using a new calibration factor that is provided by the user.	
Function signature	`uint32_t slcl_do_scale_calibration( size_t Handle, uint32_t DesiredCalFactor, uint32_t* NewCalFactor, const char* NewFilename )`	
Argument(s)	`Handle`	Handle of the Correction table that is to be changed.
	`DesiredCalFactor`	The desired new calibration factor.
	`NewCalFactor`	The calibration factor that was ultimately used.
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• **slcl_do_scale_calibration** calculates the minimum calibration factor. It is allowed to provide a larger factor, but if it is smaller, the minimum calibration factor is used instead. If the provided factor is zero, the minimum calibration factor is used automatically. • **slcl_do_scale_calibration** requires parameters from the Readme file!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_do_stretch_calibration	
Purpose	Category: Calibration Optimization Functions. Calculates the stretch factors for the selected table for a given set of target and measured positions.	
Function signature	`uint32_t slcl_do_stretch_calibration( size_t Handle, uint16_t NPoints, const double* XTargetsMM, const double* YTargetsMM, const double* ZOffsetMM, const double* XMeasurementsMM, const double* YMeasurementsMM, const char* NewFilename, slcl_stretch_factors* CalculatedStretchFactors )`	
Argument(s)	`Handle`	Handle for the Correction table to be used. Its stretch factors will be modified by the function.
	`NPoints`	Number of measured points.
	`XTargetsMM`	Pointer to an array of dimension `NPoints`. This array must contain the targeted x values in mm.
	`YTargetsMM`	Pointer to an array of dimension `NPoints`. This array must contain the targeted y values in mm.
	`ZOffsetMM`	Pointer to an array of dimension `NPoints`. This array must contain the offset of the targeted Z-plane in mm.
	`XMeasurementsMM`	Pointer to an array of dimension `NPoints`. This array must contain the measured x values in mm.
	`YMeasurementsMM`	Pointer to an array of dimension `NPoints`. This array must contain the measured y values in mm.
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
	`CalculatedStretchFactors`	Pointer to the struct `slcl_stretch_factors` that the coefficients are written into.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• –	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_enable_logging
Purpose	Category: Administration Functions. Enables CalibrationLibrary logging.
Function signature	`uint32_t slcl_enable_logging(void)`
Argument(s)	–
Return value	See Standard return values of CalibrationLibrary functions, page 62.
Comment(s)	• Logging is by default enabled after activation.
Code example	–
Version info	Available as of CalibrationLibrary DLL V1.0.0.
References	**slcl_disable_logging**

Name of the function	slcl_get_current_abc_coeffs	
Purpose	Category: Correction table Parameter Access Functions. Returns the current coefficients of the ABC polynomial for the selected Correction table.	
Function signature	`uint32_t slcl_get_current_abc_coeffs( size_t Handle, slcl_abc_pol_coefficients* CurrentCoeffs )`	
Argument(s)	`Handle`	Handle for the Correction table to be used.
	`CurrentCoeffs`	Pointer to struct `slcl_abc_pol_coefficients`.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• –	
Code example	See Example.cpp.	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	**slcl_get_current_calibration_factor**
Purpose	Category: Correction table Parameter Access Functions. Returns the current calibration factor of the selected Correction table.
Function signature	`uint32_t slcl_get_current_calibration_factor( size_t Handle, uint32_t* CurrentCalFactor )`
Argument(s)	`Handle` Handle for the desired Correction table.
	`CurrentCalFactor` Pointer to the variable that the calibration factor are written into.
Return value	See Standard return values of CalibrationLibrary functions, page 62.
Comment(s)	• If the file is Ctb and not 3d, parameters from the Readme file are needed!
Code example	See Example.cpp.
Version info	Available as of CalibrationLibrary DLL V1.0.0.
References	–

Name of the function	**slcl_get_current_stretch_factors**
Purpose	Category: Correction table Parameter Access Functions. Returns the current stretch factors of the selected Correction table.
Function signature	`uint32_t slcl_get_current_stretch_factors( size_t Handle, slcl_stretch_factors* CurrentStretchFactors )`
Argument(s)	`Handle` Handle for the Correction table to be used.
	`CurrentStretchFactors` Pointer to the struct `slcl_stretch_factors` in which the stretch factors are written into.
Return value	See Standard return values of CalibrationLibrary functions, page 62.
Comment(s)	• –
Code example	–
Version info	Available as of CalibrationLibrary DLL V1.0.0.
References	–

Name of the function	slcl_get_lib_version
Purpose	Category: Administration Functions. Returns version info on the currently running CalibrationLibrary DLL.
Function signature	`VersionInfo slcl_get_lib_version(void)`
Argument(s)	–
Return value	See struct `VersionInfo`.
Comment(s)	• –
Code example	–
Version info	Available as of CalibrationLibrary DLL V1.0.0.
References	–

