



syncAXIS-DLL – Application Programming Interface

syncAXIS control **V1.8.0**

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

© SCANLAB GmbH

(Doc. Rev. 1.9.20 en-US - 2022-11-16)

SCANLAB GmbH reserves the right to change the information in this document without notice.

No part of this document may be processed, reproduced or distributed in any form (photocopy, print, microfilm or by any other means), electronic or mechanical, for any purpose without the written permission of SCANLAB GmbH.

All mentioned trademarks are hereby acknowledged as properties of their respective owners.

Contents

1	About this Manual	9
1.1	Related Documents	9
1.2	Manufacturer	9
1.3	Overview	10
1.4	Glossary	11
2	Software Development with the syncAXIS-DLL	16
2.1	Safety	16
2.2	About the SAFE Use of syncAXIS control – General Approach	18
	syncAXIS control Safety Features	18
2.2.1	Identifying System Limits	19
2.2.2	Establishing Safety Mechanisms	19
2.2.3	Configuring Safe syncAXIS control Instances	20
2.2.4	Simulating and Improving Jobs	24
2.3	About the Main Structures of a syncAXIS-DLL-Based User Program (Exemplary)	25
2.3.1	Structure to Comply with when Defining Jobs	25
2.4	About Initializing syncAXIS control-based User Programs	26
2.5	About the syncAXIS control Simulation Mode	31
2.6	About Optimizing syncAXIS control-based User Programs	36
2.6.1	Possible Optimizations	36
2.6.2	Iterative Approach	38
2.7	About Processes at Run Time of the User Program	41
2.7.1	About the Buffers of the syncAXIS control Instances	42
	Avoiding Buffer Underruns	42
2.7.2	About the Point in Time when Output Signals are actually set	45
2.8	About the Logging in syncAXIS control	47
2.9	About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")	48
2.9.1	Activation of the "Automatic Laser Control"	48
2.9.2	Definition of the Channels and ActiveChannel	48
2.9.3	About how ActiveChannel Values along a Contour are Calculated	51
2.9.4	About Ramps	53
	Example – Linear (Simple) Ramp	54
	Example – Multi-part (more Complex) Ramp	57
2.9.5	About the "Contour-dependent speed calculation"	60
2.10	About Heuristic and Characteristics for Speed Reductions	64
2.11	About Working with "Modules"	65
2.12	About the Mode "Manual Positioning"	70
2.12.1	Allowed/Not Allowed syncAXIS control Functions	71
2.12.2	Example – Temporarily Releasing the Positioning Stage and Changing the Target Positioning Stage	74
3	Functions Available in the API	76
3.1	Functional Overview	76
3.1.1	Configuration Functions (slsc_cfg_*)	76
	syncAXIS control instance-related Functions	78
	Functions for Changing the Configuration of the Present syncAXIS-DLL Instance	80
	Functions for Registering "Callback Events"	81
3.1.2	Job Functions (slsc_list_*)	85
	Functions for Defining Job-Beginnings/Ends	85
	Functions for Defining Jumps	85

Functions for Defining Markings	87
[*]dashed[*] Functions	91
Functions for Changing Target Point Coordinates	92
Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)	92
Functions for Setting Signals	93
Functions for Changing Speeds	93
Function for Changing Minimum Speeds	93
Functions for Changing Trajectory planning Values	94
Functions for Changing the Behavior of Blending Curves	94
Function for the "Contour-dependent Speed Calculation"	94
Function for Setting the Value of a Free Variable on the RTC6	95
Function for Influencing the Laser Pulse Output by HalfPeriod/PulseLength	95
Functions for "Modules"	95
3.1.3 Control Functions (slsc_ctrl_*)	96
Laser-related Functions	96
Execution-related Functions	97
Correction File-related Functions	99
Error-related Functions	99
Functions for Querying Measured Values	99
Functions Only for Mode "Manual Positioning"	99
Functions for Managing the Value of a Free Variable on the RTC6	99
Functions for Optimizing Parameter Values	100
Functions for Starting/Ending the Mode "Manual Positioning"	100
Functions for Querying Positions	100
Simulation Setting-related Function	100
Functions for Setting Signals	100
Function for Influencing the Laser Pulse Output by HalfPeriod/PulseLength	100
3.1.4 Utility Functions (slsc_util_*)	101
RTC6 board-related Function	101
3.2 Alphabetical Overview	102
3.3 Function Reference	114
3.3.1 General Structure of the Reference Tables	114
3.3.2 Data Types of the syncAXIS-DLL Functions	115
3.3.3 Reference Tables	117
slsc_cfg_acquire_stage (deprecated)	117
slsc_cfg_delete	118
slsc_cfg_delete_trajectory_config	119
slsc_cfg_get_calculation_dynamics_jump_scan_device	120
slsc_cfg_get_calculation_dynamics_mark_scan_device	121
slsc_cfg_get_calculation_dynamics_stage	122
slsc_cfg_get_dynamic_limits_scan_device	123
slsc_cfg_get_dynamic_limits_stage	124
slsc_cfg_get_dynamic_violation_reaction	125
slsc_cfg_get_field_limits_scan_device	126
slsc_cfg_get_field_limits_stage	127
slsc_cfg_get_jump_time	128
slsc_cfg_get_mode	130
slsc_cfg_get_operation_status	131
slsc_cfg_get_scan_device_dynamic_monitoring_level	132
slsc_cfg_get_simulation_setting	133
slsc_cfg_get_stage_dynamic_monitoring_level	134
slsc_cfg_get_sync_axis_version	135

slsc_cfg_get_trajectory_config	135
slsc_cfg_initialize_copy	136
slsc_cfg_initialize_from_file	137
slsc_cfg_register_callback_job_end_planned	139
slsc_cfg_register_callback_job_finished_executing	140
slsc_cfg_register_callback_job_is_executing	141
slsc_cfg_register_callback_job_loaded_enough	142
slsc_cfg_register_callback_job_progress_planned	143
slsc_cfg_register_callback_job_start_planned	144
slsc_cfg_reinitialize	145
slsc_cfg_reinitialize_from_file	147
slsc_cfg_release_stage (deprecated)	149
slsc_cfg_select_heuristic	152
slsc_cfg_select_stage	153
slsc_cfg_select_stage_axis (deprecated)	154
slsc_cfg_set_bandwidth	155
slsc_cfg_set_calculation_dynamics_jump_scan_device	156
slsc_cfg_set_calculation_dynamics_mark_scan_device	158
slsc_cfg_set_calculation_dynamics_stage	160
slsc_cfg_set_contour_dependent_speed_control_2d	162
slsc_cfg_set_dynamic_limits_scan_device	164
slsc_cfg_set_dynamic_limits_stage	165
slsc_cfg_set_dynamic_violation_reaction	166
slsc_cfg_set_field_limits_scan_device	167
slsc_cfg_set_field_limits_stage	168
slsc_cfg_set_jump_speed	170
slsc_cfg_set_list_handling_mode	171
slsc_cfg_set_list_handling_mode_with_context	173
slsc_cfg_set_mark_speed	174
slsc_cfg_set_matrix_and_offset	175
slsc_cfg_set_mode	176
slsc_cfg_set_part_displacement	177
slsc_cfg_set_rot_and_offset_2d	178
slsc_cfg_set_scan_device_dynamic_monitoring_level	179
slsc_cfg_set_simulation_setting	180
slsc_cfg_set_stage_dynamic_monitoring_level	181
slsc_cfg_set_trajectory_config	182
slsc_ctrl_disable_laser	183
slsc_ctrl_enable_laser	184
slsc_ctrl_follow	185
slsc_ctrl_get_error	186
slsc_ctrl_get_error_count	187
slsc_ctrl_get_exec_state	188
slsc_ctrl_get_free_variable	189
slsc_ctrl_get_job_characteristic	190
slsc_ctrl_get_scan_device_position	191
slsc_ctrl_get_simulation_filename	192
slsc_ctrl_get_stage_position	193
slsc_ctrl_get_syncaxis_simulation_filename	194
slsc_ctrl_get_value	195
slsc_ctrl_is_list_input_buffer_full	196
slsc_ctrl_laser_signal_off	197

slsc_ctrl_laser_signal_on	198
slsc_ctrl_move_scanner_abs	199
slsc_ctrl_move_stage_abs	200
slsc_ctrl_refresh_correction_file	201
slsc_ctrl_select_correction_file	202
slsc_ctrl_set_free_variable	203
slsc_ctrl_set_laser_pulses	204
slsc_ctrl_start_execution	205
slsc_ctrl_stop	206
slsc_ctrl_stop_controlled	207
slsc_ctrl_unfollow	208
slsc_ctrl_write_analog_x	209
slsc_ctrl_write_digital_out	210
slsc_ctrl_write_digital_out_mask	211
slsc_list_arc_abs	212
slsc_list_begin	214
slsc_list_begin_absolute	215
slsc_list_begin_module	217
slsc_list_begin_relative	218
slsc_list_circle_2d_abs	220
slsc_list_dashed_arc_abs	221
slsc_list_dashed_circle_2d_abs	223
slsc_list_dashed_mark_abs	225
slsc_list_end	226
slsc_list_jump_abs	227
slsc_list_jump_abs_min_time	228
slsc_list_mark_abs	229
slsc_list_multi_para_arc_abs	230
slsc_list_multi_para_circle_2d_abs	231
slsc_list_multi_para_dashed_arc_abs	232
slsc_list_multi_para_dashed_circle_2d_abs	234
slsc_list_multi_para_dashed_mark_abs	236
slsc_list_multi_para_mark_abs	238
slsc_list_para_arc_abs	239
slsc_list_para_circle_2d_abs	241
slsc_list_para_dashed_arc_abs	243
slsc_list_para_dashed_circle_2d_abs	245
slsc_list_para_dashed_mark_abs	247
slsc_list_para_disable	249
slsc_list_para_enable	250
slsc_list_para_jump_abs	251
slsc_list_para_jump_abs_min_time	252
slsc_list_para_mark_abs	253
slsc_list_para_playback_module	254
slsc_list_playback_module	255
slsc_list_set_approx_blend_limit	257
slsc_list_set_calculation_dynamics_jump_scan_device	258
slsc_list_set_calculation_dynamics_mark_scan_device	259
slsc_list_set_contour_dependent_speed_control_2d	260
slsc_list_set_free_variable	261
slsc_list_set_jump_speed	262
slsc_list_set_laser_on_move	263

slsc_list_set_laser_pulses	264
slsc_list_set_mark_speed	265
slsc_list_set_matrix_and_offset	266
slsc_list_set_min_mark_speed	267
slsc_list_set_rot_and_offset_2d	268
slsc_list_suppress_spotdistance_control	270
slsc_list_unsuppress_spotdistance_control	271
slsc_list_wait_with_laser_off	272
slsc_list_wait_with_laser_on	273
slsc_list_write_analog_x	274
slsc_list_write_digital_out	276
slsc_list_write_digital_out_mask	277
slsc_util_reset_pcie	278
4 Standard Return Values of the syncAXIS-DLL Functions	279
5 Error Codes with slsc_ctrl_get_error, Log File and Console	282
6 Structures	289
slsc_GeometryConfig	289
slsc_MarkConfig	293
slsc_MultiParaTarget	297
slsc_ParaSection	298
slsc_TrajectoryConfig	299
VersionInfo	299
7 Enumerated Types enum	300
slsc_AnalogOutput	300
slsc_BlendModes	301
slsc_DynamicsMonitoringLevel	303
slsc_DynamicViolationReaction	304
slsc_ExecState	305
slsc_JobCharacteristic	306
slsc_ListHandlingMode	312
slsc_MeasurementSignal	313
slsc_OperationMode	314
slsc_OperationStatus	315
slsc_PositionType	315
slsc_ScanDevice	316
slsc_SimulationSetting	317
slsc_SplineModes	318
slsc_Stage	320
8 Appendix A: Using syncAXIS control V1.2.4 and Higher with XL SCAN Multi-Head Systems	321
8.1 About this Appendix	321
8.2 Usage of syncAXIS control V1.2.4 and Higher	324
8.2.1 Prerequisites for this Appendix	324
8.2.2 Adapting syncAXISConfig.xml for syncAXIS control V1.2.4 and Higher	324
Step 1 of 2: Adapting syncAXISConfig.xml Technically	324
Step 2 of 2: Adapting syncAXISConfig.xml in Regards to Content	324
8.2.3 Further Notes on the Use of syncAXIS control V1.2.4 and Higher	331
8.3 About Transformations in syncAXIS control V1.2.4 and Higher	332
9 Appendix B: Application Note – Handling Lists with syncAXIS control	335
9.1 List Handling Mode “ReturnAtOnce”	337
9.2 List Handling Mode “RepeatWhileBufferFull”	339



9.3 List Handling Mode "RepeatWhilePredicate"	343
10 Appendix C: Application Note – Marking Texts by Using Modules	344
11 Appendix D: Application Note – Avoiding Buffer Underruns by Using Modules	347
12 Appendix E: Application Note – C#	350
12.1 Differences in the syncAXIS-DLL function signatures	350
12.1.1 Notation der Datentypen	350
12.1.2 Pointer-Replacements for C#	350
12.2 Differences in the Use of Callback Functions	351
12.3 Code Example 1 (C#)	352
12.4 Code Example 2 (C#)	357
13 Appendix F: Reference of syncAXISConfig.xml Tags	358
13.1 xml-Structure Overview	358
13.2 xml Tags	361
14 Change Index	476

1 About this Manual

This manual describes the SCANLAB syncAXIS-DLL and its usage to develop and to execute user programs for the synchronous control of a laser, a scan head (excelliSCAN only) and a positioning stage in a laser-scan system.

The syncAXIS-DLL (32-bit version and 64-bit version) is a part of the syncAXIS control-software package.

For developing user programs user programs it provides a programming interface (API) in the form of functions. These allow, among other things:

- Extensive configuration possibilities for system parameters as well as for the syncAXIS-DLL itself (calculation of motion data only for the scan heads, only positioning stage, or both)
- Definition of processing **Jobs** (for example, labeling patterns)
- Loading, execution and stopping **Jobs** – also in a simulation mode as file output
- Status monitoring of **Jobs**
- Registering **Callback events**

Precondition for the usage of the syncAXIS-DLL:

- Installed, configured and calibrated hardware (see **Figure 1, page 10**): scan head⁽¹⁾ (excelliSCAN only), positioning stage, **ACS** components, RTC6 PCI Express Board, power supply, cables, PC, **Dongle**
- Operating system: MS Windows 7, 8, 10 (32-bit and 64-bit variants)
- Installed syncAXIS control-software package⁽²⁾
- Installed **ACS** software package

(1) syncAXIS-DLL can be used for scan head calibration.

(2) Among others, the package contains RTC6 files (RTC6DAT.dat, RTC6DLL.dll, RTC6OUT.out, RTC6RBF.rbf). These files must not be replaced by those RTC6 files which have been delivered along with the RTC6 board itself or have been downloaded from the SCANLAB website.

Notice!

Carefully read the document “syncAXIS control Software License Agreement” before installing and using syncAXIS control. 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

- RTC6 Manual
- “Installation of SCANLAB XL SCAN Components and Initial Operation of the XL SCAN System” Manual
- “syncAXIS Viewer” Manual
- “syncAXIS Configurator” Manual
- “syncAXIS Master-Slave-Synchronizer” Manual

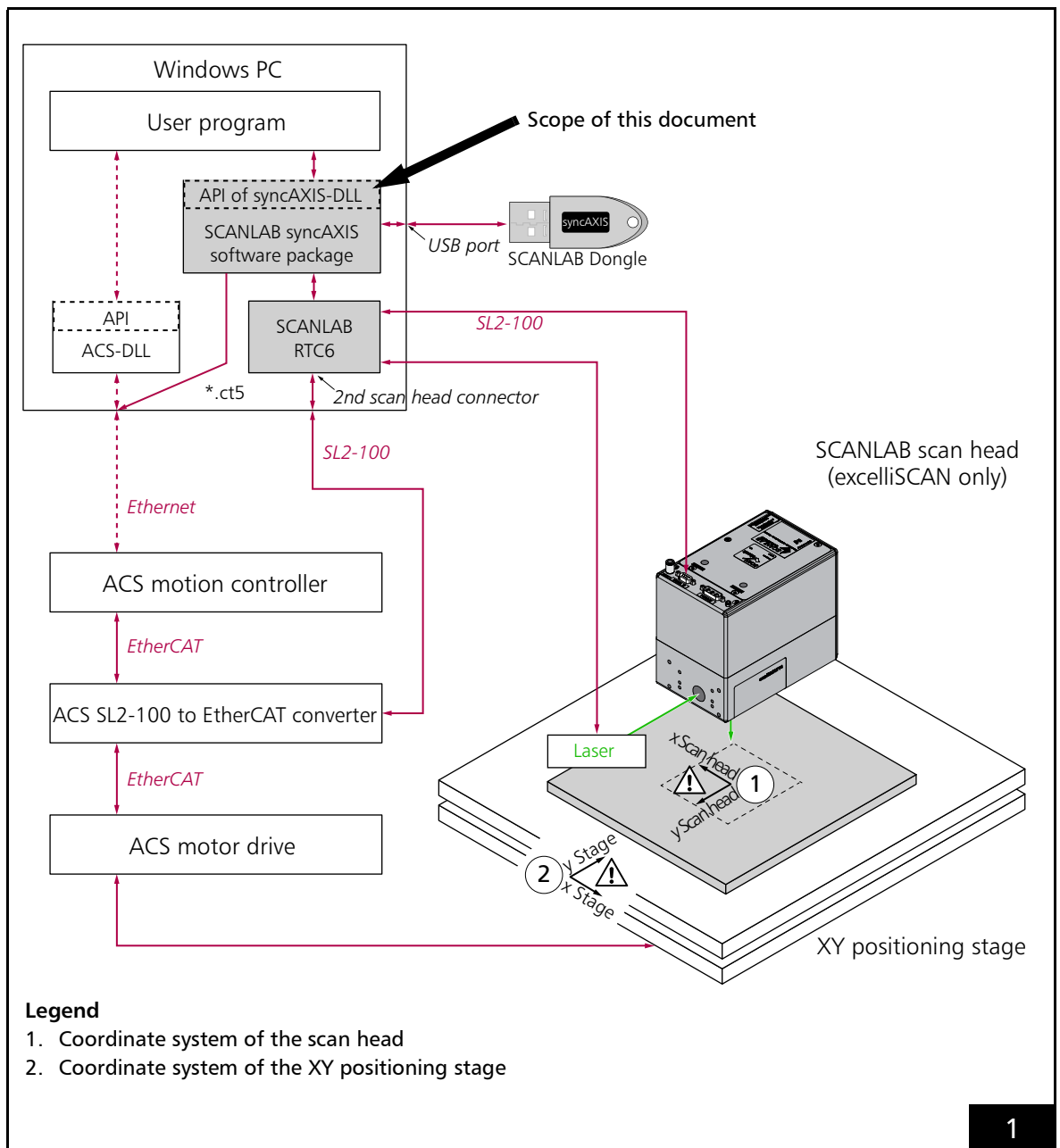
1.2 Manufacturer

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 Overview

A scenario for the application of syncAXIS-DLL (as a part of the syncAXIS control-software package) shows [Figure 1, page 10](#).