Name of the function	slcl_get_z_distance	
Purpose	Category: Correction table Evaluation Functions. Returns the focal length for a given Point in the 3D volume.	
Function signature	`uint32_t slcl_get_z_distance( size_t Handle, int32_t InputX, int32_t InputY, int32_t InputZ, int32_t* ZDistance );`	
Argument(s)	`Handle`	Handle for the 3D-Correction table to be used. Must not be a 2D-Correction table.
	`InputX`	x coordinate of the given point: • As 20-bit value, if the Handle is Ct5 • As 16-bit value, if the Handle is Ctb
	`InputY`	y coordinate of the given point: • As 20-bit value, if the Handle is Ct5 • As 16-bit value, if the Handle is Ctb
	`InputZ`	z coordinate of the given point: • As 20-bit value, if the Handle is Ct5 • As 16-bit value, if the Handle is Ctb
	`ZDistance`	Pointer to the resulting value for the focal length: • As 20-bit value, if the Handle is Ct5 • As 16-bit value, if the Handle is Ctb
Return value	See struct `VersionInfo`.	
Comment(s)	• The returned `ZDistance` is the difference in focal length between the provided point (`InputX`|`InputY`|`InputZ`) and (0|0|0). • If the calculation is impossible due to the input point being out of range for the system, a very large value are written into `ZDistance`. • **slcl_get_z_distance** and the RTC command **get_z_distance** are equivalent.	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_inverse_transform_points_2d	
Purpose	Category: Correction table Evaluation Functions. Conduct an inverse 2d transformation using a ct5-Correction table.	
Function signature	`uint32_t slcl_inverse_transform_points_2d( size_t Handle, uint32_t NPoints, const int32_t* InputX, const int32_t* InputY, int32_t* ResultX, int32_t* ResultY )`	
Argument(s)	`Handle`	Handle for the ct5-Correction table to be used.
	`NPoints`	Number of points to be transformed.
	`InputX`	Pointer to an array of dimension `NPoints`. This array must contain the x values as 20-bit values.
	`InputY`	Pointer to an array of dimension `NPoints`. This array must contain the y values as 20-bit values.
	`ResultX`	Pointer to an array of dimension `NPoints`. The results for X are written into this array as 20-bit values.
	`ResultY`	Pointer to an array of dimension `NPoints`. The results for Y are written into this array as 20-bit values.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the transformation is impossible due to the input points being out of range for the system, a very large value are written into each position of the output arrays for that set of measurements. All other points will still be attempted to be transformed normally. This function may also be called for a 3D-ct5-Correction table, in which case the result is the intersection of the laser beam with the z = 0 plane. • The specified Handle table must be that of a ct5-Correction table. Inverse transformation is not available for ctb-Correction tables!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_inverse_transform_points_2d_io	
Purpose	Category: Correction table Evaluation Functions. Conduct an inverse 2d transformation using a ct5-Correction table.	
Function signature	`uint32_t slcl_inverse_transform_points_2d_io( size_t Handle, uint32_t NPoints, int32_t* IOX, int32_t* IOY )`	
Argument(s)	`Handle`	Handle for the ct5-Correction table to be used.
	`NPoints`	Number of points to be transformed.
	`IOX`	Pointer to an array of dimension `NPoints`. This array must contain the x values. The results for X are written into this array as 20-bit values.
	`IOY`	Pointer to an array of dimension `NPoints`. This array must contain the y values. The results for Y are written into this array as 20-bit values.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the transformation is impossible due to the input points being out of range for the system, a very large value are written into each position of the output arrays for that set of measurements. All other points will still be attempted to be transformed normally. This function may also be called for a 3D-ct5-Correction table, in which case the result is the intersection of the laser beam with the z=0 plane. • The specified Handle table must be that of a ct5-Correction table. Inverse transformation is not available for ctb-Correction tables!	
Code example	See Example.cpp.	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_inverse_transform_points_3d	
Purpose	Category: Correction table Evaluation Functions. Conduct an inverse 3d transformation using a ct5-Correction table.	
Function signature	`uint32_t slcl_inverse_transform_points_3d( size_t Handle, uint32_t NPoints, const int32_t* InputX, const int32_t* InputY, const int32_t* InputZ, int32_t* ResultX, int32_t* ResultY, int32_t* ResultZ )`	
Argument(s)	`Handle`	Handle for the 3D-ct5-Correction table to be used.
	`NPoints`	Number of points to be transformed.
	`InputX`	Pointer to an array of dimension `NPoints`. This array must contain the x values as 20-bit values.
	`InputY`	Pointer to an array of dimension `NPoints`. This array must contain the y values as 20-bit values.
	`InputZ`	Pointer to an array of dimension `NPoints`. This array must contain the z values as 20-bit values.
	`ResultX`	Pointer to an array of dimension `NPoints`. The results for X-are written into the array as 20-bit values.
	`ResultY`	Pointer to an array of dimension `NPoints`. The results for Y are written into the array as 20-bit values.
	`ResultZ`	Pointer to an array of dimension `NPoints`. The results for Z are written into the array as 20-bit values.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the transformation is impossible due to the input points being out of range for the system, a very large value are written into each position of the output arrays for that set of measurements. All other points will still be attempted to be transformed normally. • The specified Handle table must be that of a ct5-Correction table. Inverse transformation is not available for ctb-Correction tables!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_inverse_transform_points_3d_fixed_z	
Purpose	Category: Correction table Evaluation Functions. Conduct an inverse 3d transformation using a ct5-Correction table and a fixed Z-value in the working field.	
Function signature	`uint32_t slcl_inverse_transform_points_3d_fixed_z( size_t Handle, uint32_t NPoints, const int32_t* InputX, const int32_t* InputY, const int32_t* ImageFieldZ, int32_t* ResultX, int32_t* ResultY );`	
Argument(s)	`Handle`	Handle for the ct5-Correction table to be used.
	`NPoints`	Number of points to be transformed.
	`InputX`	Pointer to an array of dimension `NPoints`. This array must contain the x values as 20-bit values.
	`InputY`	Pointer to an array of dimension `NPoints`. This array must contain the y values as 20-bit values.
	`ImageFieldZ`	Pointer to an array of dimension `NPoints`. This array must contain the z coordinates in the working field as 20-bit values.
	`ResultX`	Pointer to an array of dimension `NPoints`. The results for X-are written into the array as 20-bit values.
	`ResultY`	Pointer to an array of dimension `NPoints`. The results for Y-are written into the array as 20-bit values.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the transformation is impossible due to the input points being out of range for the system, a very large value are written into each position of the output arrays for that set of measurements. All other points will still be attempted to be transformed normally. • The specified Handle table must be that of a ct5-Correction table. Inverse transformation is not available for ctb-Correction tables!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_inverse_transform_points_3d_fixed_z_io	
Purpose	Category: Correction table Evaluation Functions. Conduct an inverse 3d transformation using a ct5-Correction table, using a fixed z-value in the working field.	
Function signature	`uint32_t slcl_inverse_transform_points_3d_fixed_z_io( size_t Handle, uint32_t NPoints, int32_t* IOX, int32_t* IOY, const int32_t* ImageFieldZ );`	
Argument(s)	`Handle`	Handle for the ct5-Correction table to be used.
	`NPoints`	Number of points to be transformed.
	`IOX`	Pointer to an array of dimension `NPoints`. This array must contain the x values. The results for X are written into the array as 20-bit values.
	`IOY`	Pointer to an array of dimension `NPoints`. This array must contain the y values. The results for Y are written into the array as 20-bit values.
	`ImageFieldZ`	Pointer to an array of dimension `NPoints`. This array must contain the z coordinates in the working field as 20-bit values.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the transformation is impossible due to the input points being out of range for the system, a very large value are written into each position of the output arrays for that set of measurements. All other points will still be attempted to be transformed normally. • The specified Handle table must be that of a ct5-Correction table. Inverse transformation is not available for ctb-Correction tables!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_inverse_transform_points_3d_io	
Purpose	Category: Correction table Evaluation Functions. Conduct an inverse 3d transformation using a ct5-Correction table.	
Function signature	`uint32_t slcl_inverse_transform_points_3d_io( const size_t Handle, const uint32_t NPoints, int32_t* IOX, int32_t* IOY, int32_t* IOZ )`	
Argument(s)	`Handle`	Handle for the 3D-ct5-Correction table to be used.
	`NPoints`	Number of points to be transformed.
	`IOX`	Pointer to an array of dimension `NPoints`. This array must contain the x values. The results for X are written into the array as 20-bit values.
	`IOY`	Pointer to an array of dimension `NPoints`. This array must contain the y values. The results for Y are written into the array as 20-bit values.
	`IOZ`	Pointer to an array of dimension `NPoints`. This array must contain the z values. The results for Z are written into the array as 20-bit values.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the transformation is impossible due to the input points being out of range for the system, a very large value are written into each position of the output arrays for that set of measurements. All other points will still be attempted to be transformed normally. • The specified Handle table must be that of a ct5-Correction table. Inverse transformation is not available for ctb-Correction tables!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_inverse_transform_points_3d_io_with_trafos	
Purpose	Category: Correction table Evaluation Functions. Conduct an inverse 3d transformation using a ct5-Correction table.	
Function signature	`uint32_t slcl_inverse_transform_points_3d_io_with_trafos( const size_t Handle, const slcl_additional_transformations* AdditionalTrafos, const uint32_t NPoints, uint32_t* IOX, uint32_t* IOY, uint32_t* IOZ);`	
Argument(s)	`Handle`	Handle for the 3D-ct5-Correction table to be used.
	`AdditionalTrafos`	Additional transformations to take into account. See struct **slcl_additional_transformations**.
	`NPoints`	Number of points to be transformed.
	`IOX`	Pointer to an array of dimension `NPoints`. This array must contain the x values. The results for X are written into the array as 20-bit values.
	`IOY`	Pointer to an array of dimension `NPoints`. This array must contain the y values. The results for Y are written into the array as 20-bit values.
	`IOZ`	Pointer to an array of dimension `NPoints`. This array must contain the z values. The results for Z are written into the array as 20-bit values.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the transformation is impossible due to the input points being out of range for the system, a very large value are written into each position of the output arrays for that set of measurements. All other points will still be attempted to be transformed normally. • The specified Handle table must be that of a ct5-Correction table. Inverse transformation is not available for ctb-Correction tables!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.1.0.	
References	–	

Name of the function	slcl_inverse_transform_points_3d_with_trafos	
Purpose	Category: Correction table Evaluation Functions. Conduct an inverse 3d transformation using a ct5-Correction table.	
Function signature	`uint32_t slcl_inverse_transform_points_3d_with_trafos( size_t Handle, const slcl_additional_transformations* AdditionalTrafos, uint32_t NPoints, const int32_t* InputX, const int32_t* InputY, const int32_t* InputZ, int32_t* ResultX, int32_t* ResultY, int32_t* ResultZ);`	
Argument(s)	`Handle`	Handle for the 3D-ct5-Correction table to be used.
	`AdditionalTrafos`	Additional transformations to take into account. See struct **slcl_additional_transformations**.
	`NPoints`	Number of points to be transformed.
	`InputX`	Pointer to an array of dimension `NPoints`. This array must contain the x values as 20-bit values.
	`InputY`	Pointer to an array of dimension `NPoints`. This array must contain the y values as 20-bit values.
	`InputZ`	Pointer to an array of dimension `NPoints`. This array must contain the z values as 20-bit values.
	`ResultX`	Pointer to an array of dimension `NPoints`. The results for X are written into the array as 20-bit values.
	`ResultY`	Pointer to an array of dimension `NPoints`. The results for Y are written into the array as 20-bit values.
	`ResultZ`	Pointer to an array of dimension `NPoints`. The results for Z are written into the array as 20-bit values.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the transformation is impossible due to the input points being out of range for the system, a very large value are written into each position of the output arrays for that set of measurements. All other points will still be attempted to be transformed normally. • The specified Handle table must be that of a ct5-Correction table. Inverse transformation is not available for ctb-Correction tables!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.1.0.	
References	–	

Name of the function	**slcl_load_correction_table**	
Purpose	Category: Administration Functions. Initializes a Handle from the specified file name.	
Function signature	`uint32_t slcl_load_correction_table( size_t* Handle, const char* CorrFilename, const char* ReadMeFilename )`	
Argument(s)	`Handle`	Memory address of the newly created Correction table Handle.
	`CorrFilename`	Name of the desired Correction file.
	`ReadMeFilename`	Name of the corresponding Readme file. If null, the Readme file is assumed to be in the same folder as the Correction file, that it has the exact same name as the Correction file, appended by "`_ReadMe`" (for ctb-Correction files) or "`_ct5_ReadMe`" (for ct5-Correction files), and that its extension is "`*.txt`".
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• **slcl_load_correction_table** creates a new Correction table object and returns a Handle to interface with it. If **slcl_load_correction_table** fails, the Handle is set to 0 and no instance is created. • It is also possible to provide a path for the Readme file belonging to the Correction file. Information from this Readme file are needed (for calculations) by: – **slcl_convert_ctb_to_ct5** – **slcl_do_scale_calibration** – **slcl_do_beam_tilt_calibration** in particular, in case of 2D-ctb-Correction tables – **slcl_get_current_calibration_factor** – **slcl_xy_calibration_mm_targets** – **slcl_xy_calibration_mm_targets_callback**	
Code example	See Example.cpp.	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	**slcl_delete_correction_table_handle**	

Name of the function	slcl_save_correction_table	
Purpose	Category: Administration Functions Saves the specified Handle as a file.	
Function signature	`uint32_t slcl_save_correction_table( size_t Handle, const char* NewFilename )`	
Argument(s)	`Handle`	Handle for the Correction table.
	`NewFilename`	Name of the file that the result shall be saved to.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the table contains a valid Readme filename, a new Readme file will be generated as well.	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_set_abc_manually	
Purpose	Category: Correction table Parameter Access Functions. Manually sets the ABC coefficients for the selected Handle to given values and then saves it as a file.	
Function signature	`uint32_t slcl_set_abc_manually( size_t Handle, const slcl_abc_pol_coefficients* NewCoeffs, const char* NewFilename )`	
Argument(s)	`Handle`	Handle for the Correction table whose ABC coefficients shall be changed.
	`NewCoeffs`	Pointer to the struct `slcl_abc_pol_coefficients` that the coefficients are written into.
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• –	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_set_ct5_parameters_manually	
Purpose	Category: Administration Functions. Supplies the needed parameters from the Readme file manually for a ct5-Correction file.	
Function signature	`uint32_t slcl_set_ct5_parameters_manually( size_t Handle, const slcl_ct5_table_readme_parameters* Params, const char* NewFilename )`	
Argument(s)	`Handle`	Handle for the ct5-Correction table to be improved.
	`Params`	Pointer to the struct `slcl_ct5_table_readme_parameters`.
	`NewFilename`	Name of the file that the result shall be saved to.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• For ct5 beam tilt correction, the z-range (for 3D systems) and the lens distortion coefficients (for systems with f-theta objective) are needed. These parameters normally must be read from the Readme file. If there is no Readme file available, but the necessary parameters are known, they can be supplied by **slcl_set_ct5_parameters_manually**.	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_set_stretch_factors_manually	
Purpose	Category: Correction table Parameter Access Functions. Manually sets the stretch factors for the selected Correction table to given values.	
Function signature	`uint32_t slcl_set_stretch_factors_manually( size_t Handle, const slcl_stretch_factors* NewStretchFactors, const char* NewFilename )`	
Argument(s)	`Handle`	Handle for the Correction table whose stretch factors shall be changed.
	`NewStretchFactors`	Pointer to the struct `slcl_stretch_factors` containing the new stretch factors.
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• –	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_transform_points_2d	
Purpose	Category: Correction table Evaluation Functions. Conduct a forward 2d transformation using a Correction table.	
Function signature	`uint32_t slcl_transform_points_2d( size_t Handle, uint32_t NPoints, const int32_t* InputX, const int32_t* InputY, int32_t* ResultX, int32_t* ResultY )`	
Argument(s)	`Handle`	Handle for the Correction table to be used.
	`NPoints`	Number of points to be transformed.
	`InputX`	Pointer to an array of dimension `NPoints`. This array must contain the x values: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`InputY`	Pointer to an array of dimension `NPoints`. This array must contain the y values: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`ResultX`	Pointer to an array of dimension `NPoints`. The results for X are written into the array: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`ResultY`	Pointer to an array of dimension `NPoints`. The results for Y are written into the array: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the transformation is impossible due to the input points being out of range for the system, a very large value are written into each position of the output arrays for that set of measurements. All other points will still be attempted to be transformed normally. • **slcl_transform_points_2d** may also be called for a 3D-Correction table in which case it is assumed that the desired z value is 0.	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	**slcl_transform_points_2d_io**	
Purpose	Category: Correction table Evaluation Functions. Conduct a forward 2d transformation using a Correction table.	
Function signature	`uint32_t slcl_transform_points_2d_io( size_t Handle, uint32_t NPoints, int32_t* IOX, int32_t* IOY )`	
Argument(s)	`Handle`	Handle for the Correction table to be used.
	`NPoints`	Number of points to be transformed.
	`IOX`	Pointer to an array of dimension `NPoints`. This array must contain the x values. The results for X are written into the array: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`IOY`	Pointer to an array of dimension `NPoints`. This array must contain the y values. The results for Y are written into the array: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the transformation is impossible due to the input points being out of range for the system, a very large value are written into each position of the output arrays for that set of measurements. All other points will still be attempted to be transformed normally. This function may also be called for a 3d-table, in which case it is assumed that the desired z value is zero.	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_transform_points_3d	
Purpose	Category: Correction table Evaluation Functions. Conduct a forward 3d transformation using a Correction table.	
Function signature	`uint32_t slcl_transform_points_3d( size_t Handle, uint32_t NPoints, const int32_t* InputX, const int32_t* InputY, const int32_t* InputZ, int32_t* ResultX, int32_t* ResultY, int32_t* ResultZ )`	
Argument(s)	`Handle`	Handle for the 3D-Correction table to be used.
	`NPoints`	Number of points to be transformed.
	`InputX`	Pointer to an array of dimension `NPoints`. This array must contain the x values: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`InputY`	Pointer to an array of dimension `NPoints`. This array must contain the y values: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`InputZ`	Pointer to an array of dimension `NPoints`. This array must contain the y values: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`ResultX`	Pointer to an array of dimension `NPoints`. The results for X are written into the array: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`ResultY`	Pointer to an array of dimension `NPoints`. The results for Y are written into the array: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`ResultZ`	Pointer to an array of dimension `NPoints`. The results for Z are written into the array: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the transformation is impossible due to the input points being out of range for the system, a very large value are written into each position of the output arrays for that set of measurements. All other points will still be attempted to be transformed normally.	
Code example	See Example.cpp.	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_transform_points_3d_io	
Purpose	Category: Correction table Evaluation Functions. Conduct a forward 3d transformation using a Correction table.	
Function signature	`uint32_t slcl_transform_points_3d_io( size_t Handle, uint32_t NPoints, int32_t* IOX, int32_t* IOY, int32_t* IOZ )`	
Argument(s)	`Handle`	Handle for the 3D-Correction table to be used.
	`NPoints`	Number of points to be transformed.
	`IOX`	Pointer to an array of dimension `NPoints`. This array must contain the x values. The results for X are written into the array: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`IOY`	Pointer to an array of dimension `NPoints`. This array must contain the y values. The results for Y are written into the array: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`IOZ`	Pointer to an array of dimension `NPoints`. This array must contain the z values. The results for Z are written into the array: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the transformation is impossible due to the input points being out of range for the system, a very large value are written into each position of the output arrays for that set of measurements. All other points will still be attempted to be transformed normally.	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_xy_calibration_bit_targets	
Purpose	Category: Calibration Optimization Functions. Recalculates the values of the X- and Y-tables as well as the inverse X- and Y-tables (for Ct5) based on a given input of target points and measured points.	
Function signature	`uint32_t slcl_xy_calibration_bit_targets(size_t Handle, uint16_t NPoints, const int32_t* XTargetsBit, const int32_t* YTargetsBit, const double* XMeasurementsMM, const double* YMeasurementsMM, const slcl_xy_calibration_settings* OtherParameters, slcl_xy_calibration_interpolation_results* InterpolationResults, const char* NewFilename)`	
Argument(s)	`Handle`	Handle for the Correction table to be improved.
	`NPoints`	Number of measured points.
	`XTargetsBit`	Pointer to an array of dimension `NPoints`. This array must contain the targeted x values: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`YTargetsBit`	Pointer to an array of dimension `NPoints`. This array must contain targeted y values: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`XMeasurementsMM`	Pointer to an array of dimension `NPoints`. This array must contain the measured x values in mm.
	`YMeasurementsMM`	Pointer to an array of dimension `NPoints`. This array must contain the measured y values in mm.
	`OtherParameters`	Pointer to a struct `slcl_xy_calibration_settings`. This struct must contain the parameters for the interpolation.
	`InterpolationResults`	Pointer to a struct `slcl_xy_calibration_interpolation_results`. This struct must contain additional information about the interpolation results.
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• –	
Code example	Example.cpp	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_xy_calibration_bit_targets_callback	
Purpose	Category: Calibration Optimization Functions. Recalculates the values of the X- and Y-tables as well as the inverse X- and Y-tables (for Ct5) based on a given input of target points and measured points.	
Function signature	`uint32_t slcl_xy_calibration_bit_targets_callback(size_t Handle, uint16_t NPoints, const int32_t* XTargetsBit, const int32_t* YTargetsBit, const double* XMeasurementsMM, const double* YMeasurementsMM, const slcl_xy_calibration_settings* OtherParameters, slcl_xy_calibration_interpolation_results* InterpolationResults, void (*Callback)(const char*, uint32_t, void*) Callback, void* Context, const char* NewFilename)`	
Argument(s)	`Handle`	Handle for the Correction table to be improved.
	`NPoints`	Number of measured points.
	`XTargetsBit`	Pointer to an array of dimension `NPoints`. This array must contain the targeted x values: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`YTargetsBit`	Pointer to an array of dimension `NPoints`. This array must contain the targeted y values: • As 20-bit values, if the Handle is Ct5 • As 16-bit values, if the Handle is Ctb
	`XMeasurementsMM`	Pointer to an array of dimension `NPoints`. This array must contain the measured x values in mm.
	`YMeasurementsMM`	Pointer to an array of dimension `NPoints`. This array must contain the measured y values in mm.
	`OtherParameters`	Pointer to struct `slcl_xy_calibration_settings` containing the parameters for the interpolation.
	`InterpolationResults`	Pointer to struct `slcl_xy_calibration_interpolation_results` containing additional information about the interpolation results.
	`Callback`	Function pointer to a Callback function which informs about the progress of the calculation. It contains a message, a progress percentage value, and a context pointer.
	`Context`	Pointer to the context of the Callback function.
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• –	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_xy_calibration_mm_targets	
Purpose	Category: Calibration Optimization Functions. Recalculates the values of the X- and Y-tables as well as the inverse X- and Y-tables (for Ct5) based on a given input of target points and measured points.	
Function signature	`uint32_t slcl_xy_calibration_mm_targets(size_t Handle, uint16_t NPoints, const double* XTargetsMM, const double* YTargetsMM, const double* XMeasurementsMM, const double* YMeasurementsMM, const slcl_xy_calibration_settings* OtherParameters, slcl_xy_calibration_interpolation_results* InterpolationResults, const char* NewFilename)`	
Argument(s)	`Handle`	Handle for the Correction table to be improved.
	`NPoints`	Number of measured points.
	`XTargetsMM`	Pointer to an array of dimension `NPoints`. This array must contain the targeted x values in mm.
	`YTargetsMM`	Pointer to an array of dimension `NPoints`. This array must contain the targeted y values in mm.
	`XMeasurementsMM`	Pointer to an array of dimension `NPoints`. This array must contain the measured x values in mm.
	`YMeasurementsMM`	Pointer to an array of dimension `NPoints`. This array must contain the measured y values in mm.
	`OtherParameters`	Pointer to a struct `slcl_xy_calibration_settings` containing the parameters for the interpolation.
	`InterpolationResults`	Pointer to a struct `slcl_xy_calibration_interpolation_results` containing additional information about the interpolation results.
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the file is Ctb and not 3d, parameters from the Readme file are needed!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