The simultaneous control of a 2D scan head and a mechanical XY positioning stage with two servo axes is illustrated. This combined system increases the working area in relation to the scan head working field and the marking results do not exhibit stitching errors.



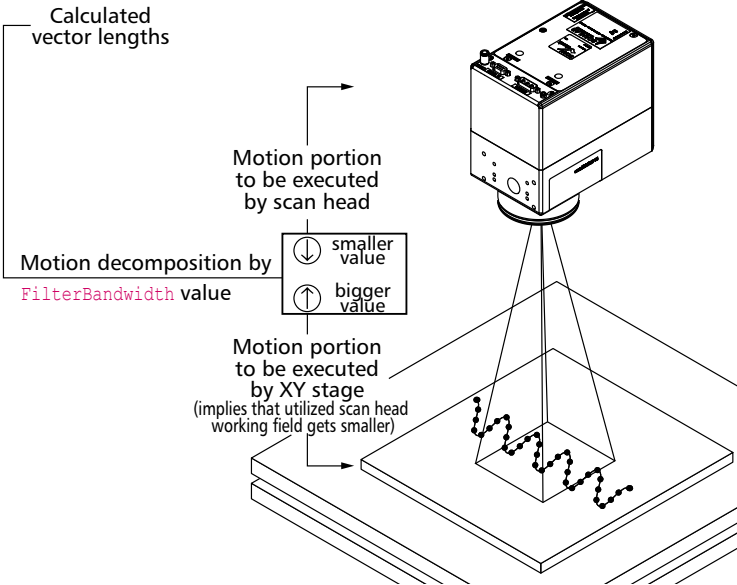
syncAXIS-DLL application scenario. ACS: see [page 11](#).

The illustrated system has 1 positioning stage only. Optionally, an additional positioning stage can be added (requires no additional RTC6 board nor an additional SL2-100 to EtherCAT converter).

1.4 Glossary

ACS	Designates the manufacturer of machine control systems whose components must be used at present.
ACS Motion Controller	XL SCAN component from ACS, see Figure 1, page 10 .
API	Abbreviation of Application Programming Interface. Program part (here: of the syncAXIS-DLL) which is available for other programs for connecting to the system (here: functions of the syncAXIS-DLL). See Chapter 3 "Functions Available in the API", page 76 .
Buffer	Designates syncAXIS-DLL-internal temporary memories. See Chapter 2.7 "About Processes at Run Time of the User Program", page 41 .
Buffer underrun	After execution start, the RTC6 board has processed all RTC6 micro vector commands in its list memory because the syncAXIS-DLL has been too slow in the calculation and transmission of further RTC6 micro vector commands. See also Figure 11, page 41 . A Buffer underrun occurs, for example, when the Input buffer is filled with a lot of data relative to the execution time (that is, short vector segments and high marking speed). New data is not written to the Output buffer in time. As a consequence, the RTC6 board cannot continue to execute its list and sends a static position continuously ("hard stop"). This results in exceeding a system dynamic limit. An Buffer underrun puts XL SCAN into an error state. As a rule, an error at the ACS axes is also detected, because there has been a sudden (acceleration and jerk unrestricted) change in speed (the last approached position is outputted). For possible measures to avoid an Buffer underrun , see Chapter 2.7.1 "About the Buffers of the syncAXIS control Instances", page 42 .
Callback event	One of several syncAXIS-DLL-internal events in Figure 12, page 43 and Table on page 81 that occur when a Job goes through its calculation, transfer and execution status.
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 syncAXIS control instance via a Function for registering "Callback events" . Therefore, it must comply to a dictated function signature (<code>slsc_JobCallback</code> or <code>slsc_ExecTimeCallback</code>).
Configuration function (<code>slsc_cfg_*</code>)	Designates a function in the API with the prefix <code>slsc_cfg_</code> . These functions serve, for example, to manage the configuration of the syncAXIS control instance and to configure Event Callbacks. See Chapter 3.1.1 "Configuration Functions (<code>slsc_cfg_*</code>)", page 76 .
Control function (<code>slsc_ctrl_*</code>)	Designates a function in the API with the prefix <code>slsc_ctrl_</code> . These functions serve to influence the user program flow, for example, to start/to stop the execution, to query errors, to set the jump speed and marking speed etc. See Chapter 3.1.3 "Control Functions (<code>slsc_ctrl_*</code>)", page 96 .
Dongle	Short for "SCANLAB USB dongle for XL SCAN". USB copy protection device which contains unique licensing information, see Multi-Instance and Multi-Stage . No storage device! As of syncAXIS control \geq V1.2, another version is available (#139941 "Office dongle"). This allows Sky Writings to use the software without hardware connection on the local PC, for example, to create simulation files (that is, generate control values).

Execution Layer	syncAXIS-DLL-internal submodule, which is responsible for the communication with the RTC6 board and its monitoring. Its state can be queried with <code>slsc_ctrl_get_exec_state</code> . With an <code>slsc_ctrl_start_execution</code> call, it triggers the Job execution.
Function for registering "Callback events"	One of the syncAXIS-DLL- Configuration functions with prefix <code>slsc_cfg_register_callback_</code> , see page 81 . For every "Callback event" there is a corresponding "Function for registering "Callback events"" where the to-be-executed "Callback function" is specified. Thereby, Job -related information can be captured and responded to.
Handle	Computer programming term: abstract reference to a resource. In this manual, this term refers to a certain syncAXIS control instance (most of the time). Its Handle value is assigned by an Initialization function . With the (most) syncAXIS control functions, the Handle value of the desired target- syncAXIS control instance must be specified.
Heuristic	<p>The following applies to \geq V1.5.0:</p> <ul style="list-style-type: none"> Is a syncAXIS-DLL-internal algorithm Only effective in Operation mode <code>ScannerAndStage</code> Requires at least 1 defined characteristic, see DynamicReductionFunction Evaluates Jump Segments and Mark Segments <ul style="list-style-type: none"> Optionally only Jump Segments, if in <code>syncAXISConfig.xml</code> is set: <pre><cfg:HeuristicForJumpsOnly>true</cfg:HeuristicForJumpsOnly></pre> See also Chapter 2.10 "About Heuristic and Characteristics for Speed Reductions", page 64. <p>The Heuristic:</p> <ol style="list-style-type: none"> (1) Determines the spatial extent of each Segment = distance between starting point and end point (exception: diameter of the circle with circular Segments (<code>[*]arc[*]</code>, <code>[*]circle[*]</code>) which sweep more than a semicircle). (2) Checks the characteristic (= DynamicReductionFunction) if there is a corresponding <code>Velocity</code> value for the result from (1) (= <code>Length</code> value). (3) Only if the <code>Velocity</code> value from (2) is smaller than the original marking speed: the Heuristic reduces the marking speed of the Segment to this <code>Velocity</code> value.
Initialization function	Collective term for syncAXIS-DLL functions which create a syncAXIS control instance : <code>slsc_cfg_initialize_copy</code> and <code>slsc_cfg_initialize_from_file</code> as well as <code>slsc_cfg_reinitialize</code> and <code>slsc_cfg_reinitialize_from_file</code> . See also Chapter 2.4 "About Initializing syncAXIS control-based User Programs" , page 26 .
Input buffer	syncAXIS-DLL-internal temporary memory at the start of the processing chain. See Buffer and Figure 11 , page 41 in Chapter 2.7 "About Processes at Run Time of the User Program" , page 41 .
Job	Designates a mandatory sequence of Job functions . See also Section "Structure to Comply with when Defining Jobs" , page 25 .
Job-ID	Is returned by <code>slsc_list_begin</code> , <code>slsc_list_begin_absolute</code> and <code>slsc_list_begin_relative</code> .
Job function (<code>slsc_list_*</code>)	Designates a function in the API with the prefix <code>slsc_list_</code> . These functions serve to define Jobs (for example, for markings, jumps as well as to switch signals at output ports etc.). See Chapter 3.1.2 "Job Functions (<code>slsc_list_*</code>)" , page 85 .

Job queue	Designates a sequence of Jobs (unlimited number). A syncAXIS control instance manages one Job queue at a time only. It is created during initialization and is initially empty. See also Figure 13, page 44 .
Job status	See Figure 12, page 43 .
Jump command	syncAXIS-DLL-function that serves to move the scan system axes to a new position while the laser is <i>off</i> . Example: <code>[*]jump[*]</code> .
"Laser Active" Operation	See Figure 42 as well as RTC6 Manual, Chapter 7.4 "Laser Control", page 183 , in particular Signals for "Laser Active" Operation, page 185 . In conclusion, the laser emits.
"Laser Standby" Operation	See Figure 42 as well as RTC6 Manual, Chapter 7.4 "Laser Control", page 183 , in particular Signals for "Laser Standby" Operation, page 185 . In conclusion, the laser does <i>not</i> emit.
List	This term is reserved for a direct context with RTC boards. Therefore, in <i>direct context with the syncAXIS-DLL</i> it is not used in this manual. However, compare to prefix <code>slsc_list_</code> of Job functions .
LSB	Least Significant Bit.
Mark function	syncAXIS-DLL-function that serves to trigger a marking motion while the laser is switched on. Examples: <code>[*]mark[*]</code> , <code>[*]arc[*]</code> , <code>[*]ellipse[*]</code> .
Module	A Job that has been recorded in simulation mode by <code>slsc_list_begin_module</code> , see Chapter 2.11 "About Working with "Modules"" , page 65.
Motion decomposition	<p>Applies to Operation mode "ScannerAndStage" only. Following the Trajectory planning (= after the laser spot path has been calculated; see Figure 11, page 41), splitting occurs according to the FilterBandwidth value to a portion to be carried out by the scan head and a portion to be carried out by the positioning stage.</p> 

MSB	Most Significant Bit.
Multi-Head	Designates an XL SCAN system where 1 syncAXIS control instance controls more than one excelliSCAN scan head and 1 positioning stage.
Multi-Instance	Designates the (optional) ability to run more than one syncAXIS control instance on a PC at the same time. Requires a Dongle that explicitly supports this option ^(a) . Multi-Instance and Multi-Head are basically compatible ^(b) .
Multi-Stage	Designates the (optional) functionality of the syncAXIS control instance to be able to change the positioning stage. Requires a Dongle that explicitly supports this option.
Operation mode	See enum slsc_OperationMode .
Output buffer	syncAXIS-DLL-internal temporary memory at the end of the processing chain. See Buffer and Figure 11 , page 41 in Chapter 2.7 "About Processes at Run Time of the User Program" , page 41.
Ramp	Designates a variation of the output values at ActiveChannel along curves (curve = marking pattern consisting of straight and/or curved parts - compare to Trajectory). See Chapter 2.7 "About Processes at Run Time of the User Program" , page 41.
Segment	Sections of the Trajectory which have been defined by the User specifying target coordinates by the following Job functions (slsc_list_*): <ul style="list-style-type: none"> • See To define jumps to target coordinates, page 86 • See To define markings to target coordinates, page 86 Accordingly, there is the Segment type: <ul style="list-style-type: none"> • Jump Segment • Mark Segment
Simulation Setting	State of the syncAXIS control instance with respect to simulation mode/hardware mode, see slsc_SimulationSetting . See also Chapter 2.5 "About the syncAXIS control Simulation Mode" , page 31.
Subcycle Switching	Feature of current RTC6 Software Packages, which is also used by the [*]dashed[*] Functions . This allows the RTC6 board to switch the laser up to 20× at each microstep (10 μs; see RTC6 Manual , Chapter 7.1.2 "Microstepping" , page 139).
syncAXIS control instance	A software object which is created in the PC-RAM when a valid syncAXISConfig.xml is called by a syncAXIS control-based user program. A syncAXIS control instance is configured for either the hardware mode or the simulation mode, see Chapter 2.4 "About Initializing syncAXIS control-based User Programs" , page 26. Every syncAXIS control instance can be addressed by a unique Handle .
syncAXISConfig.xml	XML configuration file. Although the file name can be freely chosen, it is denominated as " syncAXISConfig.xml " throughout this document. The complete tag descriptions can be found in Chapter 13 "Appendix F: Reference of syncAXISConfig.xml Tags" , page 358.

syncAXISSysConfig.xml	Omitted for syncAXIS control-software package \geq V1.2: XML system configuration file.
Trajectory	Curve with time parameterization.
Trajectory planning	See Figure 11, page 41 .
User	Designates a person (= "system programmer") who develops user programs using the syncAXIS control software package. Not meant is the "user or operator of a syncAXIS control system".
Utility Function (slsc_util_*)	Designates a function in the API with the prefix slsc_util_ . These functions are for special purposes outside the regular syncAXIS control operation. See Chapter 3.1.4 "Utility Functions (slsc_util_*)", page 101 .

- (a) The number of allowed [syncAXIS control instances](#) is coded on the [Dongle](#).
- (b) Not compatible with certain special systems where 2 [syncAXIS control instances](#) on a single PC control 2 master/slave connected RTC6 boards that feed into the same EtherCAT network.

2 Software Development with the syncAXIS-DLL

2.1 Safety

For developing syncAXIS control-based user programs, observe the security concept of your system control.

Make sure that neither the laser is not switched on unexpectedly nor the positioning stage is moved unexpectedly.



Warning!

Risk of injury due to laser radiation! Comply with laser safety regulations!



Warning!

Risk of injury due to laser radiation! `slsc_cfg_initialize_from_file` and `slsc_util_reset_pcie` can lead to undefined states of the RTC6 board(s) in which the laser could be switched on unexpectedly! Make sure that the laser is switched off before calling these functions!



Caution!

Make sure that laser safety is ensured in the entire system. In the safety concept of your system control, take into account that the RTC laser control signals are enabled by `slsc_cfg_initialize_from_file` and `slsc_ctrl_enable_laser`.



Caution!

Make sure that laser safety is ensured in the entire system. In the safety concept of your system control, take into account that the laser is on during `Job` execution.



Warning!

Risk of injury due to positioning stage movement! No persons in the danger zone!



Caution!

Risk of property damage due to positioning stage movement! No foreign objects in the danger zone!



Caution!

A moving positioning stage poses mechanical hazards. There are risks of injuries to fingers and hands from crushing. In the safety concept of your system control, take into account that `slsc_ctrl_start_execution` can move the positioning stage (possibly with a certain delay). Make sure that all bystanders keep sufficient distance to the appliance during execution.

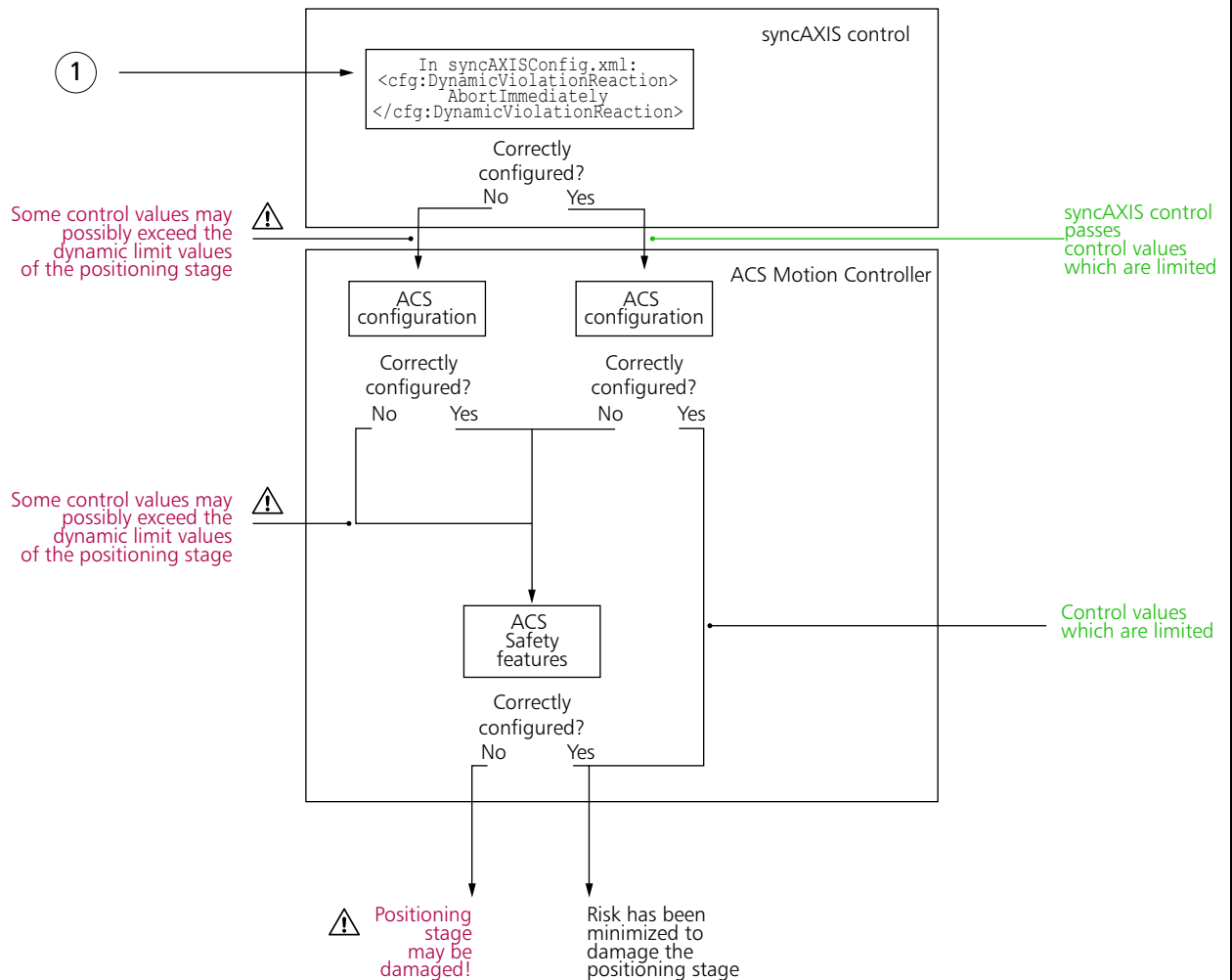


Warning!

Code sections in this manual must never be executed on actual XL SCAN systems without prior adaptation and simulation. Otherwise there is a risk of personal injury and damage to property. Disclaimer: SCANLAB accepts no liability for damages or consequential losses resulting from non-observance of this warning. SCANLAB does not take any responsibility on the correctness or functionality of these code sections.

Notes

- For a quick start the syncAXIS control-software package includes the "Installation_Project" (source code in C++). See also "Installation of SCANLAB XL SCAN Components and Initial Operation of the XL SCAN System" Manual.



Legend

- User entries in `syncAXISConfig.xml`. syncAXIS control uses to monitor working field and dynamics as:
 - scan device dynamic limits the `DynamicLimits` values, [page 388](#)
 - scan device working field limits the `FieldLimits` values, [page 395](#)
 - scan device monitoring criterion the `MonitoringLevel` value, [page 397](#)
 - positioning stage dynamic limits the `FieldLimits` values, [page 454](#)
 - positioning stage working field limits the `DynamicLimits` values, [page 456](#)
 - positioning stage monitoring criterion the `MonitoringLevel` values, [page 452](#)
 - reaction on exceedances the `DynamicViolationReaction` values, [page 365](#)

Ensure the prerequisites that syncAXIS control \geq V1.2 can pass limited control values to ACS Motion Controller!