Name of the function	slcl_xy_calibration_mm_targets_callback	
Purpose	Category: Calibration Optimization Functions. Recalculates the values of the X- and Y-tables as well as the inverse X- and Y-tables (for Ct5) based on a given input of target points and measured points.	
Function signature	`uint32_t slcl_xy_calibration_mm_targets_callback(size_t Handle, uint16_t NPoints, const double* XTargetsMM, const double* YTargetsMM, const double* XMeasurementsMM, const double* YMeasurementsMM, const slcl_xy_calibration_settings* OtherParameters, slcl_xy_calibration_interpolation_results* InterpolationResults, void (*Callback)(const char*, uint32_t, void*) Callback, void* Context, const char* NewFilename)`	
Argument(s)	`Handle`	Handle for the Correction table to be improved.
	`NPoints`	Number of measured points.
	`XTargetsMM`	Pointer to an array of dimension `NPoints`. This array must contain the targeted x values in mm.
	`YTargetsMM`	Pointer to an array of dimension `NPoints`. This array must contain the targeted y values in mm.
	`XMeasurementsMM`	Pointer to an array of dimension `NPoints`. This array must contain the measured x values in mm.
	`YMeasurementsMM`	Pointer to an array of dimension `NPoints`. This array must contain the measured y values in mm.
	`OtherParameters`	Pointer to a struct containing the parameters for the interpolation.
	`InterpolationResults`	Pointer to a struct containing additional information about the interpolation results.
	`Callback`	Function pointer to a callback function which informs about the progress of the calculation. It contains a message, a progress percentage value, and a context pointer.
	`Context`	Pointer to the context of the callback function.
	`NewFilename`	Name of the file that the result shall be saved to. If NULL, the result is not saved.
Return value	See Standard return values of CalibrationLibrary functions, page 62.	
Comment(s)	• If the file is Ctb and not 3d, parameters from the Readme file are needed!	
Code example	–	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	
References	–	

4 Structures struct

In this Chapter:

Name of the structure	slcl_abc_pol_coefficients		
Description	This structure defines: • The coefficients of an ABC polynomial		
Used by	This structure is used with: • slcl_get_current_abc_coeffs • slcl_do_abc_calibration • slcl_set_abc_manually		

Syntax	`struct slcl_abc_pol_coefficients`
	`{`
	`  double  OffsetTermA;`
	`  double  LinTermB;`
	`  double  QuadTermC;`
	`};`		

Argument(s)	`double`	`OffsetTermA`	Offset coefficient A.
	`double`	`LinTermB`	Linear coefficient B.
	`double`	`QuadTermC`	Square coefficient C.
Comment(s)	• –		
Version info	Available as of CalibrationLibrary DLL V1.0.0.		