2.2 About the SAFE Use of syncAXIS control – General Approach

The syncAXIS control software can be flexibly adapted to different laser scan system hardware components with configuration parameters. syncAXIS control-based user programs can also be optimized for throughput and accuracy with additional parameters.

However, this openness to customization and optimization means that users must configure syncAXIS control very carefully. Only then can the built-in safety mechanism become effective at which the control values transmitted to the **ACS Motion Controller** are limited in a way that further following mechanisms could prevent damage as well, see [Figure 2, page 17](#) and [Section “syncAXIS control Safety Features”, page 18](#).

Damage from such improper control may include:

- Injuries to personnel (for example, from laser radiation or by positioning stage movements)
- Damage to the laser scan system (for example, by vignetting at the scan head and objective, or from unrestrained excursions of the positioning stage against its stop)
- Increased production costs (for example, due to production process interruptions or when workpieces are rendered unusable due to missing or truncated markings)

To prevent such damage, absolutely conduct the following steps before running the first user programs on your laser scan system:

- [Identifying System Limits, page 19](#)
- [Establishing Safety Mechanisms, page 19](#)
- [Configuring Safe syncAXIS control Instances, page 20](#)
- [Simulating and Improving Jobs, page 24](#)

syncAXIS control Safety Features

As of syncAXIS control \geq V1.2, the `syncAXISConfig.xml` tag `DynamicViolationReaction` is available as an essential safety feature⁽¹⁾.

The `DynamicViolationReaction` value determines the reaction to limit value exceedances.

These limit value exceedances may include

- working field violations as well as
- dynamic violations of
- scan devices and
- positioning stages.

Which limit value exceedances are to be specifically monitored (“monitoring criteria”), is set in the `syncAXISConfig.xml` separately for scan devices and positioning stages under:

- `<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:MonitoringLevel>`
- `<cfg:Configuration> → <cfg:StageConfig> → <cfg:MonitoringLevel>`

The exact reaction to such limit value exceedances is determined by the `DynamicViolationReaction` .value. Possible reactions are to create **[WARN] log file lines** (`WarningOnly`), an immediate interruption of the movement (“emergency stop”; `AbortImmediately`) or an attempt to perform a controlled deceleration movement (`StopAndReport`).

(1) Remark on syncAXIS control \leq V1.1: these version only create **[WARN] log file lines**. The movement is stopped abruptly, if the **ACS Motion Controller** has been actually unable to follow the trajectory due to the limit value exceedance. Then both systems, syncAXIS control and **ACS Motion Controller**, are set to an error state.

2.2.1 Identifying System Limits

Identify the real-world limits of your XL SCAN system – for example, by referencing the corresponding user manuals or by asking the respective manufacturers:

- usable positioning stage working range
- max. positioning stage speed
- max. positioning stage acceleration
- max. positioning stage jerk
- max. usable scan head working field (that is, max. deflection angle and max. image field, depending on the used objective)
- max. galvanometer scanner speed⁽¹⁾
- max. galvanometer scanner acceleration⁽¹⁾
- max. galvanometer scanner jerk⁽¹⁾
- max. allowed laser power (and laser power density)

The positioning stage limits depend on the used motor model and the used axis servos.

2.2.2 Establishing Safety Mechanisms

Establish safety mechanisms to prevent personnel injuries and material damage:

- Employ appropriate safety mechanisms, safety warnings and protective gear to sufficiently protect against the radiation of the used working laser.
- Implement appropriate safety mechanisms to sufficiently prevent personnel injuries due to positioning stage motions.
- Possibly in consultation with manufacturers of the positioning stage and its axis servos, implement mechanisms to ensure compliance with the limits of the positioning stage (for example, by automatic braking or shutting down the positioning stage). This way, you can prevent positioning stage damage, as well as related consequential damage.
- If needed, implement safety mechanisms to ensure that laser power does not exceed the allowed range.

(1) This limit value is typically predefined by SCANLAB in `syncAXISConfig.xml` and generally does not need to be changed there.

2.2.3 Configuring Safe syncAXIS control Instances

Notes

- For syncAXIS control \geq V1.2, the XML system configuration file `syncAXISsysConfig.xml` is omitted.
- For all details on the `syncAXISConfig.xml`, see Chapter 13 "Appendix F: Reference of syncAXISConfig.xml Tags", page 358.

In the `syncAXISConfig.xml`, correct planning parameter values and monitoring limit values for the to-be-used scan devices and positioning stages must be entered.

- (1) Planning parameters for the scan devices, page 20
- (2) Monitoring limit values for the scan devices, page 20
 - a. Scan device working fields, page 20
 - b. Scan device dynamic limits, page 21
- (3) Planning parameters for the positioning stages, page 21
- (4) Monitoring limit values for the positioning stages, page 21
 - a. Positioning stage working fields, page 21
 - b. Positioning stage dynamic limits, page 22

(1) Planning parameters for the scan devices

⚠ Caution! syncAXIS control uses these values to plan trajectories for the **Operation modes** "ScannerOnly" and "ScannerAndStage".

Make sure that the entered acceleration and jerk values are correct. For velocity values, see page 390.

```
<cfg:Configuration>
  <cfg:ScanDeviceConfig>
    <cfg:CalculationDynamics>
      <cfg:MarkDynamics>
        <cfg:Acceleration>
        <cfg:Jerk>
      <cfg:JumpDynamics>
        <cfg:Acceleration>
        <cfg:Jerk>
```

(2) Monitoring limit values for the scan devices

a. Scan device working fields

```
<cfg:Configuration>
  <cfg:ScanDeviceConfig>
    <cfg:FieldLimits>
      <cfg:XDirection ...>(1)
      <cfg:YDirection ...>(1)
```

These values are used for monitoring only. If as scan device-MonitoringLevel the monitoring criterion Position, Velocity, Acceleration or Jerk is entered,

```
<cfg:Configuration>
  <cfg:ScanDeviceConfig>
    <cfg:MonitoringLevel>
      Position
    </cfg:MonitoringLevel>
```

then exceedances automatically trigger the reaction defined in DynamicViolationReaction (WarningOnly OR AbortImmediately OR StopAndReport).

(1) The tag has a Unit attribute value that is, the unit can be set.

b. Scan device dynamic limits

```
<cfg:Configuration>
  <cfg:ScanDeviceConfig>
    <cfg:DynamicLimits>
      <cfg:Velocity ...>
      <cfg:Acceleration ...>
      <cfg:Jerk ...>
```

These values are used for monitoring only. If as scan device-MonitoringLevel the monitoring criterion Position, Velocity, Acceleration or Jerk is entered,

```
<cfg:Configuration>
  <cfg:ScanDeviceConfig>
    <cfg:MonitoringLevel>
      Position
    </cfg:MonitoringLevel>
```

then exceedances automatically trigger the reaction defined in DynamicViolationReaction (WarningOnly OR AbortImmediately OR StopAndReport).

(3) Planning parameters for the positioning stages

```
<cfg:Configuration>
  <cfg:StageConfig>
    <cfg:StageList>
      <cfg:Stage>
        <cfg:CalculationDynamics>
          <cfg:Velocity ...>(1)
          <cfg:Acceleration ...>(1)
          <cfg:Jerk ...>(1)
```

⚠ Caution! syncAXIS control uses the values at Velocity, Acceleration and Jerk to plan trajectories for the Operation mode “StageOnly” as well as for the end motion at Job ends. Make sure that the entered values are correct.

(4) Monitoring limit values for the positioning stages

a. Positioning stage working fields

```
<cfg:Configuration>
  <cfg:StageConfig>
    <cfg:StageList>
      <cfg:Stage>
        <cfg:FieldLimits>
          <cfg:XDirection ...>(1)
          <cfg:YDirection ...>(1)
```

These values are used for monitoring only. If as positioning stage-MonitoringLevel the monitoring criterion Position is entered,

```
<cfg:Configuration>
  <cfg:StageConfig>
    <cfg:MonitoringLevel>
      Position
    </cfg:MonitoringLevel>
```

then exceedances automatically trigger the reaction defined in DynamicViolationReaction (WarningOnly OR AbortImmediately OR StopAndReport).

b. Positioning stage dynamic limits

```
<cfg:Configuration>
  <cfg:StageConfig>
    <cfg:StageList>
      <cfg:Stage>
        <cfg:DynamicLimits>
          <cfg:Velocity>
          <cfg:Acceleration>
          <cfg:Jerk>
```

These values are used for monitoring only. If as positioning stage-MonitoringLevel the monitoring criterion Velocity, Acceleration or Jerk is entered (Acceleration also subsumes Velocity, Jerk also subsumes Acceleration),

```
<cfg:Configuration>
  <cfg:ScanDeviceConfig>
    <cfg:MonitoringLevel>
      Velocity OR Acceleration OR Jerk
    </cfg:MonitoringLevel>
```

then exceedances automatically trigger the reaction defined in DynamicViolationReaction (WarningOnly OR AbortImmediately OR StopAndReport).

To attain good system performance, ensure that the limit values entered under (1) Planning parameters for the scan devices, page 20 and (3) Planning parameters for the positioning stages, page 21 match the real limits of your system.

In the interests of high system security, ensure that no limit values are entered under (2) Monitoring limit values for the scan devices, page 20 and (4) Monitoring limit values for the positioning stages, page 21 that lie outside the real system limits.

Additionally ensure that syncAXIS control-based user programs only use syncAXISConfig.xml in which correct limit values are entered (a syncAXIS control instance is initialized using the syncAXISConfig.xml which is specified at slsc_cfg_initialize_from_file) for subsequent simulations and real-world marking.

Notes

- If you subsequently make any hardware changes to your XL SCAN system (for example, a new objective with a different focal length, or a different positioning stage), then you must also recheck the settings in all pertaining XML configuration files. If necessary, adjust the entered limit values according to the then valid system limits.
- syncAXIS control utilizes values entered in `syncAXISConfig.xml` for
 - (1) Planning parameters for the scan devices, page 20 and
 - (3) Planning parameters for the positioning stages, page 21
 for tasks such as calculating trajectory control values for the scan head and positioning stage⁽¹⁾, but only in the sense to calculate the most efficient trajectories possible.

Important: syncAXIS control does not perform automatic clipping of control values and process speeds to the limits of the system.

Some control values and process speeds may occur that exceed the system limits, particularly during fast jump motions executed by the scan head and positioning stage between the actual markings.

Before real-world marking with a laser, scan head and positioning stage, you absolutely must therefore perform a simulation to check that control values and process speeds calculated by syncAXIS control are within the system limits, and you must modify the **Job** as necessary until compliance is attained, see Chapter 2.2.4 “Simulating and Improving Jobs”, page 24.

- syncAXIS control uses to monitor working field and dynamics as:
 - scan device dynamic limits the `DynamicLimits` values, page 388
 - scan device working field limits the `FieldLimits` values, page 395
 - scan device monitoring criterion the `MonitoringLevel` value, page 397
 - positioning stage dynamic limits the `FieldLimits` values, page 454
 - positioning stage working field limits the `DynamicLimits` values, page 456
 - positioning stage monitoring criterion the `MonitoringLevel` values, page 452
 - reaction on exceedances the `DynamicViolationReaction` values, page 365

Notice!

With multi-head systems, the following applies:

- syncAXIS control does not check for working field overlaps of the scan devices. Make sure that each scan device can only process its “own” workpiece.
- syncAXIS control cannot check whether scan head working field boundaries are violated due to `slsc_cfg_set_part_displacement` transformations. A controlled deceleration as a `DynamicViolationReaction` is not possible.



Caution!

Despite the entered limit values in `syncAXISConfig.xml`, syncAXIS control can, under some circumstances, calculate control values and process speeds that exceed the system limits.

Therefore, always initially run syncAXIS control-based user programs in simulation mode (to check for system limit violations) before first-time execution of real-world marking with the laser, scan head and positioning stage.

(1) ...and a positioning stage compensation motion at the end of a **Job**. Additionally, syncAXIS control writes warnings to the log file when system limits are violated (if enabled in `syncAXISConfig.xml`, see “About the Logging in syncAXIS control”, page 47).

2.2.4 Simulating and Improving Jobs

Before you run a syncAXIS control-based user program for the first time to mark in the real world with a laser, scan head and positioning stage, proceed as follows:

- (1) Run the user program in simulation mode to record⁽¹⁾ the control values that syncAXIS control calculates, see [Chapter 2.5 "About the syncAXIS control Simulation Mode"](#), page 31. See also the Note to the right, page 24.
- (2) Analyze the recorded control values to detect violations of system limits – for example, by displaying them graphically⁽²⁾. Here, take into account concerns such as the following:
 - Are control values issued for the positioning stage that it cannot traverse or that would cause it to hit the stop or that could trigger its emergency shut-down?
 - Are control values issued for the scan head that could cause vignetting at the scan head or scan objective?
 - Are speeds issued that would force the positioning stage to exceed its actually achievable speeds, accelerations or jerks?
 - You could also optionally carry out an optimization analysis here:
 - How large is the working field actually used by the scan head?
 - How long is the overall process time of your syncAXIS control-based user program?
- (3) Repeatedly improve the syncAXIS control-based user program until you no longer find limit violations in subsequent simulation and analysis steps.

Notes

- With syncAXIS control \geq V1.2.4, the following applies: the extrema of the current **Job** can be queried by `slsc_ctrl_get_job_characteristic`. The file output⁽¹⁾ does *not* need to be switched on.
- With syncAXIS control \geq V1.6.0, see also [Section "Functions for Changing Trajectory planning Values"](#), page 94.
- With syncAXIS control \geq V1.7.0, the following applies: In **Job** pre-analyses for optimization the duration of jumps can be calculated by `slsc_cfg_get_jump_time`.

(1) See [DisableFileOutput](#).

(2) for example, in syncAXIS Viewer.

2.3 About the Main Structures of a syncAXIS-DLL-Based User Program (Exemplary)

The following describes (exemplary) the fundamental parts of a user program which uses the syncAXIS-DLL.

- (1) Program part to initialize and to configure the syncAXIS control instance
 - Calling `slsc_cfg_initialize_from_file`, see Chapter 2.4 “About Initializing syncAXIS control-based User Programs”, page 26.
 - A check whether the syncAXIS control instance status is “green”: calling `slsc_cfg_get_operation_status`.
 - Note: At this point (as described in the Installations Manual), the **Operation mode** could already be changed (`slsc_cfg_set_mode`). Example: For system calibration (among other things) some **Jobs** must be executed (marking the reference grids) in different **Operation modes** “ScannerOnly” and “StageOnly”.
- (2) Program part (code) that handles the **Job** definitions and **Job** sequences
 - Assembling **Jobs** from the suitable **Mark functions** and **Jump commands** (each individual **Job** having `slsc_list_begin` at the beginning and `slsc_list_end` at the end, see Chapter 2.3.1 “Structure to Comply with when Defining Jobs”, page 25)
 - As required: insert functions (`slsc_list_write_analog_x`, `slsc_list_write_digital_out`, `slsc_list_write_digital_out_mask`) which set signals at output ports (corresponding to the RTC commands (`list_write_da*`, `list_write_io*`) on run-time of the user program.
 - Supplying all function parameters with the desired values (for example, coordinates).
 - Note: the sequence of **Jobs** in the source code corresponds to the sequence of execution (principle “first in – first out”).

- (3) Program part that starts and monitors the execution of **Jobs**
 - Checking the prerequisite (`slsc_ctrl_get_exec_state`): is the RTC6 board ready to execute (`slsc_ExecState_ReadyForExecution`)
 - Triggering the execution start (`slsc_ctrl_start_execution`)
 - Monitoring the execution status, for example evaluate events via callbacks (for example, has the execution terminated)
- (4) Program part that detects errors and reacts as defined by the user
 - Querying whether errors have been occurred (`slsc_ctrl_get_error_count`).
 - Reacting to errors (subject to user knowledge)
- (5) Program part that destructs the syncAXIS control instance
 - Calling `slsc_cfg_delete`

2.3.1 Structure to Comply with when Defining Jobs

With **Jobs**, the mandatory succession of the Job functions (`slsc_list_*`) is as follows:

- (1) `slsc_list_begin*`
- (2) 0...unlimited number of Job functions `slsc_list_*` – except `slsc_list_begin*`⁽¹⁾
- (3) `slsc_list_end`

On run-time, **Jobs** are transferred to the syncAXIS control instance by the user program function by function, see Figure 11, page 41.

The syncAXIS-DLL verifies the incoming **Jobs** whether the succession of the Job functions (`slsc_list_*`) is consistent with the mandatory function sequence (consistency check). In case of an error, the return value indicates that **Bit #07** is set (`JobStructureNotValid`).

(1) That is, **Jobs** cannot be nested.

2.4 About Initializing syncAXIS control-based User Programs

Notice!

In order to operate several **syncAXIS control instances** in hardware mode on a PC simultaneously a correspondingly configured **Dongle** must be used!

Otherwise, the return value indicates that Bit #30 is set (**MaxInstancesReached**).

A **syncAXIS control instance** is configured for either

- hardware mode
(USE CASE 1 in [Figure 3, page 27](#)) or
- simulation mode
(USE CASE 2 in [Figure 3, page 27](#)).

Notes on the Hardware Mode

- Every **syncAXIS control instance** requires 1 RTC6 board (installed in the PC, and not acquired by another user program).



Warning!

Risk of injury due to laser radiation! Comply with laser safety regulations!



Warning!

Risk of injury due to positioning stage movement! No persons in the danger zone!



Caution!

Risk of property damage due to positioning stage movement! No foreign objects in the danger zone!

In order to create the **syncAXIS control instance**, **slsc_cfg_initialize_from_file** is called from within the user program ("Executable").

In the specified **syncAXISConfig.xml**, the following must be entered:

- For hardware mode:

```
<cfg:SimulationMode>>false
</cfg:SimulationMode>
```
- For simulation mode:

```
<cfg:SimulationMode>>true
</cfg:SimulationMode>
```

The **syncAXIS control instance** is created and gets a **Handle** assigned by **slsc_cfg_initialize_from_file** (also: **slsc_cfg_initialize_copy**). Via this **Handle**, it can be addressed by the (most of) syncAXIS-DLL-functions (**Handle** value as parameter). During the creation the following processes take place⁽¹⁾:

- Scan head: is acquired. The scan head mirrors are taken to the zero position.
- RTC6: is acquired. The RTC6 laser control is armed. The laser control signals are already released at the RTC6.
- Positioning stage: is acquired. The positioning stage position remains unchanged.
- Output signals: values are applied as specified in the **syncAXISConfig.xml** under

```
<cfg:IOConfig><cfg:DefaultOutputs>
(= LaserPinOut, AnalogOut1, AnalogOut2).
```

Which of the available RTC6 boards (if there are more than one available in a PC at all) is used, is determined by the corresponding **syncAXISConfig.xml**-entries under **<cfg:RTCConfig>** for determining their identification by serial number.

```
<cfg:BoardIdentificationMethod>BySerialNumber
</cfg:BoardIdentificationMethod>
```

and also

```
<cfg:SerialNumber>123456
</cfg:SerialNumber>.
```

If only a single RTC6 board is installed, the following entry is sufficient:

```
<cfg:BoardIdentificationMethod>UseFirstFound
</cfg:BoardIdentificationMethod>.
```

(1) See [page 118](#) on processes during destruction.