Name of the structure	**slcl_additional_transformations**		
Description	This structure defines: • Parameters for “additional” transformations That is, for calculating also such transformations which otherwise can be done only by means of RTC commands (on top of the Correction file transformation).		
Used by	This structure is used with: • **slcl_inverse_transform_points_3d_io_with_trafos** • **slcl_inverse_transform_points_3d_with_trafos**		

Syntax	`struct slcl_additional_transformations`
	`{`
	`  double  PreOffsetX;`
	`  double  PreOffsetY;`
	`  double  GainX;`
	`  double  GainY;`
	`  double  Defocus;`
	`  double  PostOffsetX;`
	`  double  PostOffsetY;`
	`  double  PostOffsetZ;`
	`  double  MatrixXX;`
	`  double  MatrixXY;`
	`  double  MatrixYX;`
	`  double  MatrixYY;`
	`};`		

Argument(s)	`double`	`PreOffsetX`	Before Correction file transformation: Added to `InputX`
	`double`	`PreOffsetY`	Before Correction file transformation: Added to `InputY`.
	`double`	`GainX`	Before Correction file transformation: Multiplied with the resulting x value.
	`double`	`GainY`	Before Correction file transformation: Multiplied with the resulting y value.
	`double`	`Defocus`	During Correction file transformation: Subtracted from the focal shift.
	`double`	`PostOffsetX`	After Correction file transformation: Added to `ResultX`.
	`double`	`PostOffsetY`	After Correction file transformation: Added to `ResultY`.
	`double`	`PostOffsetZ`	After Correction file transformation: Added to `ResultZ`.

Name of the structure	slcl_additional_transformations		
Argument(s) (cont'd)	double	MatrixXX	x Matrix coefficient by which the resulting x value is multiplied.
	double	MatrixXY	x Matrix coefficient by which the resulting y value is multiplied.
	double	MatrixYX	y Matrix coefficient by which the resulting x value is multiplied.
	double	MatrixYY	y Matrix coefficient by which the resulting y value is multiplied.
Comment(s)	• –		
Version info	Available as of CalibrationLibrary DLL V1.1.0.		

Name of the structure	**slcl_ct5_table_readme_parameters**	
Description	This structure defines: • Extra parameters needed for certain calibration tools for ct5-Correction files	
Used by	This structure is used with: • **slcl_set_ct5_parameters_manually**	

Syntax	`struct slcl_ct5_table_readme_parameters`
	`{`
	`  double  ZRange;`
	`  double  F1;`
	`  double  F2;`
	`  double  F3;`
	`  double  F4;`
	`};`	

Argument(s)	`double`	`ZRange`	Z-Range of the system. Corresponds to the “Max. Z-Range“ entry in the ct5 Readme file.
	`double`	`F1`	Distortion coefficient of the f-theta objective. Corresponds to the “f1” entry in the ct5 Readme file.
	`double`	`F2`	Distortion coefficient of the f-theta objective. Corresponds to the “f2” entry in the ct5 Readme file.
	`double`	`F3`	Distortion coefficient of the f-theta objective. Corresponds to the “f3” entry in the ct5 Readme file.
	`double`	`F4`	Distortion coefficient of the f-theta objective. Corresponds to the “f4” entry in the ct5 Readme file.
Comment(s)	• –	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	

Name of the structure	slcl_stretch_factors	
Description	This structure defines: • The 2 values for the stretch factors	
Used by	This structure is used with: • **slcl_do_stretch_calibration** • **slcl_get_current_stretch_factors** • **slcl_set_stretch_factors_manually**	

Syntax	`struct slcl_stretch_factors`
	`{`
	`  double  XStretchFactor;`
	`  double  YStretchFactor;`
	`};`	

Argument(s)	`double` `XStretchFactor`	Stretch factor in x direction.
	`double` `YStretchFactor`	Stretch factor in y direction.
Comment(s)	• –	
Version info	Available as of CalibrationLibrary DLL V1.0.0.	

Name of the structure	**slcl_xy_calibration_interpolation_results**	
Description	This structure defines: • Extra information about the results of the interpolation	
Used by	This structure is used with: • **slcl_xy_calibration_bit_targets** • **slcl_xy_calibration_bit_targets_callback** • **slcl_xy_calibration_mm_targets** • **slcl_xy_calibration_mm_targets_callback**	

Syntax	`struct slcl_xy_calibration_interpolation_results`
	`{`
	`  double  MaxDeviationOverallMM;`
	`  double  StdDeviationOverallMM;`
	`  double  StdDeviationXMM;`
	`  double  StdDeviationYMM;`
	`  double  RegularizationFactor;`
	`  double  NewXMinMM;`
	`  double  NewXMaxMM;`
	`  double  NewYMinMM;`
	`  double  NewYMaxMM;`
	`  uint32_t  MaxPolyOrder;`
	`  uint32_t  GalvoLimitsReached;`
	`};`	

Argument(s)	`double` `MaxDeviationOverallMM`	Largest deviation of the interpolation fit to any one measured point.
	`double` `StdDeviationOverallMM`	Standard deviation of the measured points to the interpolation fit.
	`double` `StdDeviationXMM`	Standard deviation of the measured points to the interpolation fit for the x coordinate only.
	`double` `StdDeviationYMM`	Standard deviation of the measured points to the interpolation fit for the y coordinate only.
	`double` `RegularizationFactor`	Regularization factor used for the thin plate spline interpolation.
	`double` `NewXMinMM`	Left-most coordinate of the calculated working field.
	`double` `NewXMaxMM`	Right-most coordinate of the calculated working field.
	`double` `NewYMinMM`	Bottom coordinate of the calculated working field.
	`double` `NewYMaxMM`	Top coordinate of the calculated working field.

Name of the structure	slcl_xy_calibration_interpolation_results		
Argument(s) (cont'd)	uint32_t	MaxPolyOrder	Degree of the base polynomial used for interpolation.
	uint32_t	GalvoLimitsReached	If the galvanometer scanner limits are reached during the calculation of the x table and y table, this is set to 1, otherwise 0.
Comment(s)	• –		
Version info	Available as of CalibrationLibrary DLL V1.0.0.		

Name of the structure	slcl_xy_calibration_settings		
Description	This structure defines: • This structure defines extra parameters for XY Calibration		
Used by	This structure is used with: • **slcl_xy_calibration_bit_targets** • **slcl_xy_calibration_bit_targets_callback** • **slcl_xy_calibration_mm_targets** • **slcl_xy_calibration_mm_targets_callback**		

Syntax	`struct slcl_xy_calibration_settings`
	`{`
	`  uint32_t  XYCalibrationOptions;`
	`  uint32_t  NewCalibrationFactor;`
	`  double  RestrictionScaling;`
	`  double  ToleranceUM;`
	`  uint32_t  MaxFitOrder;`
	`};`		

Argument(s)	`uint32_t`	`XYCalibrationOptions`	Bit field containing **slcl_xy_calibration_options**.
	`uint32_t`	`NewCalibrationFactor`	The desired calibration factor for the new file. Irrelevant if **slcl_xy_calibration_options** `SET_MANUAL_CALIBRATION` is not set. If it is set and `NewCalibrationFactor` is 0, the calibration factor is automatically adjusted to the smallest value that does not exceed galvanometer scanner limits.
	`double`	`RestrictionScaling`	Factor by which the restricted area is enlarged. Irrelevant if **slcl_xy_calibration_options** `RESTRICT_CORRECTION_FILE` is not set in `XYCalibrationOptions`.
	`double`	`ToleranceUM`	Distance by which the interpolation fit may deviate from the measured points. In µm. Irrelevant if **slcl_xy_calibration_options** `USE_AUTO_TOLERANCE` is set in `XYCalibrationOptions`.

Name of the structure	slcl_xy_calibration_settings		
Argument(s) (cont'd)	uint32_t	MaxFitOrder	≥ V1.4.0. Only relevant, if USE_MAX_FIT_ORDER from enum **slcl_xy_calibration_options** is set. The highest degree of the base polynomial to be used for interpolation. Allowed values: 1…9 (default value). Other values are interpreted as '9'. Background: With xy calibration, CalibrationLibrary DLL performs an interpolation. A polynomial is determined as part of this interpolation. Its degree can be at most 9. By MaxFitOrder, the possible degree can even be set lower. The finally determined polynomial degree is returned by MaxPolyOrder from struct **slcl_xy_calibration_interpolation_results**. Accordingly, the following applies: MaxPolyOrder ≤ MaxFitOrder.
Comment(s)	• –		
Version info	Available as of CalibrationLibrary DLL V1.0.0. Changed in CalibrationLibrary DLL V1.4.0. MaxFitOrder.		