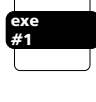
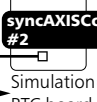
For both hardware mode and simulation mode, the following applies: the **syncAXIS control instance** is started in that **Operation mode** which is specified in **syncAXISConfig.xml**:

```
<cfg:InitialOperationMode>
ScannerOnly|StageOnly|ScannerAndStage
</cfg:InitialOperationMode> (1)
```

(1) After initialization, the **Operation mode** can be changed by **slsc_cfg_set_mode**.

For reinitializing a **syncAXIS control instance**, **slsc_cfg_reinitialize_from_file** is available, see USE CASE 1a, in the middle of **Figure 3, page 27**. This function deletes the specified **syncAXIS control instance** and creates it again. However, it does not change the **Handle** value of the **syncAXIS control instance**.

For more information on simulation mode, see **Chapter 2.5 "About the syncAXIS control Simulation Mode", page 31**. In the RAM of a PC, only 1 single **syncAXIS control instance** in simulation mode may exist, see also **Figure 6, page 30, bottom**.

syncAXIS control-executable	Call	Argument(s)	Resulting software object	Effect on RTC6 status (here: 3 RTC6 installed)
USE CASE 1 	slsc_cfg_initialize_from_file	 Simulation=FALSE BoardIdentificationMethod=BySerialNumber	syncAXIS control Instance - in "Hardware mode" - Handle value = Handle#1	RTC6#1 = acquired by Handle#1 RTC6#2 = not acquired RTC6#3 = not acquired
USE CASE 1a 	slsc_cfg_reinitialize_from_file	 Simulation=FALSE BoardIdentificationMethod=BySerialNumber	-Handle remains -deletes -re-creates syncAXIS control Instance - in "Hardware mode" - Handle value = Handle#1	RTC6#1 = acquired by Handle#1 RTC6#2 = not acquired RTC6#3 = not acquired
USE CASE 2 	slsc_cfg_initialize_from_file	 Simulation=TRUE RTC board=<entry ignored>	syncAXIS control Instance - in "Simulation mode" - Handle value = Handle#2	RTC6#1 = not acquired RTC6#2 = not acquired RTC6#3 = not acquired

3

Overview on initializations: USE CASE 1, 1a 2. Not illustrated: along with the RTC6 board, the positioning stage is also acquired (applies also to **Operation mode "ScannerOnly"**).

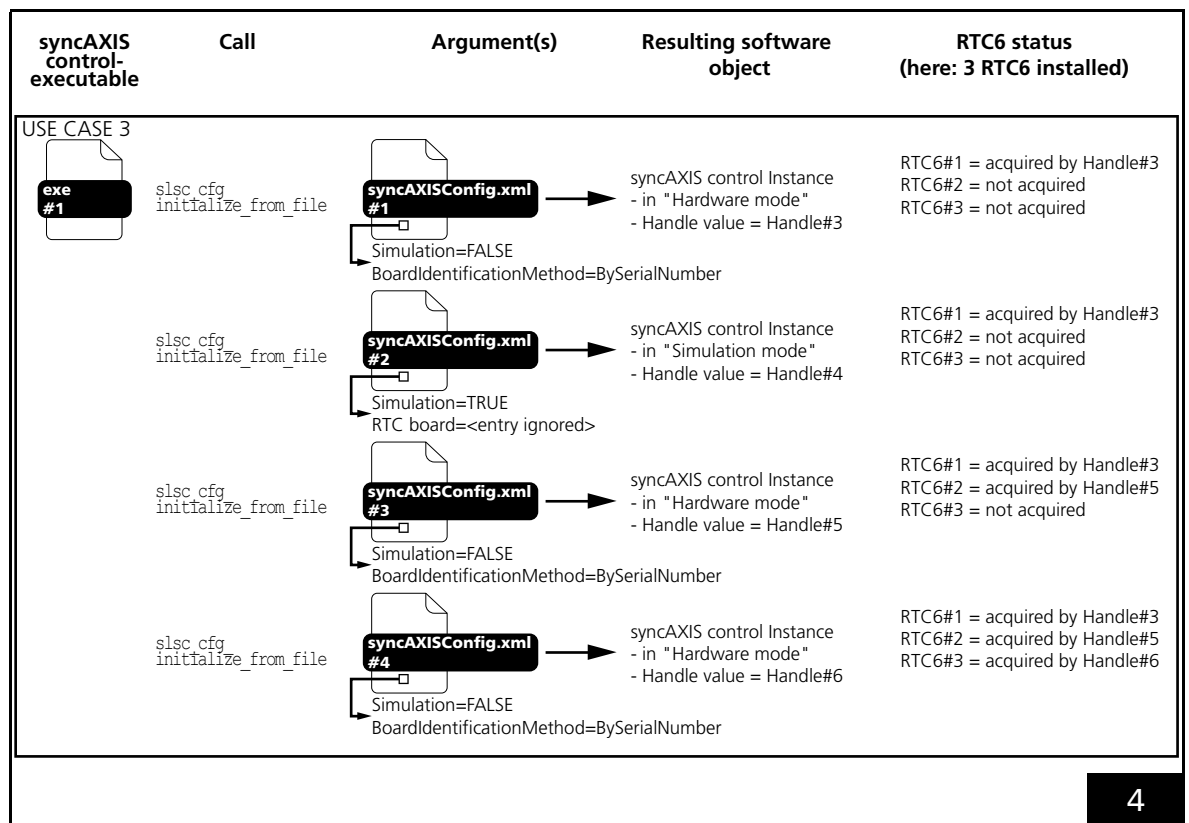
USE CASE 3 in [Figure 4, page 28](#) illustrates how a single user program sequentially calls 4 different (!) `syncAXISConfig.xml`.

Notice!

In order to operate several **syncAXIS control instances** in hardware mode on a PC simultaneously a correspondingly configured **Dongle** must be used!

Otherwise, the return value indicates that Bit #30 is set (`MaxInstancesReached`).

This results in 3 separate **syncAXIS control instances** in hardware mode and 1 **syncAXIS control instance** in simulation mode. All 3 available RTC6 boards are acquired in the end.



Overview on initializations: USE CASE 3. Not illustrated: along with the RTC6 board, the positioning stage is also acquired (applies also to **Operation mode "ScannerOnly"**).

USE CASE 4 in [Figure 5, page 29](#) illustrates how two different user programs call 2 different (!)

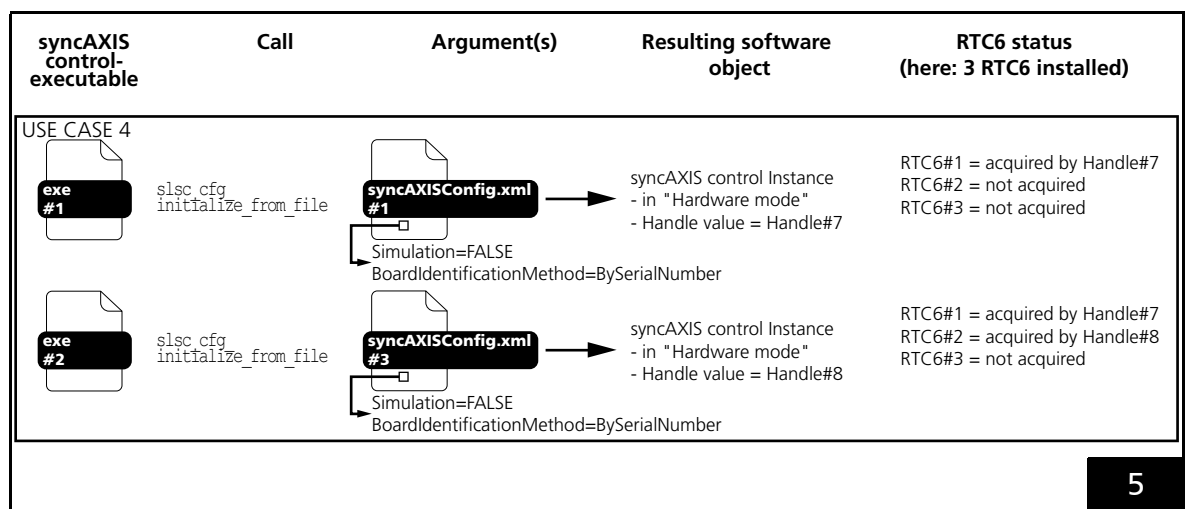
[syncAXISConfig.xml](#).

Notice!

In order to operate several [syncAXIS control instances](#) on a PC simultaneously a correspondingly configured [Dongle](#) must be used!

Otherwise, the return value indicates that Bit #30 is set ([MaxInstancesReached](#)).

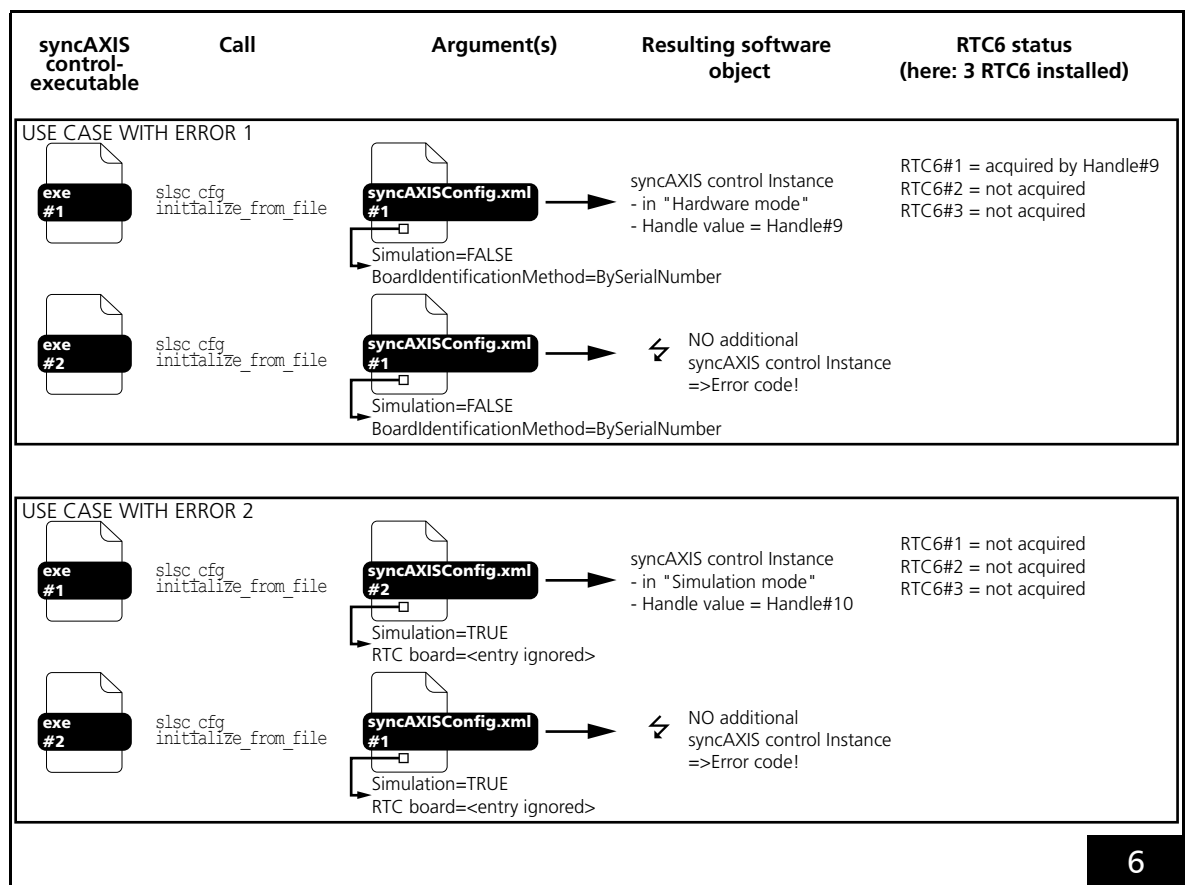
This results in 2 separate [syncAXIS control instances](#) in hardware mode. Subsequent to this, 2 (of 3) available RTC6 boards are acquired.



Overview on initializations: USE CASE 4. Not illustrated: along with the RTC6 board, the positioning stage is also acquired (applies also to [Operation mode "ScannerOnly"](#)).

In **Figure 6, page 30** on the top, **USE CASE WITH ERROR 1** illustrates the result, if two different user programs call the same (!) `syncAXISConfig.xml`. This results in only 1 **syncAXIS control instance** in hardware mode. A second one is not created. Subsequent to this, only 1 (of 3) available RTC6 boards is acquired.

In **Figure 6, page 30** on the bottom, **USE CASE WITH ERROR 2**, illustrates how two different user programs call 2 different `syncAXISConfig.xml`, where both having the same entry `<cfg:SimulationMode>true</cfg:SimulationMode>`. This results in only 1 **syncAXIS control instance** in simulation mode. A second one is not created. Subsequent to this, none (of 3) available RTC6 boards is acquired.



Overview on initializations: **USE CASE WITH ERRORS 1** and **USE CASE WITH ERRORS 2**. Not illustrated: along with the RTC6 board, the positioning stage is also acquired (applies also to **Operation mode "ScannerOnly"**).

2.5 About the syncAXIS control Simulation Mode

syncAXIS control-based user programs can be run with the **syncAXIS control instance** in simulation mode (\geq V1.5.0: to query **Simulation Setting** by **slsc_cfg_get_simulation_setting** and to change by **slsc_cfg_set_simulation_setting**).

The simulation mode:

- Requires the **Dongle**
- *Does not* require nor use any hardware (such as an RTC6 etc.)

Execution in simulation mode includes recording the control values calculated by syncAXIS control in a text file ("simulation file").

By analyzing those recorded values, users (for example, **Job** designers) then can determine – even before real-world testing with a laser and workpieces – which parts of a **Job** can or must be optimized:

- A **Job** *must* receive optimization, if calculated control values exceed system limits, **Chapter 2.2.1 "Identifying System Limits", page 19_**
- A **Job** *can* benefit from optimization in terms of process time (throughput) and execution precision (marking result quality), see **Chapter 2.6 "About Optimizing syncAXIS control-based User Programs", page 36**.

To run syncAXIS control-based user programs in simulation mode:

- The **Dongle** must be plugged in
- In the **syncAXISConfig.xml** (specified as an argument in **slsc_cfg_initialize_from_file**), simulation mode must already be set within the tag `<cfg:SimulationMode>`:

```
<cfg:SimulationMode>true
</cfg:SimulationMode>
```

(for initialization, see also **Chapter 2.4 "About Initializing syncAXIS control-based User Programs", page 26**),
- In the same XML configuration file, the target folder path for the simulation file must be specified as `<cfg:SimOutputFileDirectory>` value

With those settings, the syncAXIS control-based user program initializes the **syncAXIS control instance** in simulation mode.

Once it reaches **slsc_ctrl_start_execution** during execution, 1 (one) (\geq V1.5.0) simulation file is generated in the specified target folder path.

With syncAXIS control \geq V1.3, the following file naming convention is applied:

- `Simulation_ID_<Job-ID>_TS_<13 numerical digits>.txt(1)`

(1) Example: `Simulation_ID_2_TS_1546938743472.txt`.
 TS = timestamp. Note: V1.2 uses
`*_<scan device>_TS_*` instead of `*_TS_*`.

A simulation file (\geq V1.5) contains:

- (1) The configuration *actually* used for the **Job** in xml format⁽¹⁾⁽²⁾
 - These parameters have been read out from `syncAXISConfig.xml` during initialization
 - The parameter values may have remained unchanged since or may have been changed (by Configuration functions (`slsc_cfg_*`) called prior to **Job** execution, see Chapter 3.1.1 “Configuration Functions (`slsc_cfg_*`)”, page 76) in the meantime
- (2) A single line (starting with “`<!--Simulation` output: ”) with the column header names, see Table “Simulation files \geq V1.5: possible values per data record (see 3, page 32)”, Seite 33
- (3) Subsequent lines = the individual data records
 - Each data record
 - corresponds to an RTC6 micro vector command (10 μ s)
 - contains several semicolon-separated values
 - The number of values in the data sets of simulation files \geq V1.5 is no longer constant, but can vary, see Table “Simulation files \geq V1.5: possible values per data record (see 3, page 32)”, Seite 33. Thus, the lines can be of different lengths with \geq V1.5
- (4) Last line: “`-->`”

To check calculated control values for violations of system limits, you can use a suitable software tool to automatically read, evaluate or graphically display data from the simulation files.

An example of such a graphic representation shows Figure 7, page 34:

- It displays position values (X,Y) of the scan head and positioning stage as well as “Toggle signal” at end of micro vector. that resulted from executing an example **Job** in simulation mode and recording the values to a simulation file.
- Additionally, calculated speed, acceleration and jerk of the positioning stage are plotted which have been obtained through simple or multiple numerical derivation from the position values for the positioning stage.
- The figure shows that the Y position values of the scan head come into a borderline range at a single point of the example **Job**.
- The following applies to the ScannerAndStage Operation mode: Initially the curve values are still 0⁽³⁾, because the algorithms for Motion decomposition only takes effect after a few tenths of a second (“starting response time of the filter”). Furthermore, after the final motion command, the syncAXIS control software initiates motions to return the system to a stable state.

(1) Evaluation tools are able to read out the system limits directly.

(2) Can be exported as xml file with syncAXIS Viewer \geq V1.5.

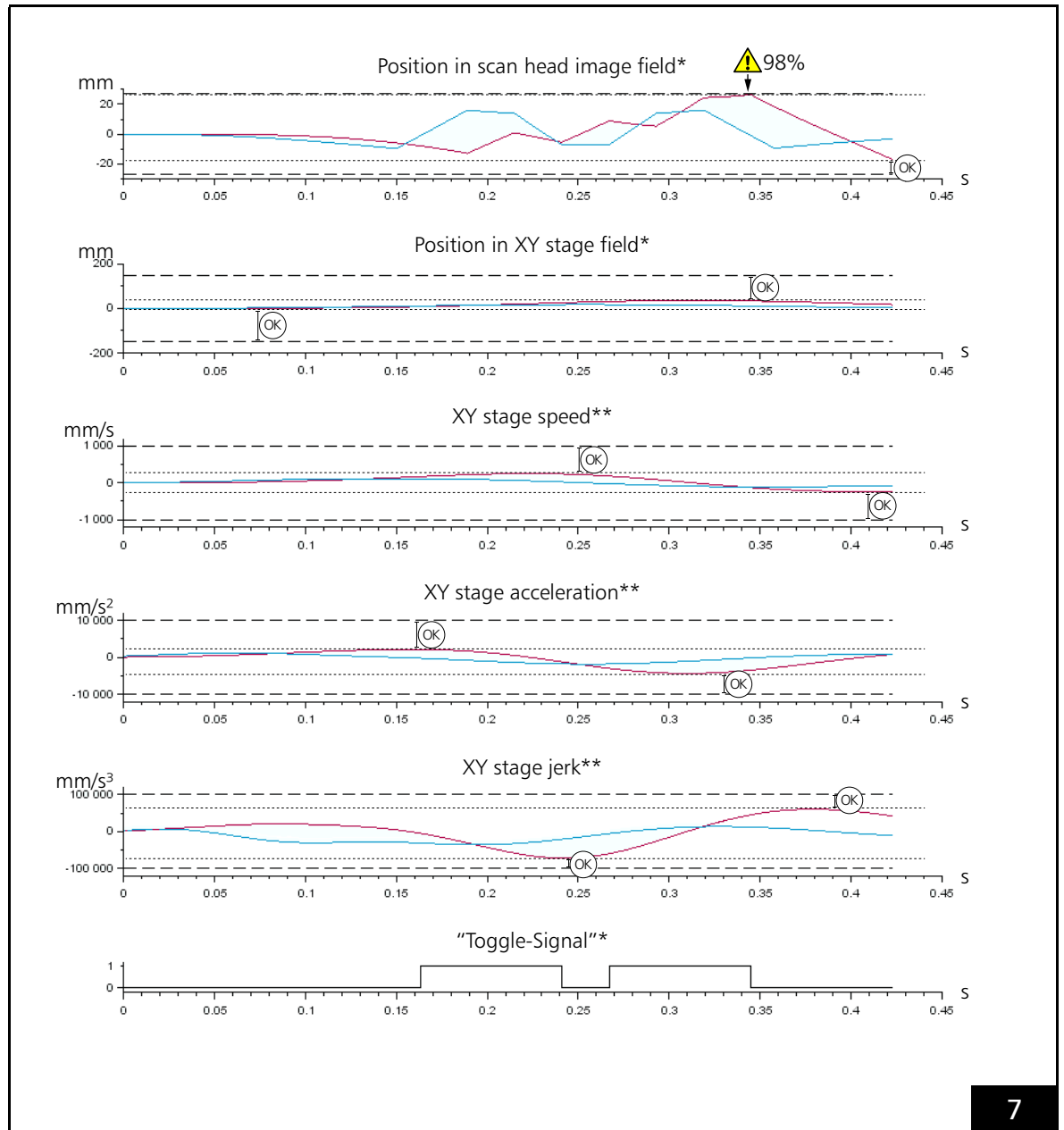
(3) The absolute position does not have to be 0, the relative position does.

Simulation files ≥ V1.5: possible values per data record (see 3, page 32)		
Occurrences per data record (see 3, page 32)	Name (see 2, page 32)	Meaning
1	ScanDevice1X	x value for scan device 1. In mm.
1	ScanDevice1Y	y value for scan device 1. In mm.
0...1	ScanDevice2X	x value for scan device 2. In mm.
0...1	ScanDevice2Y	y value for scan device 2. In mm.
0...1	ScanDevice3X	x value for scan device 3. In mm.
0...1	ScanDevice3Y	y value for scan device 3. In mm.
0...1	ScanDevice4X	x value for scan device 4. In mm.
0...1	ScanDevice4Y	y value for scan device 4. In mm.
0...1	StageX	x value for positioning stage. In mm.
0...1	StageY	y value for positioning stage. In mm.
1	NumLaserOn	Number of subsequent LaserOnDelays values [0...10].
0...10 (see NumLaserOn)	LaserOnDelays	"Laser Active" Operation starting point in time. ^(a) Relative to the micro vector start. In μ s.
1	NumLaserOff	Number of subsequent LaserOffDelays values [0...10].
0...10 (see NumLaserOff)	LaserOffDelays	"Laser Standby" Operation starting point in time. ^(a) Relative to the micro vector start. In μ s.
1	LaserToggle	"Toggle signal" at end of micro vector. 1: "Laser Active" Operation. 0: "Laser Standby" Operation.
1	ActiveChannel0	1. ActiveChannel value ^(b) : Unit is that of the set channel ^(c) . If no ActiveChannel is defined: 0.
1	ActiveChannel1	2. ActiveChannel value ^(b) . See 1. ActiveChannel.
1	CommandCount	Consecutive number of the Job function (slsc_list_*) in the Job.

(a) If there are no "Laser Active" Operations or "Laser Standby" Operations, this can be recognized in ≥ V1.5 by NumLaserOn/NumLaserOff.

(b) See Chapter 2.9 "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48.

(c) If SpotDistance is set as the ActiveChannel: Laser spot speed. In mm/s.



Graphic depiction of simulation data for an example **Job**:

* Data from the simulation file (position values and "Toggle signal" at end of micro vector.)

** Data derived from simulation file values (speed, acceleration, jerk)

blue line: X-values

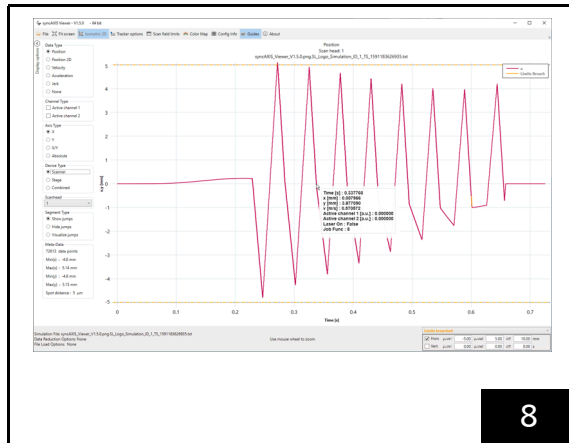
red line: Y-values

— — — — : System limits

- - - - - : Guide (plotted at the height of the maximum value of the curve to provide visual orientation)

Notes

- Data from simulation files can be visualized using the software tool SCANLAB syncAXIS Viewer, which is included in the syncAXIS control-software package, see [Figure 8, page 35](#).



8

SCANLAB syncAXIS Viewer.

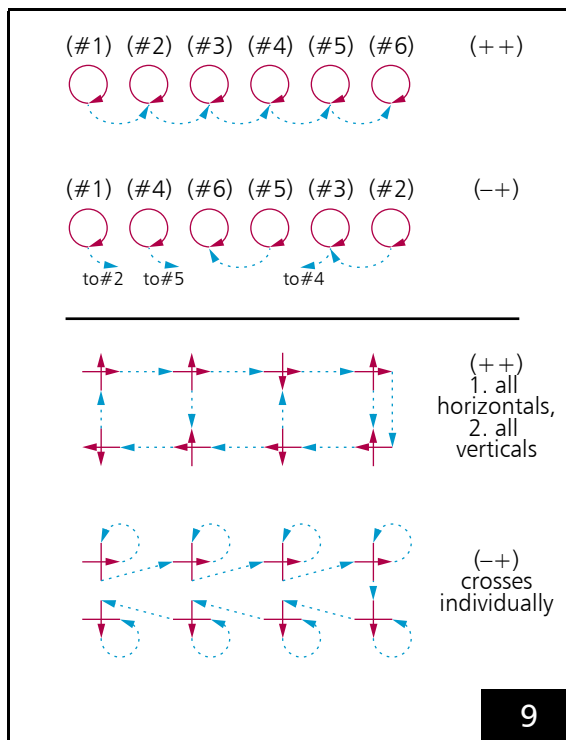
- Simulations cannot predict real-world marking quality. Therefore, after evaluating the simulation, you must always actually mark the marking pattern ("real-world test"⁽¹⁾) in order to evaluate the resulting quality. As of syncAXIS control \geq V1.0, hardware mode allows you to query certain characteristics (by `slsc_ctrl_get_job_characteristic`) after the **Job** has been calculated (for example, max. scan head position in the **Job**). Under certain circumstances, the syncAXIS control-based user program may have to be reworked and then executed yet again in simulation mode (the latter in order to recheck compliance with system limits).

(1) This requires to switch off simulation mode in the `syncAXISConfig.xml`: `<cfg:SimulationMode>>false`
`</cfg:SimulationMode>`.

2.6 About Optimizing syncAXIS control-based User Programs

2.6.1 Possible Optimizations

As a rule, a syncAXIS control-based user program must not only (as described in [Chapter 2.2.4 "Simulating and Improving Jobs"](#), page 24 and [Chapter 2.5 "About the syncAXIS control Simulation Mode"](#), page 31) be checked and improved for compliance with the system limits, but also optimized with regard to process time (throughput) and execution accuracy (marking result quality). Such an optimization can take place in the syncAXIS control-based user program itself, but also by changing initialization values (in the used `syncAXISConfig.xml`) for the syncAXIS control instance. One approach to increasing throughput is to cut marking pattern execution times by even cleverly arranging the sequence of **Mark functions** and **Jump commands** in the source code (for example, due to shorter jumps as illustrated in figure 9).



Different arrangements of functions in the source code can lead to identical marking results, but with differing execution times.

Another way to increase throughput is by raising process speeds.

- You can do this for all **Jobs** by defining a "global" default setting for jump speed and marking speed in the `syncAXISConfig.xml` – in the tags `<cfg:JumpSpeed>` and `<cfg:MarkSpeed>` (these apply as long as no "local" values were specified for a **Job**, see below). Both speeds affect motions of the scan head and positioning stage.
 - Mark speed is often dictated by the laser process.
 - Jump speed can typically be set higher than the marking speed. Although a higher jump speed reduces process times (higher throughput), it can sometimes introduce imprecision. High jump speeds can also sometimes lead to exceedance of system limits.
- To fine-tune process time and execution precision, you can additionally specify "local" jump speed and marking speed within a **Job** (that is, within the syncAXIS control-based user program) – by `slsc_list_set_jump_speed` and `slsc_list_set_mark_speed`:
 - in principle even only for one individual mark vector or jump vector
 - changes apply as of the insert position but only to the end of the **Job** at most.

In some cases, you can also boost throughput by dividing the **Job** (in the source code) into several sub-**Jobs**

- On the one hand into sub-**Jobs** that contain only those marking pattern portions that should be generated with the (typically faster) scan head (subsequent execution then in the **Operation mode** `ScannerOnly`)
- On the other hand into sub-**Jobs** that contain only those marking pattern portions that should be generated with **Motion decomposition** by the scan head *and* the positioning stage (subsequent execution then in the **Operation mode** `ScannerAndStage`).

The `FilterBandwidth` value can be specified for those sub-Jobs that are to be marked with simultaneous movement of the scan head and positioning stage (that is, in **Operation mode** "`ScannerAndStage`")⁽¹⁾.

In this **Operation mode**, this value affects syncAXIS control calculations of the **Motion decomposition** between the scan head and positioning stage:

- **Higher** `FilterBandwidth` values lead to greater motion participation by the positioning stage (and thus reduced usage of the scan head working field).
 - Possible advantages of this setting:
 - In some situations, the positioning stage is more precise than the scan head.
 - The positioning stage has a larger working field than the scan head and can thus also traverse along longer vector lengths.
 - Possible disadvantages of this setting:
 - The process speed may need to be reduced (the positioning stage is slower than the scan head and the positioning stage may not be able to follow the trajectory).
 - The positioning stage needs to move more mass. Thus with high jerks, the positioning stage might emit vibrations.
 - The positioning stage consumes more power than the scan head.
 - System limits might be violated, particularly the positioning stage dynamics.
- **Lowering** `FilterBandwidth` values lead to less motion participation by the positioning stage (and thus more motion participation by the scan head and greater usage of the scan head working field).
 - Possible advantages of this setting:
 - The scan head is faster than the positioning stage. Thus higher speeds can be set.
 - The scan head consumes less power than the positioning stage.
 - Possible disadvantages of this setting:
 - In some situations, the scan head is less precise than the positioning stage. For example, the scan head achieves less precise results near its working field boundaries. Accordingly, the overall accuracy is worse.
 - Because the scan head working field is smaller than that of the positioning stage, it cannot traverse long vector lengths by itself.
 - System limits might be violated, particularly the control values for the scan head.
 - For example, if users notice that the positioning stage motions are no longer keeping up (possibly indicated by a stage control warning), then they can reduce the used positioning stage dynamics (that is, reducing positioning stage motions) by lowering the `FilterBandwidth` value.
- If you merely change only the `FilterBandwidth` value, throughput will only be affected slightly (**Job** time is determined primarily by mark and jump lengths, as well as the applied speeds).
- The following additional relationship approximations apply to `FilterBandwidth` values:
 - $1/\langle \text{cfg:FilterBandwidth} \rangle$ value \sim utilized scan head working field
 - $\langle \text{cfg:FilterBandwidth} \rangle$ value \sim highest positioning stage acceleration
 - $(\langle \text{cfg:FilterBandwidth} \rangle \text{ value})^2 \sim$ highest positioning stage jerk

(1) In the `syncAXISConfig.xml`, tag `<cfg:FilterBandwidth>`. This value is the "global" default setting for all **Jobs**.

2.6.2 Iterative Approach

Unfortunately, we cannot increase throughput and precision ad finitum. Generally, throughput maximization and precision maximization are actually competing goals.

A realistic and typically iteration-based optimization strategy is to discover optimal process parameters for the specific process and marking pattern, in order to maximize throughput accompanied by still-acceptable precision (or the opposite: to maximize precision accompanied by still-acceptable throughput).

Support for this is provided by allowing syncAXIS control-based user programs to run in simulation mode, see [Chapter 2.5 "About the syncAXIS control Simulation Mode", page 31](#). Even without (more elaborate) real-world tests, you can thus discover which effects parameter changes have on process times, [Motion decomposition](#) between the scan head and positioning stage, and compliance with system limits etc.

SCANLAB recommends the following approach:

- Create and compile a syncAXIS control-based user program (source code) containing the desired marking pattern.
- Ensure that the real system boundaries are entered as limit values in the used `syncAXISConfig.xml`.
- Before the first iteration step, additionally verify the following settings in `syncAXISConfig.xml`:
 - Process speeds should be specified conservatively enough (low) to make system limit violations highly unlikely.
 - Operation mode:


```
<cfg:InitialOperationMode>
ScannerAndStage
</cfg:InitialOperationMode>
```
 - Simulation mode enabled


```
<cfg:SimulationMode>true
</cfg:SimulationMode>
```

- Specify output path for simulation files


```
<cfg:SimOutputFileDirectory>...
</cfg:SimOutputFileDirectory>
```
- Perform the following iteration steps:
 - Change program parameters:
 - for example, increase the jump speed in `syncAXISConfig.xml` to increase throughput
 - for example, lower the `FilterBandwidth` value to decrease the positioning stage motion participation and thus conversely increase the scan head motion participation and utilized working field
 - for example, modify the source code to make other optimizations (for example, sequence of functions), and then recompile it, if necessary
 - Run the syncAXIS control-based user program in simulation mode: see [Chapter 2.5 "About the syncAXIS control Simulation Mode", page 31](#)
 - Read data (position values and ["Toggle signal" at end of micro vector](#). etc.) from the generated simulation file to derive speed, acceleration and jerk.⁽¹⁾⁽²⁾
 - Analyze simulation data:
 - for example, check each parameter for system limit violations (here you should also check the generated log file, see footnote on [page 23](#). If a limit value exceedance is detected, then you absolutely must appropriately readjust the affected parameter in an additional iteration step and again perform a simulation and analysis of results.
 - Process time can be read from the generated simulation file. Here, note that calculation time (= time until a **Job** is ready for start) is significantly longer in simulation mode than in hardware mode.

(1) syncAXIS Viewer calculates these derived values.

(2) With syncAXIS control \geq V1.2.4, the following applies: the extrema of the current **Job** can be queried by `slsc_ctrl_get_job_characteristic`.

- If the simulation produced no limit value exceedances, you can also optionally run the syncAXIS control-based user program in a real-world test to mark an actual workpiece by the laser, scan head and positioning stage. You should check the generated marking for achieved quality. If needed, also perform additional iteration steps based on this quality check (here too, start with simulation and analysis of system limit violations).

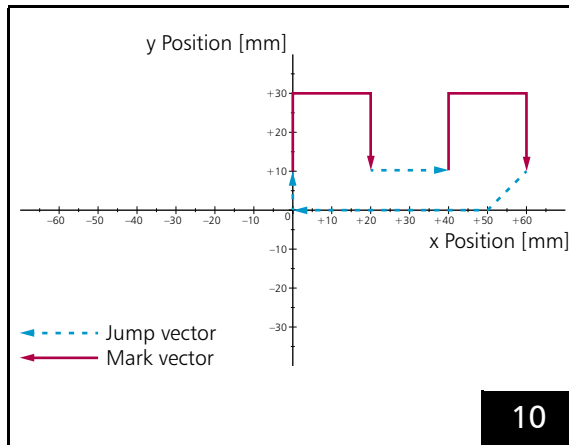
Example

The following table lists example parameter settings and simulation results for four steps of such an optimization process (using a simple marking pattern, see [Figure 10, page 40](#)).

Steps 2 and 4 are actually followed by more iteration sub-steps, because it is not initially clear which individual parameter settings will merely approach, but not exceed, the system limits.

- In iteration step 1, parameters are so conservatively set that the simulation easily achieves compliance with all system limits.
- In iteration step 2, jump speed and marking speed are increased (while the `FilterBandwidth` value remains unchanged). The simulation shows that this cuts process time considerably, but the scan head utilized its working field (possibly borderline) nearly complete.
- In iteration step 3, the `FilterBandwidth` value is increased to reduce the scan head motion participation (jump speed and marking speed are left unchanged). The simulation shows that this reduces the utilized scan head working field – with no effect on process time. But it raised positioning stage jerk to a possibly borderline range.
- In iteration step 4, jump speed and marking speed are increased even more (leading to a reduction in process time) and the `FilterBandwidth` value is lowered (although that cuts positioning stage jerk, it also enlarges the utilized scan head working field to the borderline range).

Real-world tests would be required to determine which is more advantageous: to utilize the full scan head working field or instead to transfer dynamics to the stage.