Name of the structure	VersionInfo		
Description	This structure defines: • The 3 number blocks of the currently running CalibrationLibrary DLL-Version (”Version n.n.n”)		
Used by	This structure is used with: • **slcl_get_lib_version**		

Syntax	`struct VersionInfo`
	`{`
	`  uint32_t  Major;`
	`  uint32_t  Minor;`
	`  uint32_t  Revision;`
	`};`		

Argument(s)	`uint32_t`	`Major`	Major-Version der CalibrationLibrary DLL.
	`uint32_t`	`Minor`	Minor-Version der CalibrationLibrary DLL.
	`uint32_t`	`Revision`	Revision der CalibrationLibrary DLL.
Comment(s)	• –		
Version info	Available as of CalibrationLibrary DLL V1.0.0.		

5 Enumerated Types enum

In this chapter:

- enum **slcl_error_codes**
- enum **slcl_xy_calibration_options**

Name of the enum	slcl_error_codes	
Description	This enum defines the choices for: • Standard return values of CalibrationLibrary functions	
Used by	This enum is used with: • All CalibrationLibrary-functions except **slcl_get_lib_version**.	

Syntax	`enum slcl_error_codes`
	`{`
	`  NO_ERROR = 0,`
	`  ACTIVATION_CODE_INVALID = 1,`
	`  LIB_ACCESS_DENIED = 2,`
	`  WRONG_FILE_EXTENSION = 3,`
	`  COULD_NOT_OPEN_CORR_FILE = 4,`
	`  WRONG_FILE_SIZE = 5,`
	`  CHECKSUM_INVALID = 6,`
	`  BAD_HANDLE = 7,`
	`  WRONG_TABLETYPE = 8,`
	`  TABLE_NOT_3D = 9,`
	`  CALCULATION_FAILED = 10,`
	`  MISSING_README_PARAMS = 11,`
	`  INSUFFICIENT_MEMORY = 12,`
	`  INTERPOLATION_FAILED = 13,`
	`  SPLINE_INVERSION_FAILED = 14,`
	`  BAD_DIRECTION_VECTOR = 15,`
	`  RADIUS_TOO_SMALL = 16,`
	`  FUNCTION_CALL_NOT_ALLOWED = 17,`
	`  ANGLE_OUT_OF_BOUNDS = 18`
	`};`	

Enumeration constant(s)	`NO_ERROR`	Call successful.
	`ACTIVATION_CODE_INVALID`	**slcl_activate** was called with the wrong password.
	`LIB_ACCESS_DENIED`	**slcl_activate** was never successfully called, access denied.
	`WRONG_FILE_EXTENSION`	Submitted file name has neither ".ctb" nor ".ct5" extension.
	`COULD_NOT_OPEN_CORR_FILE`	Correction file could not be opened, possibly because it does not exist.
	`WRONG_FILE_SIZE`	Specified file does not have the correct size.
	`CHECKSUM_INVALID`	Specified file does not have a valid checksum.
	`BAD_HANDLE`	The provided Handle is zero.

Name of the enum	slcl_error_codes	
Enumeration constant(s) (cont'd)	WRONG_TABLETYPE	The provided Handle is Ctb when Ct5 was expected or vice versa.
	TABLE_NOT_3D	The provided Handle is a 2D-Correction table but must be a 3D-Correction table.
	CALCULATION_FAILED	Calculation failed for miscellaneous reasons.
	MISSING_README_PARAMS	Parameters from the Readme file are missing, likely because a wrong path was provided during initialization.
	INSUFFICIENT_MEMORY	Not enough memory for calculation.
	INTERPOLATION_FAILED	XY Calibration interpolation failed.
	SPLINE_INVERSION_FAILED	XY Calibration spline inversion failed.
	BAD_DIRECTION_VECTOR	Direction vectors are either not perpendicular, or one has no x- and y-component.
	RADIUS_TOO_SMALL	Radius provided for Cylinder Calibration or Cone Calibration is not positive.
	FUNCTION_CALL_NOT_ALLOWED	The provided password does not authorize the use of this function.
	ANGLE_OUT_OF_BOUNDS	The specified ConeInclinationAngleRad value (of **slcl_do_cone_calibration**) is either zero or not between $\pm\pi/3$.
Version info	Available as of CalibrationLibrary DLL V1.0.0.	

Name of the enum	slcl_xy_calibration_options	
Description	This enum defines the choices for: • Boolean options for XY Calibration to be used in a bit field	
Used by	This enum is used with: • struct **slcl_xy_calibration_settings**	

Syntax	`enum slcl_xy_calibration_options`
	`{`
	`  RESTRICT_CORRECTION_FILE = 0x0001,`
	`  USE_POLYGON_RESTRICTION = 0x0002,`
	`  DO_AUTOMATIC_CALIBRATION_TO_RESTRICTION = 0x0004,`
	`  SET_CENTER_OFFSET_TO_ZERO = 0x0008,`
	`  USE_IMPROVE_OLD_FILE_MODE = 0x0010,`
	`  SET_MANUAL_CALIBRATION = 0x0020,`
	`  USE_AUTO_TOLERANCE = 0x0040,`
	`  FASTER_RUNTIME_FIND_FIT_ORDER = 0x0080,`
	`  USE_MAX_FIT_ORDER = 0x0100,`
	`};`	

Enumeration constant(s)	`RESTRICT_CORRECTION_FILE`	If set: Resulting Correction file will be restricted to a certain area given by the measured points.
	`USE_POLYGON_RESTRICTION`	If set: Restricted area will be a convex polygon around measured points, otherwise a rectangle. Irrelevant if `RESTRICT_CORRECTION_FILE` is not set.
	`DO_AUTOMATIC_CALIBRATION_TO_RESTRICTION`	If set: Calibration Factor is automatically set so that the maximum bit control value corresponds to the edge of the restricted area. Irrelevant if `RESTRICT_CORRECTION_FILE` is not set.
	`SET_CENTER_OFFSET_TO_ZERO`	If set: Values at x=0, y=0 are unchanged for all calculated tables.
	`USE_IMPROVE_OLD_FILE_MODE`	If set: New table is calculated by improving the old table, otherwise table is calculated from scratch.
	`SET_MANUAL_CALIBRATION`	If set: Calibration factor is set to value provided under `NewCalibrationFactor`, otherwise the previous calibration factor is used. Important: This is overridden if `DO_AUTOMATIC_CALIBRATION_TO_RESTRICTION` and `RESTRICT_CORRECTION_FILE` are set to `true`!
	`USE_AUTO_TOLERANCE`	≥ V1.1: Deprecated. The optimal fit order is now always determined automatically. ≤ V1.0: If set: XY Calibration will automatically determine the quality of the measurements and adjust the fit tolerance accordingly.

Name of the enum	slcl_xy_calibration_options	
Enumeration constant(s) (cont'd)	FASTER_RUNTIME_FIND_FIT_ORDER	≥ V1.4.0. If set: A different algorithm is used to determine the optimal fit order. It is faster, but the calibration result may be less accurate. Therefore, set this bit only if the calculation time of the calibration step has a higher priority than the accuracy of the calibration.
	USE_MAX_FIT_ORDER	≥ V1.4.0. If set: MaxFitOrder from struct **slcl_xy_calibration_settings**, Page 58 is applied.
Version info	Available as of CalibrationLibrary DLL V1.0.0. Changed in CalibrationLibrary DLL V1.1.0. USE_AUTO_TOLERANCE. Changed in CalibrationLibrary DLL V1.4.0. FASTER_RUNTIME_FIND_FIT_ORDER. USE_MAX_FIT_ORDER.	