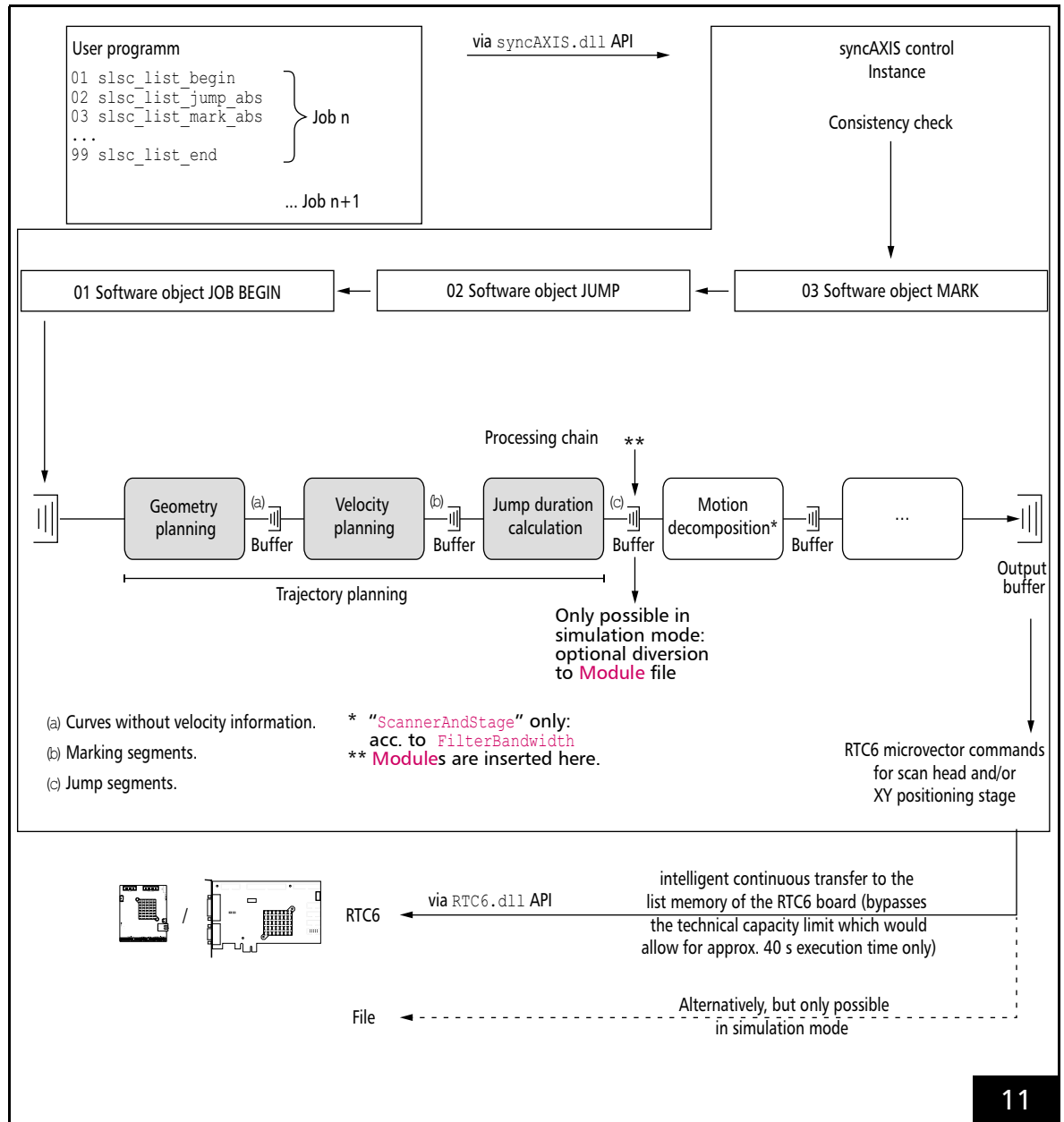
Marking pattern: two squares without base lines.

		Step 1	Step 2	Step 3	Step 4
Parameter Settings					
	Mark speed [mm/s]	480	770	770	840
	Jump speed [mm/s]	480	770	770	840
	FilterBandwidth value [Hz]	1.95	1.95	2.25	2.10
Simulation Results					
	Process time, relative to process time of 600 ms from step 1	100%	71%↓ ^(a)	71%	68%↓ ^(a)
	Utilized scan head working field, relative to the system limit (± 27 mm) x (± 27 mm)	63%	98%↑ ^(b)	87%↓ ^(a)	99%↑ ^(b)
	Max. positioning stage speed, relative to system limit 1,000 mm/s	27%	25%	29%	27%
	Max. positioning stage acceleration, relative to system limit 10,000 mm/s ²	38%	44%	58%	51%
	Max. positioning stage jerk, relative to system limit 100,000 mm/s ³	52%	72%	98%↑ ^(b)	90%

(a) green: value change relative to preceding step is interpreted as "advantageous".

(b) yellow: value is interpreted as "borderline".

2.7 About Processes at Run Time of the User Program



A **Job** is submitted to the **syncAXIS control instance** function-by-function. These are converted into single **Job** elements (software objects). Each **Job** element passes sequentially the process chain. Buffering times add to different processing times. Therefore, the status of a **Job** is a combined status out of the statuses of the single **Job** elements, see also [Figure 12, page 43](#) and [Figure 13, page 44](#).

2.7.1 About the Buffers of the syncAXIS control Instances

Buffers are syncAXIS-DLL-internal temporary memories. The software objects (as "JOB BEGIN", "JUMP", "MARK etc.) pass this processing chain, see [Figure 11, page 41](#). Buffers are necessary because there are processing times within the individual processing chain links and these are differing for different software objects. At the beginning of the processing chain is the **Input buffer**.

- When loading the buffers, user must observe the following: when the **Input buffer** is full, then syncAXIS-DLL cannot execute the called Job function (`slsc_list_*`). Depending on the `slsc_ListHandlingMode`, this can have different consequences.
 - `slsc_ListHandlingMode_ReturnAtOnce`
The last called Job function (`slsc_list_*`) is *ignored* (no exception is thrown). Due to this, some parts may possibly be missing in the marking result. To avoid this case, the following measures can be taken in the user program:
 - Prior to loading buffers, it can be queried whether the **Input buffer** is full by `slsc_ctrl_is_list_input_buffer_full`.
 - The return value of a Job function (`slsc_list_*`) can be used to check whether it has been rejected (**Bit #04** is set (`BUFFER_FULL`)).
 - `slsc_ListHandlingMode_RepeatWhileBufferFull` and `slsc_ListHandlingMode_RepeatWhilePredicate`
syncAXIS-DLL completes the execution of Job functions (`slsc_list_*`) not until sufficient **Input buffer** is free again. Only then is the user program continued.
This, users must be aware that the user program (or corresponding thread) is possibly being blocked. If further functions (for example, a **Job** start by `slsc_ctrl_start_execution`) are to be executed during blocking, this must be done by asynchronous programming (or via **Callback functions**) in another thread of the user program.

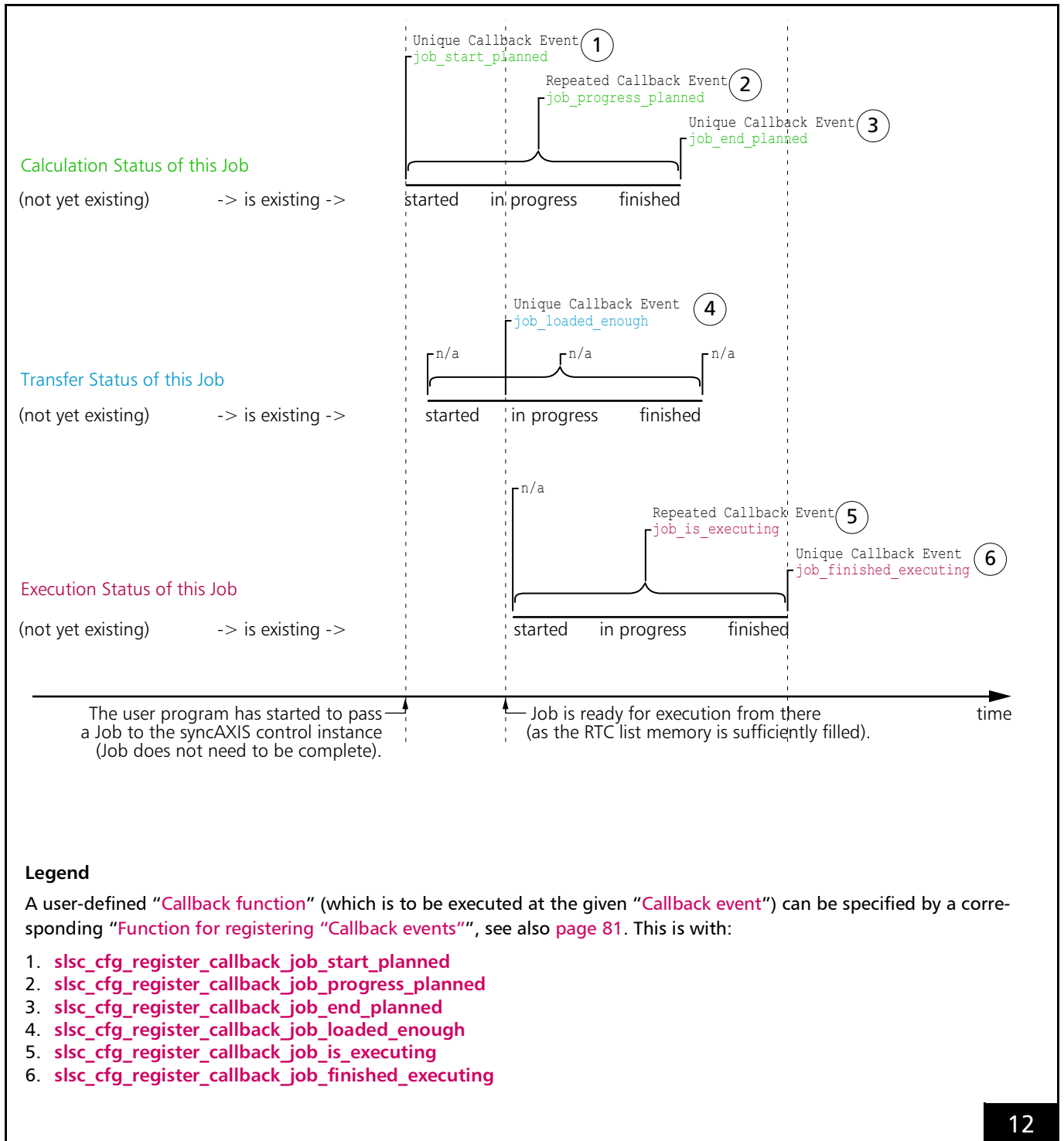
- When loading the buffers while the RTC6 board is executing a **List**, user must ensure that no **Buffer underrun** is generated (see [Glossary entry page 11](#)). See [Section "Avoiding Buffer Underruns", page 42](#).

Avoiding Buffer Underruns

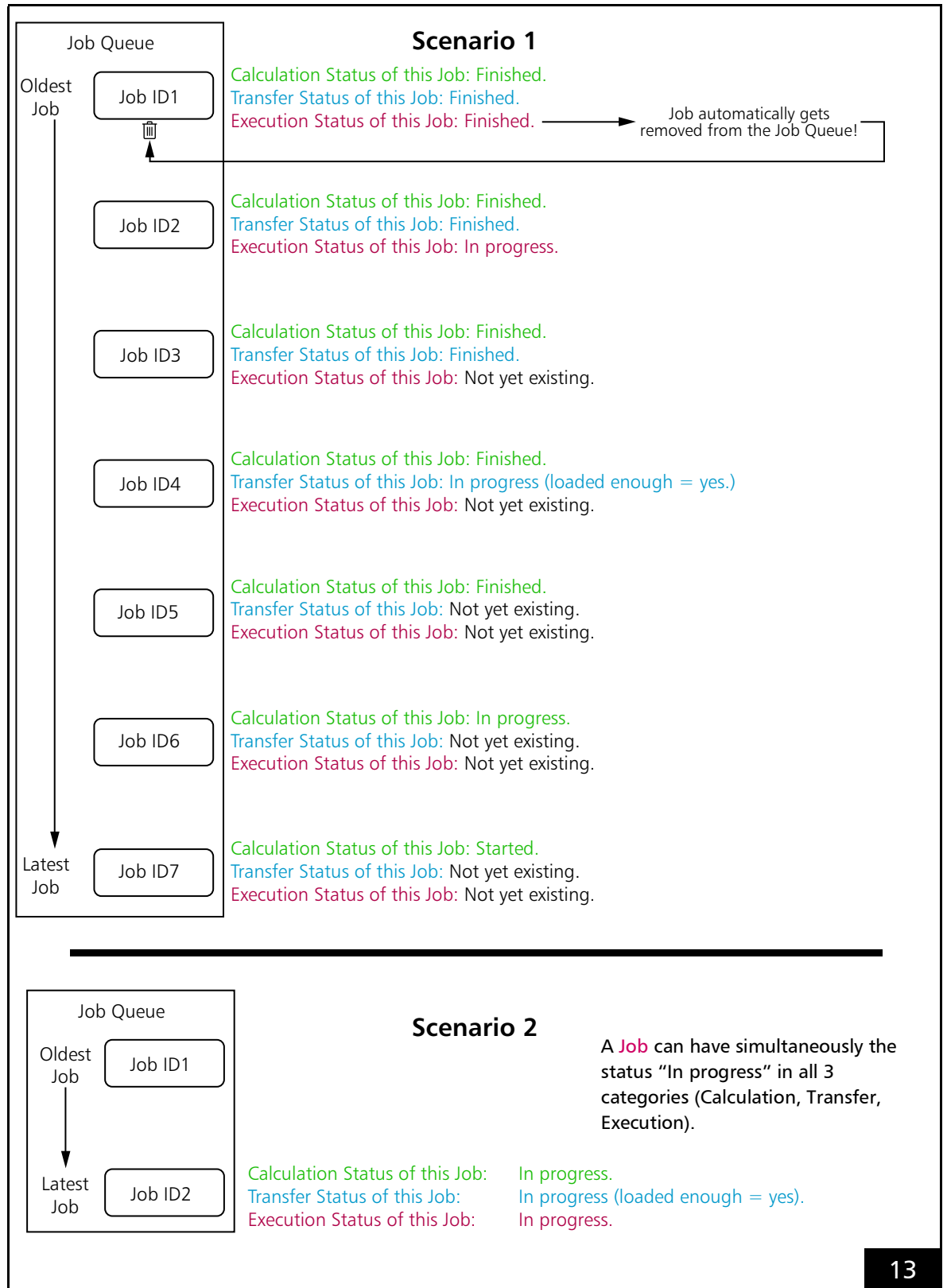
In the case of `0x 00 00 00 02 00 00 00 02`

`EXEC_BUFFER_UNDERRUN` you can take the following steps to remedy the error:

- (1) Use **Modules**, see [Chapter 2.11 "About Working with "Modules""](#), page 65.
- (2) Release PC resources, for example, by closing programs that are no longer needed.
- (3) Admit more precalculation time.
Of course, the fastest way to execute a **Job** is to start its execution when its execution state is ready to be executed = "`job_loaded_enough`". However, if you have problems with a **Buffer underrun**, you can give syncAXIS control more time for the calculation. In such cases it makes sense to let syncAXIS control precalculate the **Trajectory** as far as possible, that is, until the **Input buffer** is full. We therefore recommend that you do not start the **Job** until this buffer has been filled sufficiently.
- (4) Work with shorter **Jobs**
To make point (3) more effective, it is advisable to split long **Jobs** into several, shorter **Jobs** if you have problems with a **Buffer underrun**. This adds computing time between **Jobs**.
- (5) Parameter `VectorResolution`
Make sure that the parameter `VectorResolution` is not too finely defined. A sufficiently roughly defined `VectorResolution` can help to convert very fine input data into longer vectors.
- (6) Test your **Jobs** in advance
You can run a **Job** several times with the laser disabled to ensure that there is no risk of a **Buffer underrun**.



A **Job** is characterized by 3 different statuses of calculation, transfer and execution (which are not even exist at first). At certain instances, "Callback events" are generated. During Calculation ("In progress"), RTC6 micro vector commands are continuously generated out of the Job functions (`slsc_list_*`). These are submitted to the RTC6 board (Transfer). On Execution start, the RTC6 board begins to execute the RTC6 micro vector commands.



Job queue: two example scenarios (they are mutual exclusive).

2.7.2 About the Point in Time when Output Signals are actually set

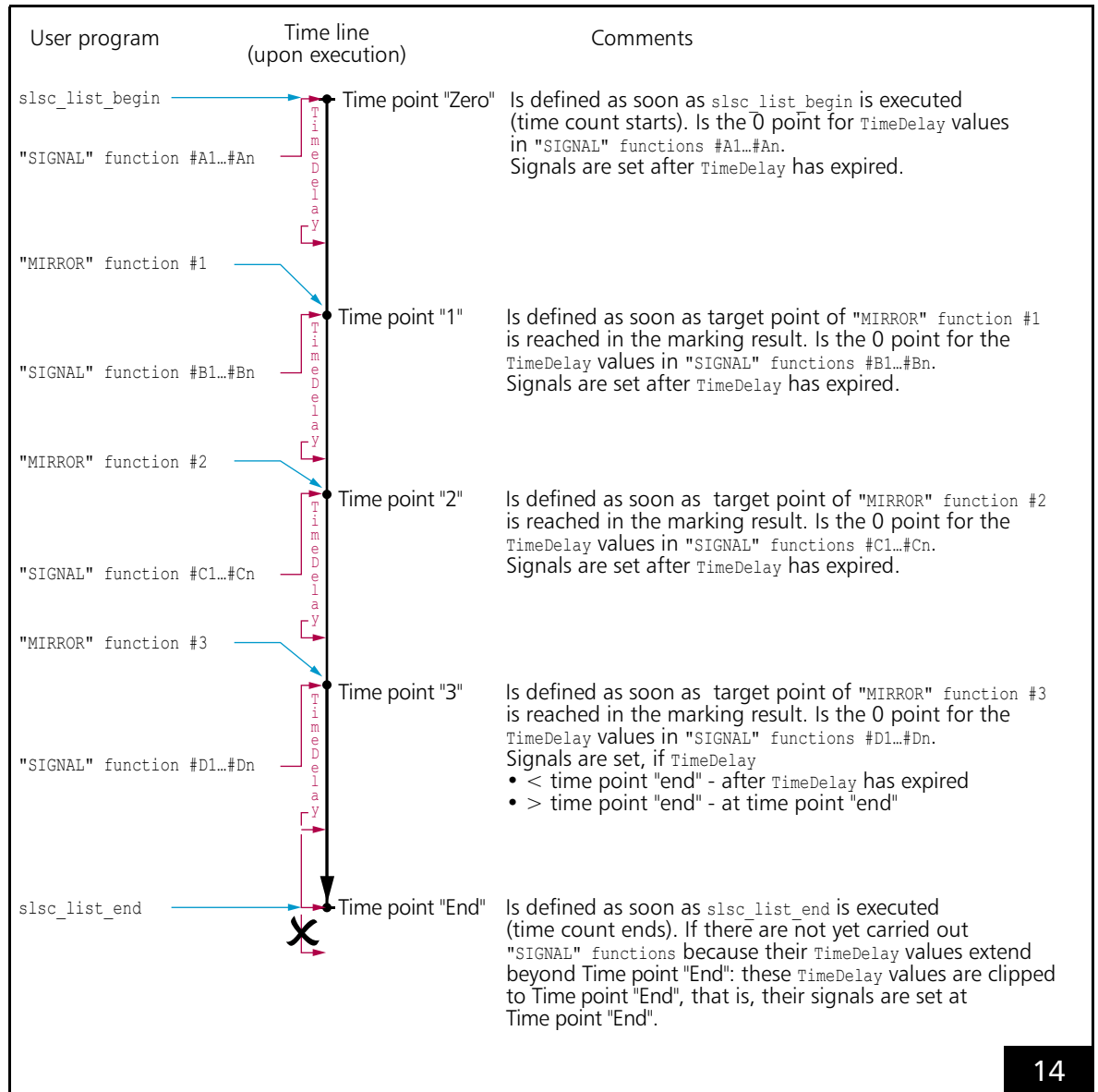
Users cannot predict exactly when an output signal (specified by “**SIGNAL**” functions) will actually be set (the execution time of the individual “**MIRROR**” functions are determined upon calculation). The following Figure 14, page 46 describes the interrelationships by way of example.

“SIGNAL” function

- **slsc_list_set_free_variable**
- **slsc_list_set_laser_pulses**
- **slsc_list_suppress_spotdistance_control**
- **slsc_list_unsuppress_spotdistance_control**
- **slsc_list_write_analog_x**
- **slsc_list_write_digital_out**
- **slsc_list_write_digital_out_mask**

“MIRROR” function

- | | |
|---|--|
| • slsc_list_arc_abs | • slsc_list_dashed_arc_abs |
| • slsc_list_multi_para_arc_abs | • slsc_list_multi_para_dashed_arc_abs |
| • slsc_list_para_arc_abs | • slsc_list_para_dashed_arc_abs |
| • slsc_list_circle_2d_abs | • slsc_list_dashed_circle_2d_abs |
| • slsc_list_multi_para_circle_2d_abs | • slsc_list_multi_para_dashed_circle_2d_abs |
| • slsc_list_para_circle_2d_abs | • slsc_list_para_dashed_circle_2d_abs |
| • slsc_list_jump_abs | • slsc_list_jump_abs_min_time |
| • slsc_list_para_jump_abs | • slsc_list_para_jump_abs_min_time |
| • slsc_list_mark_abs | • slsc_list_dashed_mark_abs |
| • slsc_list_multi_para_mark_abs | • slsc_list_multi_para_dashed_mark_abs |
| • slsc_list_para_mark_abs | • slsc_list_para_dashed_mark_abs |
| • slsc_list_wait_with_laser_off | |
| • slsc_list_wait_with_laser_on | |



The syncAXIS control instance sets output signals in relation to *preceding* time points.

2.8 About the Logging in syncAXIS control

For advanced problem analysis you can use the syncAXIS control logging mechanism. It is not only an efficient monitoring tool for users but also provides better and easier support from SCANLAB in case of problems.

Logging can be configured in the `syncAXISConfig.xml`, see [Figure 15](#).

The log file⁽¹⁾ is saved in a text file format at the location specified by the Parameter `LogfilePath`.

The `Loglevel` set defines the kind of messages that are logged:

- **Error**
 - Triggers **[ERROR]** log file lines, if applicable
 - These denote errors that actually stop the system and put it to an error state
- **Warn**
 - Subsumes **Error**
 - Triggers **[WARN]** log file lines in addition, if applicable
 - These may denote a potential problem. However, the system is *not* put to an error state. Examples: dynamic limit violations (in serious cases, an additional error would be passed from the **ACS** subsystem) and a marking speed reduction with an positioning stage-only motion. These messages do not necessarily indicate faulty behavior. Therefore, it is up to the user to react to them appropriately.

- **Info**
 - Subsumes **Warn** and **Error**
 - Triggers **[INFO]** log file lines in addition, if applicable
 - These include all messages visible to the user. Communication as well as **Job** information are logged.

Notes

- See also [Chapter 5 "Error Codes with slsc_ctrl_get_error, Log File and Console"](#), page 282.
- After the first syncAXIS control instance has been built, the logging settings from the respective `syncAXISConfig.xml` are *not* considered when building further syncAXIS control instances. Therefore:
 - Further syncAXIS control instances use the same logging settings as the first syncAXIS control instance
 - Further syncAXIS control instance write their log file lines into the log file of the first syncAXIS control instance
 - These log file lines also indicate from which thread a message originates.
 - In a log file line – which is written because a syncAXIS-DLL function returns with an error code $\neq 0$, see [Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions"](#), page 279 – the corresponding syncAXIS control instance is also denoted.

(1) syncAXIS-DLL \geq V1.5.0: only 1 log file.

```
<cfg:LogConfig>
  <cfg:LogfilePath>Log/Log.txt</cfg:LogfilePath>
  <cfg:Loglevel>Info</cfg:Loglevel>
  <cfg:EnableConsoleLogging>true</cfg:EnableConsoleLogging>
  <cfg:EnableFileLogging>true</cfg:EnableFileLogging>
  <cfg:MaxLogfileSize>26214400</cfg:MaxLogfileSize>
  <cfg:MaxBackupFileCount>0</cfg:MaxBackupFileCount>
</cfg:LogConfig>
```

15

Example: Section from an `syncAXISConfig.xml`.

2.9 About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")

Notice!

The "Automatic Laser Control" can only be used reasonably, if your laser is able to react to the signal parameter changes (see below).

A **syncAXIS control instance** can be set to periodically generate and output values during **Job** execution by the RTC6 (every 10 μ s, more often for **SpotDistance** – see column on the right).

This feature is primarily intended to get the used laser automatically controlled by the **syncAXIS control instance**. Therefore, it is referred to as "Automatic Laser Control" in syncAXIS control.

The possible "manner" of the output is referred to as "Channel" in syncAXIS control (in the **RTC6 Manual**, the term "signal parameter" is used).

The actual "manner" of the output is referred to as "ActiveChannel".

2.9.1 Activation of the "Automatic Laser Control"

Upon initialization of the **syncAXIS control instance**, the "Automatic Laser Control" is switched on, if

- at least one Channel is defined, see **Chapter 2.9.2 "Definition of the Channels and ActiveChannel"**, **page 48**, and
- this channel is entered as "ActiveChannel" in **syncAXISConfig.xml**, see **Figure 16, page 49**.

2.9.2 Definition of the Channels and ActiveChannel

The settings on Channel and ActiveChannel are done in **syncAXISConfig.xml**. They are shown in **Figure 16, page 49** (on settings which can be controlled by the user program, see **Figure 17, page 52**).

Up to 5 channels can be *defined*:

- **AnalogOut1**
[VoltageFraction]
Corresponds to the RTC6 signal parameter ANALOG OUT1. The signal is outputted every 10 μ s at the dedicated RTC6-Pin.
- **AnalogOut2**
[VoltageFraction]
Corresponds to the RTC6 signal parameter ANALOG OUT2. The signal is outputted every 10 μ s at the dedicated RTC6-Pin.
- **PulseLength**
[s]
Corresponds to the RTC6 signal parameter PulseLength. Depending on the values for **LaserMode** and **LaserPortCfg** (entered in **syncAXISConfig.xml**) the signal is changed and outputted every 10 μ s at the dedicated RTC6-Pin.
- **HalfPeriod**
[s]
Corresponds to the RTC6 signal parameter HalfPulsePeriod. Depending on the values for **LaserMode** and **LaserPortCfg** (entered in **syncAXISConfig.xml**) the signal is changed and outputted every 10 μ s at the dedicated RTC6-Pin.
- **SpotDistance**
[mm]
The signal is outputted – depending on the values for **LaserMode** and **LaserPortCfg** (entered in **syncAXISConfig.xml**) – the signal is outputted at the same RTC6-Pin as **HalfPeriod**. *However, the alteration rate is much higher than the 10 μ s of HalfPeriod.*
SpotDistance bases on the laser spot speed. It is transmitted at 10 μ s intervals to the RTC6. There a conversion to suitable laser spot distances is carried out. **Factor Ir**, **Factor Iv**, **Factor Ip** do have an effect onto the laser spot speed.


```

<cfg:AutomaticLaserControl>
  <cfg:ActiveChannel>
    <cfg:Channel>AnalogOut2</cfg:Channel>
    <cfg:Channel>SpotDistance</cfg:Channel>
  </cfg:ActiveChannel>
  <cfg:AnalogOut1 DefaultOutput="0.5" Format="VoltageFraction">
    <cfg:RadiusFactor Enabled="true">
      <cfg:DataPoint Unit="mm" Radius="0" Factor="0.9" />
      <cfg:DataPoint Unit="mm" Radius="27" Factor="1" />
    </cfg:RadiusFactor>
    <cfg:VelocityFactor Enabled="true">
      <cfg:DataPoint Unit="mm/s" Velocity="0" Factor="0.9" />
      <cfg:DataPoint Unit="mm/s" Velocity="400" Factor="1.0" />
      <cfg:DataPoint Unit="mm/s" Velocity="4000" Factor="2.0" />
    </cfg:VelocityFactor>
    <cfg:Shift>0</cfg:Shift>
  </cfg:AnalogOut1>
  <cfg:AnalogOut2 DefaultOutput="0.5" Format="VoltageFraction">
    <cfg:RadiusFactor Enabled="false">
      <cfg:DataPoint Unit="mm" Radius="0" Factor="0.9" />
      <cfg:DataPoint Unit="mm" Radius="27" Factor="1" />
    </cfg:RadiusFactor>
    <cfg:VelocityFactor Enabled="false">
      <cfg:DataPoint Unit="mm/s" Velocity="0" Factor="0.9" />
      <cfg:DataPoint Unit="mm/s" Velocity="400" Factor="1.0" />
      <cfg:DataPoint Unit="mm/s" Velocity="4000" Factor="2.0" />
    </cfg:VelocityFactor>
    <cfg:Shift>0</cfg:Shift>
  </cfg:AnalogOut2>
  <cfg:PulseLength DefaultOutput="1e-5" Unit="s">
    <cfg:RadiusFactor Enabled="true">
      <cfg:DataPoint Unit="mm" Radius="0" Factor="0.9" />
      <cfg:DataPoint Unit="mm" Radius="27" Factor="1" />
    </cfg:RadiusFactor>
    <cfg:VelocityFactor Enabled="true">
      <cfg:DataPoint Unit="mm/s" Velocity="0" Factor="0.9" />
      <cfg:DataPoint Unit="mm/s" Velocity="400" Factor="1.0" />
      <cfg:DataPoint Unit="mm/s" Velocity="4000" Factor="2.0" />
    </cfg:VelocityFactor>
    <cfg:Shift>0</cfg:Shift>
  </cfg:PulseLength>
  <cfg:HalfPeriod DefaultOutput="1e-5" Unit="s">
    <cfg:RadiusFactor Enabled="true">
      <cfg:DataPoint Unit="mm" Radius="0" Factor="0.9" />
      <cfg:DataPoint Unit="mm" Radius="27" Factor="1" />
    </cfg:RadiusFactor>
    <cfg:VelocityFactor Enabled="true">
      <cfg:DataPoint Unit="mm/s" Velocity="0" Factor="0.9" />
      <cfg:DataPoint Unit="mm/s" Velocity="400" Factor="1.0" />
      <cfg:DataPoint Unit="mm/s" Velocity="4000" Factor="2.0" />
    </cfg:VelocityFactor>
    <cfg:Shift>0</cfg:Shift>
  </cfg:HalfPeriod>
  <cfg:SpotDistance DefaultOutput="0.005" Unit="mm">
    <cfg:RadiusFactor Enabled="false">
      <cfg:DataPoint Unit="mm" Radius="0" Factor="0.9" />
      <cfg:DataPoint Unit="mm" Radius="26" Factor="0" />
      <cfg:DataPoint Unit="mm" Radius="27" Factor="1.0" />
    </cfg:RadiusFactor>
    <cfg:VelocityFactor Enabled="false">
      <cfg:DataPoint Unit="mm/s" Velocity="0.0" Factor="0.9" />
      <cfg:DataPoint Unit="mm/s" Velocity="400.0" Factor="1.0" />
      <cfg:DataPoint Unit="mm/s" Velocity="800.0" Factor="1.2" />
    </cfg:VelocityFactor>
    <cfg:Shift>1e5</cfg:Shift>
  </cfg:SpotDistance>
</cfg:AutomaticLaserControl>

```

“ActiveChannel” = Channel to be used [SpotDistance AND 1 Channel] OR [1 Channel]

Channel AnalogOut1
Definitions effective with:
<cfg:Channel>AnalogOut1</cfg:Channel>

Channel AnalogOut2
Definitions effective with:
<cfg:Channel>AnalogOut2</cfg:Channel>

Channel PulseLength
Definitions effective with:
<cfg:Channel>PulseLength</cfg:Channel>

Channel HalfPeriod
Definitions effective with:
<cfg:Channel>HalfPeriod</cfg:Channel>

Channel SpotDistance
Definitions effective with:
<cfg:Channel>SpotDistance</cfg:Channel>

Legend

1. DefaultOutput attribute value
2. Ir is to be used (true) / not (false).
3. Ir characteristic data records (for 2.=true).
4. Iv is to be used (true) / not (false).
5. Iv characteristic data records (for 4.=true).
6. Shift (optionally, as of V1.2), see [Shift](#).

For each Channel you can define:

- Its DefaultOutput value
- Whether the output value calculation for this channel has to take the **Factor Ir** = “radius factor” into account or not.
Factor Ir is considered for the calculation, if enabled = true. This requires that the corresponding characteristic interpolation points (**DataPoint** tags) must have been entered.
- Whether the output value calculation for this channel has to take the **Factor Iv** = “velocity factor” into account or not.
Factor Iv is considered for the calculation, if enabled = true. This requires that the corresponding characteristic interpolation points (**DataPoint** tags) must have been entered.
- When the Channel is an “ActiveChannel”: whether the output values of this Channel are to be brought forward or to be postponed. As of V1.2, **Shift** tags are available for this purpose.

For information on calculating the output values, see [Figure 17, page 52](#).

However, only 1 or 2 of the (up to 5) *defined* channels can actually be *used*. For this, it/they must have been entered as ActiveChannel:

- 1 channel of your choice
- 1 channel of your choice together with the Channel **SpotDistance**.


Notes

- If there is no ActiveChannel entry, then the section AutomaticLaserControl is not evaluated upon initialization of the **syncAXIS control instance**. Then the “Automatic Laser Control” is not active (is not used).
- Even if syncAXIS control offers similar functionalities as the RTC6 command **set_auto_laser_control**, this RTC6 command is not used internally (syncAXIS control has a completely independent and self-contained implementation).
- If the “Automatic Laser Control” is active and **SpotDistance** is an “ActiveChannel” then the **slsc_ctrl_set_laser_pulses** and **slsc_list_set_laser_pulses** parameters **PulseLength** are effective (that is, pulse lengths of laser signal LASER1 and LASER2 are changed). However, their **HalfPeriod** parameters are not effective.

2.9.3 About how ActiveChannel Values along a Contour are Calculated

The exact value set by the **syncAXIS control instance** along a contour for each channel is the result of a calculation, see **Figure 17, page 52**.

The DefaultOutput value (see **Figure 16, page 49**) is multiplied by up to 3 (provided these are enabled = `true`) factors:

- Factor **lr**
"Radius factor". Value of the characteristic for the present excursion radius = distance from scan head working field 0,0.

- Factor **lv**
"Velocity factor". Value of the characteristic for the present laser spot velocity in the working field.

- Factor **lp**
Ramp factor. **slsc_list_para_enable** switches its processing on. Otherwise, it is *not* applied.
 - A) With **slsc_list_*** functions
Factor lp remains constant. It either equals the **ParaTargetDefault** value or, the target value which has been defined by the preceding **slsc_list_[multi_para/para]*** function.
 - B) With **slsc_list_para_*** functions
Factor lp is varied *linearly* along the vector length/arc length. The target value is defined by the argument **ParaTarget** of the current **slsc_list_para_*** function. The starting value is the last reached value (= **ParaTargetDefault** value or the target value defined by the previous **slsc_list_[multi_para/para]*** function).
 - C) With **slsc_list_[multi_para/para]*** functions
Factor lp is varied *linearly segment-by-segment* along the vector length/arc length (segments of more complicated **Ramps** are defined by **slsc_ParaSection**). Initial value of the 1st **Ramp** section is the last reached value (= **ParaTargetDefault** value or the target value defined by the previous **slsc_list_[multi_para/para]*** function).

Output value per 10 μ s
each ActiveChannel individually
=

DefaultOutput value.

This value is set in syncAXISConfig.xml separately for each Channel.
E.g. `<cfg:AnalogOut1 DefaultOutput="0.5" Format="...."`

×

Radius factor I_r .

Depends on syncAXISConfig.xml-setting: I_r is applied only, if this channel has `<cfg:RadiusFactor Enabled="true">`.
Then, the I_r value which fits the scan head excursion radius is picked from the characteristic (`=cfg:DataPoint` tags under `cfg:RadiusFactor`).

×

Velocity factor I_v .

Depends on syncAXISConfig.xml-setting: I_v is applied only, if this channel has `<cfg:VelocityFactor Enabled="true">`.
Then, the I_v value which fits the laser spot speed is picked from the characteristic (`=cfg:DataPoint` tags under `cfg:VelocityFactor`).

×

Ramp factor I_p .

Controlled by the source code: I_p is only used after `slsc_list_para_enable`.
The actually used I_p value is picked from "ramps" defined with `slsc_list_[multi_para/para]` functions.

17

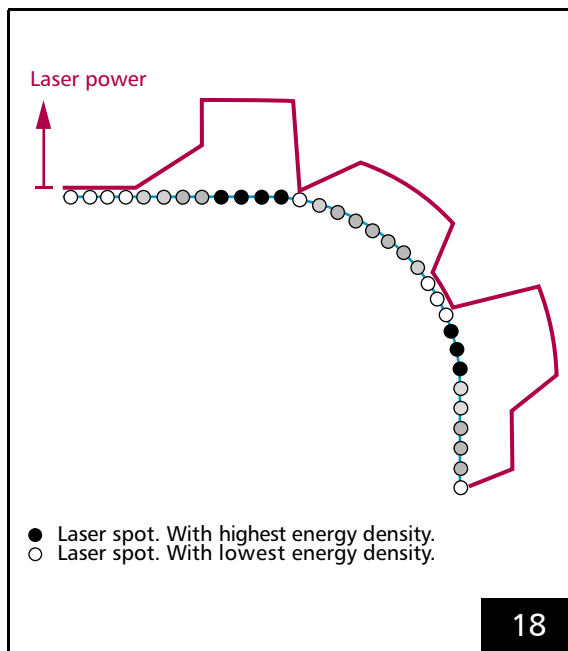
Calculation of the Output value. This is done separately for each ActiveChannel.

2.9.4 About Ramps

Ramps are defined in the source code by `slsc_list_[para/multi_para]*` functions, see [Section "Functions for Defining Ramps \(slsc_list_\[para/multi_para\]*-Functions\)"](#), page 92. That is, with syncAXIS control, the individual Ramp points are defined along with position coordinates.

Ramps of different shapes can be modeled with syncAXIS control, for example \sqcap , \sqcap , \sqcap , \sqcap .

Among other things, they are used for automatic variation of the laser power, if the marking substrate along the curve exhibits varying light absorption (that is, the work piece material consists of areas that are more sensitive to absorption and less sensitive to absorption), see [Figure 18, page 53](#).

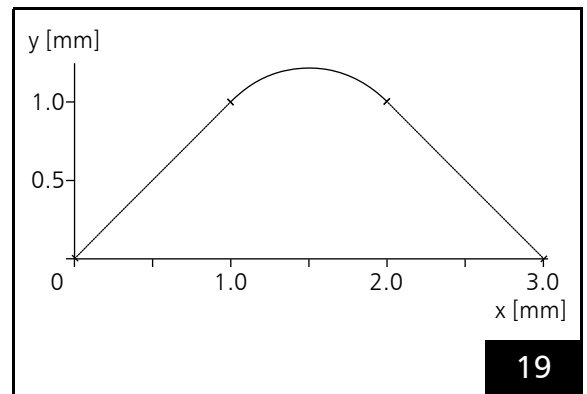


Varying the laser power.

Two concrete examples are described below:

- "Example – Linear (Simple) Ramp", page 54
- "Example – Multi-part (more Complex) Ramp", page 57

Both examples use the same marking pattern: jump to initial position – straight line – arc – straight line, see [Figure 19, page 53](#) (jumps not shown there).



2D positions of the both examples in this Section. Jumps not shown.

Example – Linear (Simple) Ramp

Source Code

- See [Figure 20](#).