6 Example.cpp

```
#include "CalibrationLibrary.h"

#define RETURN_IF_ERROR(Expression) {uint32_t ErrorCode = (Expression); if (ErrorCode != 0) { return ErrorCode; } }
// Simple RAII wrapper for the CalibrationLibrary Handle.
class CalibrationLibraryHandler
{
public:
  // Construct providing the path of the correction file.
  CalibrationLibraryHandler(const char* CorrFilePath)
  {
    LoadError = slcl_load_correction_table(&CalibrationLibraryHandle, CorrFilePath, nullptr);
  }

  CalibrationLibraryHandler(const CalibrationLibraryHandler&) = delete;
  CalibrationLibraryHandler& operator=(const CalibrationLibraryHandler&) = delete;

  CalibrationLibraryHandler(CalibrationLibraryHandler&& In) noexcept
    : CalibrationLibraryHandle(In.CalibrationLibraryHandle)
    , LoadError(In.LoadError)
  {
    In.CalibrationLibraryHandle = 0;
    In.LoadError = 0;
  }
  CalibrationLibraryHandler& operator=(CalibrationLibraryHandler&& In)& noexcept
  {
    if (&In != this)
    {
      CalibrationLibraryHandle = In.CalibrationLibraryHandle;
      LoadError = In.LoadError;
      In.CalibrationLibraryHandle = 0;
      In.LoadError = 0;
    }
    return *this;
  }

  // Automatically clean up the handle if it was successfully constructed previously.
  ~CalibrationLibraryHandler()
  {
    if (CalibrationLibraryHandle != 0)
    {
      slcl_delete_correction_table_handle(CalibrationLibraryHandle);
    }
  }
  // Returns the error code that was returned while loading the correction file.
  // Check this immediately after construction and abort if != 0.
  uint32_t getLoadErrorCode() const
  {
    return LoadError;
  }
  // Returns the successfully constructed Handle.
  size_t get() const
  {
    return CalibrationLibraryHandle;
  }
```

```
private:
  size_t CalibrationLibraryHandle = 0;
  uint32_t LoadError = 0;
};

uint32_t invTransform2dExample()
{
  const uint64_t ActivationCode = 0x0123456789ABCDEFU;
// Replace with the correct activation code..
  const char* Filename = "Folder\\D3_9999.ct5";
// Replace with the correct foldername and the correct filename.

  RETURN_IF_ERROR(slcl_activate(ActivationCode));
// First activate the DLL with the correct password.

  CalibrationLibraryHandle MyHandler(Filename);
  RETURN_IF_ERROR(MyHandler.getLoadErrorCode());

  // points that are to be transformed
  const uint32_t NumberOfPoints = 5U;
  int32_t GalvoBitX[NumberOfPoints] = { -500000, -250000, 0, 250000, 500000 };
// 20-bit values are used for all coordinates.
  int32_t GalvoBitY[NumberOfPoints] = { -400000, -100000, 0, 100000, 400000 };

  // Transform the points - in this case the input arrays will be overwritten with the results.
  return slcl_inverse_transform_points_2d_io(MyHandler.get(), NumberOfPoints, GalvoBitX, GalvoBitY);
}

uint32_t beamTiltCalibrationExample()
{
  const uint64_t ActivationCode = 0x0123456789ABCDEFU;
// Replace with the correct activation code.
  const char* OriginalFilename = "Folder\\D3_9999.ct5";
// Replace with the correct foldername and the correct filename.
  const char* CorrectedFilename = "Folder\\D3_9999_BeamTiltCalibration.ct5";
// Replace with the desired foldername and filename.

  RETURN_IF_ERROR(slcl_activate(ActivationCode));
// First activate the DLL with the correct password.

  CalibrationLibraryHandle MyHandler(OriginalFilename);
  RETURN_IF_ERROR(MyHandler.getLoadErrorCode());
  // Make sure that "D3_9999_ct5_ReadMe.txt" lies in the same folder as the correction file,
  // and if it does not, provide the readme path instead of "nullptr"!

  const double MeasuredOffsetXMM = 3.0;
// Provide the measured offset from zero-position with galvo angles at zero in mm.
  const double MeasuredOffsetYMM = 2.0;

  return slcl_do_beam_tilt_calibration(MyHandler.get(), MeasuredOffsetXMM, MeasuredOffsetYMM, CorrectedFilename);
// call beam tilt calibration
}
```

```
uint32_t xyCalibrationExample()
{
  const uint64_t ActivationCode = 0x0123456789ABCDEFU;
// Replace with the correct activation code.
  const char* OriginalFilename = "Folder\\D3_9999_BeamTiltCalibration.ct5";
// Replace with the correct foldername and the correct filename.
  const char* CorrectedFilename = "Folder\\D3_9999_XYCalibration.ct5";
// Replace with the desired foldername and filename.

  RETURN_IF_ERROR(slcl_activate(ActivationCode));
// First activate the DLL with the correct password.

  CalibrationLibraryHandler MyHandler(OriginalFilename);
  RETURN_IF_ERROR(MyHandler.getLoadErrorCode());

  const uint16_t NumberOfPoints = 25U;
// For this example we'll go with 25 (5x5) measured points, but recommended is >=121 (11x11).

  // Provide the target bits.
  // It is also possible to provide the target positions in mm as double values.
  const int32_t XTargetsBit [NumberOfPoints] =
  {
    -500000, -250000,   0,  250000,  500000,
    -500000, -250000,   0,  250000,  500000,
    -500000, -250000,   0,  250000,  500000,
    -500000, -250000,   0,  250000,  500000,
    -500000, -250000,   0,  250000,  500000,
  };
  const int32_t YTargetsBit[NumberOfPoints] =
  {
    -500000, -500000, -500000, -500000, -500000,
    -250000, -250000, -250000, -250000, -250000,
          0,       0,       0,       0,       0,
     250000,  250000,  250000,  250000,  250000,
     500000,  500000,  500000,  500000,  500000,
  };

  // Then also provide the measured positions in mm.
  const double XMeasuredPositionsMM[NumberOfPoints] =
  {
    -177.56,  -88.78,   0,   88.78,  177.56,
    -177.56,  -88.78,   0,   88.78,  177.56,
    -177.56,  -88.78,   0,   88.78,  177.56,
    -177.56,  -88.78,   0,   88.78,  177.56,
    -177.56,  -88.78,   0,   88.78,  177.56,
  };
  const double YMeasuredPositionsMM[NumberOfPoints] =
  {
    -177.56, -177.56, -177.56, -177.56, -177.56,
     -88.78,  -88.78,  -88.78,  -88.78,  -88.78,
          0,       0,       0,       0,       0,
      88.78,   88.78,   88.78,   88.78,   88.78,
     177.56,  177.56,  177.56,  177.56,  177.56,
  };
```

```
  // Set other parameters for xy calibration.
  slcl_xy_calibration_settings Params;
  Params.XYCalibrationOptions = SET_CENTER_OFFSET_TO_ZERO | USE_IMPROVE_OLD_FILE_MODE;
  Params.NewCalibrationFactor = 0;
  Params.RestrictionScaling = 0.0;
  Params.ToleranceUM = 20.0;

  // Provide a struct to write the additional results into.
  slcl_xy_calibration_interpolation_results Results;

  // Call XY Calibration.
  // If targets were provided in mm, call slcl_xy_calibration_mm_targets instead!
  return slcl_xy_calibration_bit_targets(MyHandler.get(),
       NumberOfPoints,
       XTargetsBit,
       YTargetsBit,
       XMeasuredPositionsMM,
       YMeasuredPositionsMM,
       &Params,
       &Results,
       CorrectedFilename);
}

uint32_t abcCalibrationExample()
{
  const uint64_t ActivationCode = 0x0123456789ABCDEFU;
// Replace with the correct activation code.
  const char* OriginalFilename = "Folder\\D3_9999_XYCalibration.ct5";
// Replace with the correct foldername and the correct filename.
  const char* CorrectedFilename = "Folder\\D3_9999_ABCCalibration.ct5";
// Replace with the desired foldername and filename.

  RETURN_IF_ERROR(slcl_activate(ActivationCode));
// First activate the DLL with the correct password.

  CalibrationLibraryHandler MyHandler(OriginalFilename);
  RETURN_IF_ERROR(MyHandler.getLoadErrorCode());

  const uint16_t NMeasuredPoints = 7U;

  // Provide the measured focal lengths...
  const int32_t FocalLengthMeasurements[NMeasuredPoints] = { -160000, -8000, 0, 8000, 16000, 80000, 160000 };

  // ...and the measured z-output values.
  const int32_t ZOutputMeasurements[NMeasuredPoints] = { -348800, -12880, 3200, 19120, 34880, 155200, 291200 };

  // Also a struct to output the calculated coefficients into.
  // The new coefficients are also saved in the handle, but for ease of access they are written into this
  // struct after the calculation as well.
  slcl_abc_pol_coefficients ABCCorrResults;

  // Call ABCCalibration.
  return slcl_do_abc_calibration(MyHandler.get(), NMeasuredPoints, FocalLengthMeasurements, ZOutputMeasurements,
                                 CorrectedFilename, &ABCCorrResults);
}
```

```
uint32_t cylinderCalibrationExample()
{
  const uint64_t ActivationCode = 0x0123456789ABCDEFU;
// Replace with the correct activation code.
  const char* OriginalFilename = "Folder\\D3_9999.ct5";
// Replace with the correct foldername and the correct filename.
  const char* CorrectedFilename = "Folder\\D3_9999_CylinderCalibration.ct5";
// Replace with the desired foldername and filename.

  RETURN_IF_ERROR(slcl_activate(ActivationCode));
// First activate the DLL with the correct password.

  CalibrationLibraryHandler MyHandler(OriginalFilename);
  RETURN_IF_ERROR(MyHandler.getLoadErrorCode());

  // Example: Fit correction file to coffee mug.

  const double CoffeeMugDiameter = 81.2;

  // The new coordinate origin shall not be shifted in x/y direction.
  // However, the mug is simply placed on the z=0 plane, meaning that the
  // z coordinate at the coordinate origin is equal to its diameter.
  const double RefPoint[3] = { 0.0, 0.0, CoffeeMugDiameter};
  // The cylinder is placed parallel to the x-axis.
  const double DirectionVector[3] = { 1.0, 0.0, 0.0 };
  const double RadiusCoffeeMug = CoffeeMugDiameter * 0.5;

  // Call Cylinder Calibration.
  return slcl_do_cylinder_calibration(MyHandler.get(), RefPoint, DirectionVector, RadiusCoffeeMug, CorrectedFilename);
}

uint32_t focusCorrExample()
{
  const uint64_t ActivationCode = 0x0123456789ABCDEFU;
// Replace with the correct activation code - will be provided by SCANLAB GmbH
  const char* OriginalFilename = "Folder\\D3_9999_XYCalibration.ct5";
// Replace with the correct foldername and the correct filename.
  const char* CorrectedFilename = "Folder\\D3_9999_FocusCalibration.ct5";
// Replace with the desired foldername and filename.

  RETURN_IF_ERROR(slcl_activate(ActivationCode));
// First activate the DLL with the correct password.

  CalibrationLibraryHandler MyHandler(OriginalFilename);
  RETURN_IF_ERROR(MyHandler.getLoadErrorCode());

  const uint16_t NumberOfPoints = 25U;
// For this example we'll go with 25 (5x5) measured points. It may be recommendable to use more.

  // Provide the targeted xy-bits.
  // The coordinates must be given in respect to the coordinate system of the target plane.
  const int32_t XTargetedPositionsBit[NumberOfPoints] =
  {
    -500000, -250000,   0,  250000,  500000,
    -500000, -250000,   0,  250000,  500000,
    -500000, -250000,   0,  250000,  500000,
    -500000, -250000,   0,  250000,  500000,
    -500000, -250000,   0,  250000,  500000,
  };
```

```
  const int32_t YTargetedPositionsBit[NumberOfPoints] =
  {
    -500000, -500000, -500000, -500000, -500000,
    -250000, -250000, -250000, -250000, -250000,
    0,    0,    0,    0,    0,
    250000,  250000,  250000,  250000,  250000,
    500000,  500000,  500000,  500000,  500000,
  };

  // Then also provide the z-bit values that gave the best spot size:
  // To determine the optimal z-bit values use the RTC-command load_z_table followed by a select_cor_table command.
  int32_t MeasuredOptimalZBits[NumberOfPoints] =
  {
    282843, 223607, 200000,  223607,  282843,
    223607, 141421, 100000,  141421,  223607,
    200000, 100000,  0,  100000,  200000,
    223607, 141421, 100000,  141421,  223607,
    282843, 223607, 200000,  223607,  282843,
  };

  //  The suggested RTC command procedure to find the values in the array above looks like this:
  //  ...
  //  load_correction_file(OriginalFilename, 1, 3); //load the correction file
  //  ...
  //  For all grid points X = XTargetedPositionsBit[i], Y = XTargetedPositionsBit[i] :
  //  load_z_table(MeasuredOptimalZBits[i], 0, 0); // changes A value in the ABC correction and therefore the focus position offset
  //  select_cor_table(1, 0); // applies the new A value
  //  repeat get_status(Busy, Pos) until Busy = 0; // wait for the select_cor_table done
  //  goto_xyz(X, Y, 0); //moves spot to the targeted X and Y position at Z=0(!) with modified Z offset
  //  // repeat the above while varying MeasuredOptimalZBits[i] until focal position is OK at the grid point X and Y
  //

  // Call Focus Calibration.
  return slcl_do_focus_calibration(MyHandler.get(),
    NumberOfPoints,
    XTargetedPositionsBit,
    YTargetedPositionsBit,
    MeasuredOptimalZBits,
    CorrectedFilename);
}
```

7 Change Index

The following are changes in this manual due to the technical evolution of the product as well as significant editorial changes.

In this Chapter:

Changes to Document Revision 0.9.0 en-US from Document Revision 0.0.0 en-US

Where (cont'd.)	**What (cont'd.)**
Global	Document Revision • 0.9.0 en-US applies to Calibration Library software package • V1.0.0
Global	First published version.
Change Index, page 72	

Changes to Document Revision 1.0.0 en-US from Document Revision 0.9.0 en-US

Where	What
Global	Document Revision • 1.0.0 en-US applies to Calibration Library software package[a] • V1.1.0
slcl_inverse_transform_points_3d_io_with_trafos, Page 38	Software change. New function.
slcl_inverse_transform_points_3d_with_trafos, Page 39	Software change. New function.
slcl_additional_transformations, Page 52	Software change. New struct.
slcl_xy_calibration_options, Page 64	Software change. Changed enum **slcl_xy_calibration_options**. Enumeration constant `USE_AUTO_TOLERANCE`. ≥ V1.1: Deprecated. The optimal fit order is now always determined automatically.
Change Index, page 72	

(a) #148051

Changes to Document Revision 1.1.0 en-US from Document Revision 1.0.0 en-US

Where	What
Global	Document Revision • 1.1.0 en-US applies to Calibration Library software package #148051 • V1.2.0
Change Index, page 72	

Changes to Document Revision 1.2.0 en-US from Document Revision 1.1.0 en-US

Where	What
Global	Document Revision • 1.2.0 en-US applies to Calibration Library software package #148051 • V1.4.0
slcl_do_beam_tilt_calibration_measurement_data, Page 20	See Version info, page 20.
struct **slcl_xy_calibration_interpolation_results**, Page 56	Editorial change. Correction: • `MaxPolyOrder` is of data type `uint32_t` (not: `int32_t`) • `GalvoLimitsReached` is of data type `uint32_t` (not: `int32_t`)
struct **slcl_xy_calibration_settings**, Page 58	Editorial change. Correction: • `XYCalibrationOptions` is of data type `uint32_t` (not: `int32_t`) • `NewCalibrationFactor` is of data type `uint32_t` (not: `int32_t`)
struct **slcl_xy_calibration_settings**, Page 58	Software change. See Version info, page 59.
enum **slcl_xy_calibration_options**, Page 64	Software change. See Version info, page 65.
Change Index, page 72	