Explanatory Notes on the Source Code

- Marking pattern: jump to initial position – straight line – arc – straight line, see [Figure 19](#), [page 53](#) (jumps not shown there).
- By `slsc_list_para_enable`
 - the processing of `TargetPara` arguments is enabled
 - `TargetParaDefault` is set = 1
- By `slsc_list_para_arc_abs` it is defined
 - the arc
 - one linear Ramp

Explanatory Notes on the Simulation Result

- The simulation result is shown in [Figure 21](#), [page 56](#)
- The Ramp starts with the arc
- Before and at the beginning of the Ramp the output value is 0.5 (because `DefaultOutput="0.5"` and `TargetParaDefault = 1` means “no change”)
- The target output end value is the last reached value × [Factor Ip](#) ($0.5 \times 0.5 = 0.25$)
- During the arc, the target output start value 0.5 is varied linearly to the target output end value 0.25
- Other factors do not affect this variation, because neither [Factor Ir](#) nor [Factor Iv](#) have been defined
- Beyond the end of the Ramp, the last value set remains (0.25)

Settings for the Simulation

For simulation of the code a `syncAXISConfig.xml` is used having the following settings:

- `ActiveChannel = AnalogOut2`
- Channel `AnalogOut2`
 - `DefaultOutput="0.5"`
 - `RadiusFactor Enabled="false"`
(= [Factor Ir](#) shall not be used)
 - `VelocityFactor Enabled="false"`
(= [Factor Iv](#) shall not be used)
- Deactivated splines
(`slsc_SplineModes_Deactivated` or by `slsc_cfg_set_trajectory_config`)
- Deactivated blending curves
(`slsc_BlendModes_Deactivated` or by `slsc_cfg_set_trajectory_config`)

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.
// Example: linear (=simple) Ramp by a slsc_list_para_arc_abs.
// No blending curves are activated.
// Pseudo code (not complete)

size_t JobID = 0;
slsc_list_begin(Handle, &JobID);
double TargetPosition_0[2] = { 0.0, 0.0 };
double TargetPosition_1[2] = { 1.0, 1.0 };
double TargetPosition_2[2] = { 1.025, 1.025 };
double TargetPosition_3[2] = { 2.0, 1.0 };
double TargetPosition_4[2] = { 3.0, 0.0 };

double TargetParaDefault[1] = { 1.0 };
double TargetPara_1[1] = { 0.5 };

slsc_list_jump_abs(Handle, TargetPosition_0);
slsc_list_para_enable(Handle, TargetParaDefault);
slsc_list_mark_abs(Handle, TargetPosition_1);
slsc_list_para_arc_abs(Handle, TargetPosition_2, TargetPosition_3, TargetPara_1);
slsc_list_mark_abs(Handle, TargetPosition_4);
slsc_list_jump_abs(Handle, TargetPosition_0);
slsc_list_end(Handle);
```

Code example: Linear (simple) Ramp

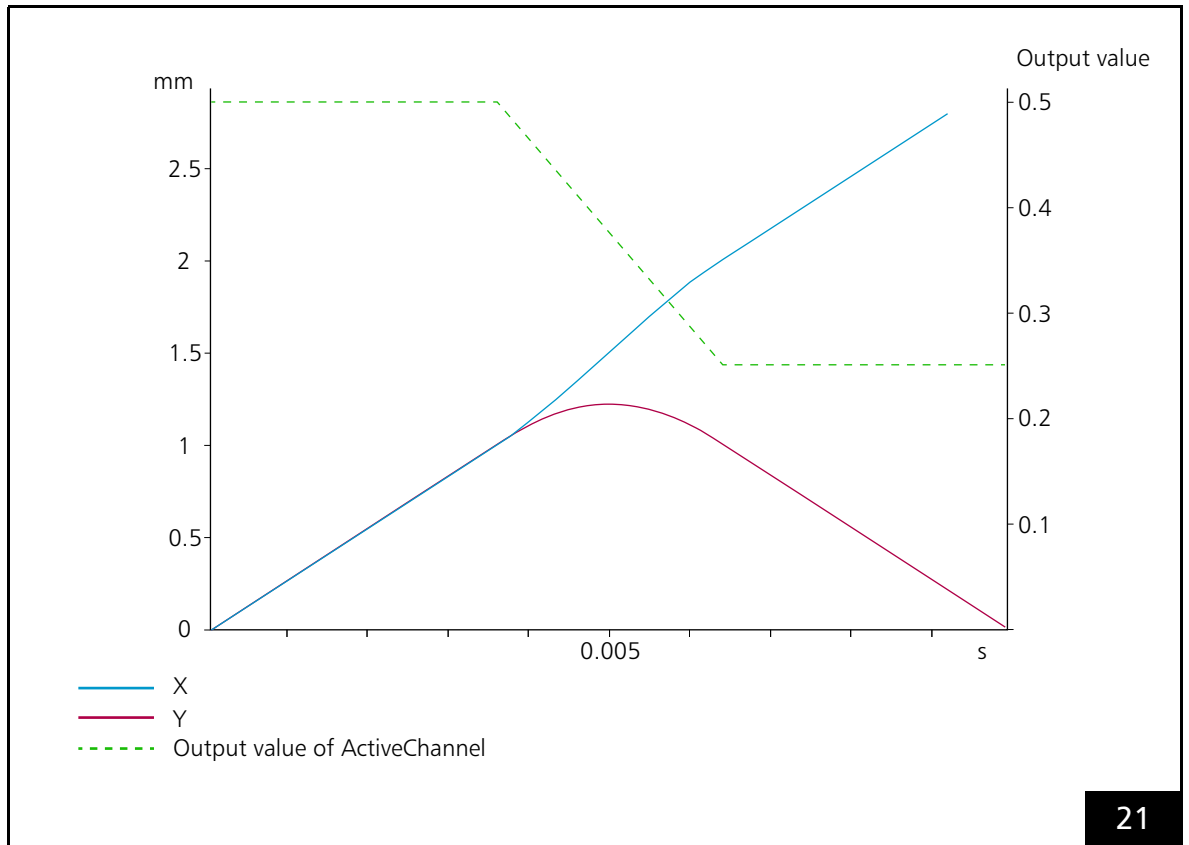


Diagram to **Section "Example – Linear (Simple) Ramp"**, page 54.

Simulation result: temporal course (values from the **Trajectory planning**) of X and Y positions, as well as of the linear (simple) **Ramp**. ActiveChannel = AnalogOut2, no **Factor Ir**, no **Factor Iv**. No splines, no blending curves.

Ramp start value: DefaultOutput \times TargetParaDefault (0.5×1).

Ramp end value: [TargetPara_1 (= **Factor Ip**) \times last reached value] (0.5×0.5).

Example – Multi-part (more Complex) Ramp

Source Code

- See [Figure 22](#).

Explanatory Notes on the Source Code

- Marking pattern: jump to initial position – straight line – arc – straight line, see [Figure 19](#), [page 53](#) (jumps not shown there).
- By `slsc_list_para_enable`
 - the processing of `TargetPara` arguments is enabled
 - `TargetParaDefault` is set = 1
- By `slsc_list_multi_para_arc_abs` it is defined
 - the arc
 - one Ramp consisting of 2 sections
 - `ParaSection[0]...[2]`

Settings for the Simulation

For simulation of the code a `syncAXISConfig.xml` is used having the following settings:

- `ActiveChannel = AnalogOut2`
- `Channel AnalogOut2`
 - `DefaultOutput="0.5"`
 - `RadiusFactor Enabled="false"`
(= `Factor Ir` shall not be used)
 - `VelocityFactor Enabled="false"`
(= `Factor Iv` shall not be used)
- Deactivated splines
(`slsc_SplineModes_Deactivated` or by `slsc_cfg_set_trajectory_config`)
- Deactivated blending curves
(`slsc_BlendModes_Deactivated` or by `slsc_cfg_set_trajectory_config`)

Explanatory Notes on the Simulation Result

- The simulation result is shown in [Figure 23](#), [page 59](#)
- The Ramp starts with the arc
- Before and at the beginning of the Ramp the output value is 0.5 (because `DefaultOutput="0.5"` and `TargetParaDefault = 1` means "no change")
- 1st Ramp section
 - Target output end value (a) is the `DefaultOutput` value × `Factor Ip@ParaSection[0]` ($0.5 \times 0.5 = 0.25$)
 - This value shall be reached after 0.25 mm marking distance
 - In the process, it is linearly (declining) varied
- 2nd Ramp section
 - Target output end value (b) is the `DefaultOutput` value × `Factor Ip@ParaSection[1]` ($0.5 \times 0.5 = 0.25$)
 - This value shall be reached after 0.61(...) mm marking distance
 - the output value remains constant (because start value and end value are 0.25)
- 3rd Ramp section
 - Target output end value (c) is the `DefaultOutput` value × `Factor Ip@ParaSection[2]` ($0.5 \times 1.0 = 0.5$)
 - This value shall be reached after 0.25 mm marking distance
 - In the process, it is linearly (increasing) varied
- Other factors do not affect this variation, because neither `Factor Ir` nor `Factor Iv` have been defined
- Beyond the end of the Ramp, the last value set remains (0.5)

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.
// Example MultiPara by slsc\_list\_multi\_para\_arc\_abs
// No blending curves are activated.
// Pseudo code (not complete)

size_t JobID = 0;
slsc_list_begin(Handle, &JobID);
double TargetPosition_0[2] = { 0.0, 0.0 };
double TargetPosition_1[2] = { 1.0, 1.0 };
double TargetPosition_2[2] = { 1.025, 1.025 };
double TargetPosition_3[2] = { 2.0, 1.0 };
double TargetPosition_4[2] = { 3.0, 0.0 };

slsc_list_jump_abs(Handle, TargetPosition_0);
slsc_list_mark_abs(Handle, TargetPosition_1);
double TargetParaDefault[1] = { 1.0 };
slsc_list_para_enable(Handle, TargetParaDefault);

// Here is created: array of type slsc_ParaSection with dimension 3.
slsc_ParaSection ParaSection[3];
ParaSection[0] = slsc_ParaSection{ 0.25, 0.5 };
ParaSection[1] = slsc_ParaSection{ 0.6186678697087737, 0.5 };
ParaSection[2] = slsc_ParaSection{ 0.25, 1.0 };

// Here is created: array of type slsc\_MultiParaTarget with dimension 1 (due ActiveChannel == 1).
slsc\_MultiParaTarget MultiTargetPara[1];
MultiTargetPara[0].Targets = ParaSection;
MultiTargetPara[0].NumParaTargets = 3;
slsc\_list\_multi\_para\_arc\_abs(Handle, TargetPosition_2, TargetPosition_3, MultiTargetPara);

slsc_list_mark_abs(Handle, TargetPosition_4);
slsc_list_jump_abs(Handle, TargetPosition_0);
slsc_list_end(Handle);
```

Code example: Multi-part (more complex) [Ramp](#)

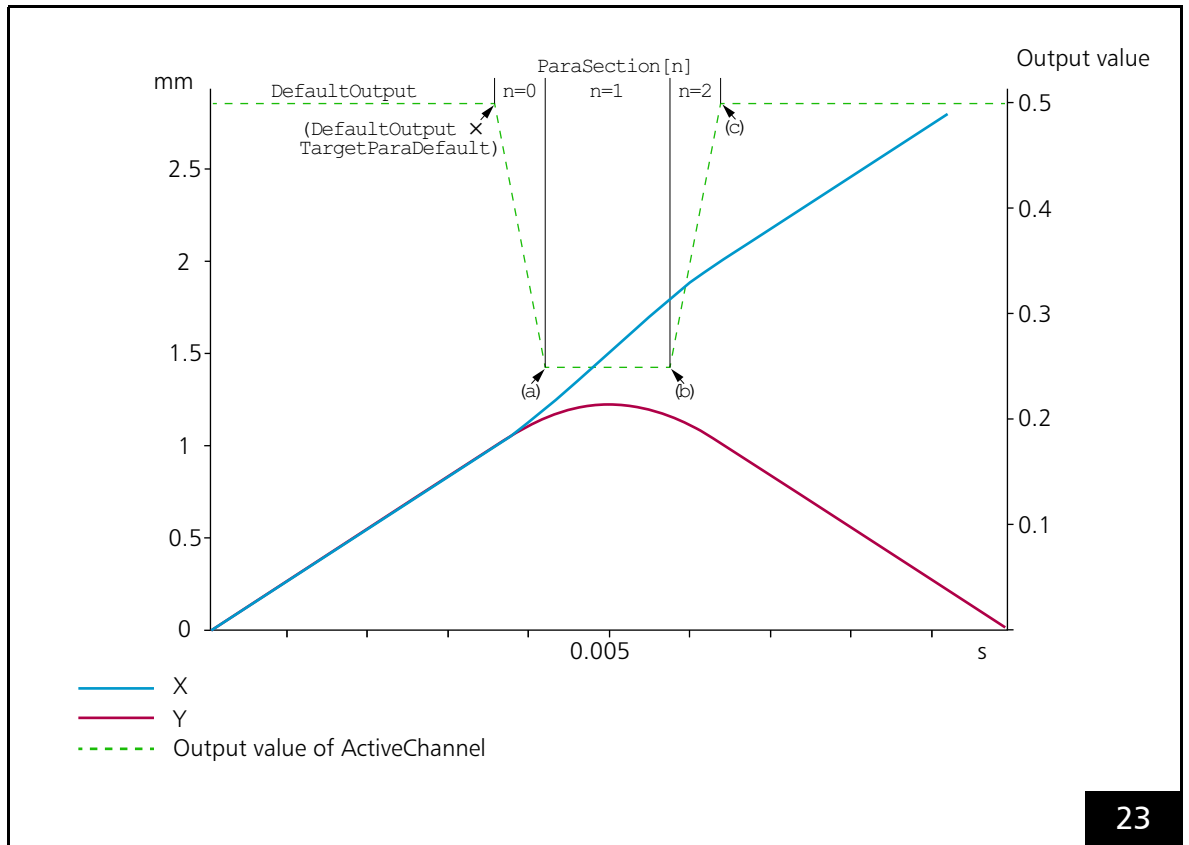


Diagram to Section "Example – Multi-part (more Complex) Ramp", page 57.

Simulation result: temporal course (values from the Trajectory planning) of X and Y positions, as well as of the multi-part (more complex) Ramp. ActiveChannel = AnalogOut2, no Factor I_r , no Factor I_v . No splines, no blending curves. For further details see text.

2.9.5 About the “Contour-dependent speed calculation”

“Contour-dependent speed calculation” is a functionality of the “Automatic Laser Control”.

The “Contour-dependent speed calculation” requires that the “Automatic Laser Control” is configured in `syncAXISConfig.xml` (at least 1 channel is defined and entered as “ActiveChannel”) and switched on (initializing the `syncAXIS` control instance with this `syncAXISConfig.xml`), see Chapter 2.9.1 “Activation of the “Automatic Laser Control””, page 48.

After initialization of the `syncAXIS` control instance by `slsc_cfg_initialize_from_file` the “Contour-dependent speed calculation” is *not* switched on at first.

The main use case for “Contour-dependent speed calculation” is that users can influence on which calculation basis the energy input along curves is to take place (and thus ultimately the marking result).

Example: there are (although laser spots are equidistant and there is a constant marking speed) burn-ins with curves, and some of these burn-ins extend into areas that should actually be used as a workpiece, see Figure 24, page 61.

To achieve evenness of the signal output (at all “ActiveChannel”, for example, equidistance of the laser spot distances with `SpotDistance`), the `syncAXIS` control instance calculates – without “Contour-dependent speed calculation” – their output in relation to the speed along the *middle* of the contour line. This case is illustrated in Figure 24, page 61.

Users can change this calculation basis (for this speed) by the following functions:

- `slsc_cfg_set_contour_dependent_speed_control_2d`
- `slsc_list_set_contour_dependent_speed_control_2d`

By selecting suitable parameters it can be set whether the speed is calculated related to

- a certain distance to the *middle* of the contour line (`SpotRadius`), and
- furthermore, whether this distance shall be
 - to the right of the contour (`Direction = +1`) see Figure 25, page 62
 - to the left of the contour (`Direction = -1`) see Figure 26, page 63.

Notes

- “Contour-dependent speed calculation” is switched off by:
 - `slsc_cfg_set_contour_dependent_speed_control_2d` with `Direction=0` or `SpotRadius=0`
 - `slsc_list_set_contour_dependent_speed_control_2d` with `Direction=0` or `SpotRadius=0`
- For the ActiveChannel `SpotDistance` (only) it is possible with these two functions to refer the spot distance calculation to the speed at the inner radius or outer radius (instead of the contour line center) and then (during the **Job** execution) to actually output the laser spots more often than only in a 10 μ s clock cycle.
- For all ActiveChannels (also `SpotDistance`) the following applies:
 - if the “Contour-dependent speed calculation” is *switched on*:
`Factor Iv` is related to the above described speed
 - if the “Contour-dependent speed calculation” is *switched off*:
`Factor Iv` is related to the laser spot speed in the working field

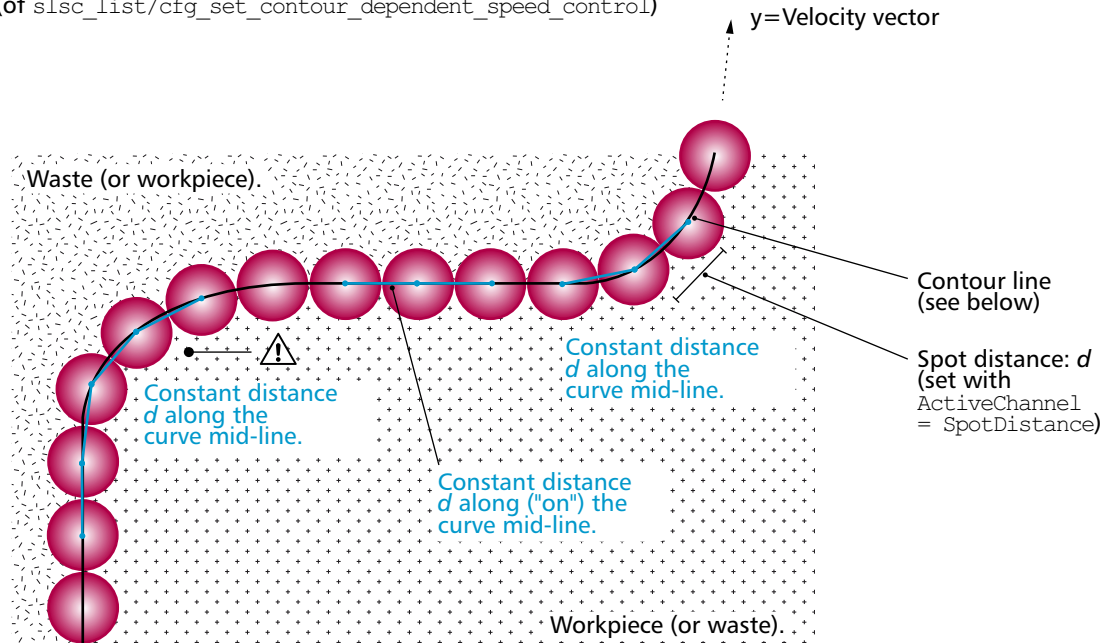
Automatic Laser Control = ON:

- ActiveChannel = SpotDistance

Contour-dependent speed calculation = OFF:

- Direction=0

(of slsc_list/cfg_set_contour_dependent_speed_control)



24

The "Automatic Laser Control" is on, because **SpotDistance** is set as "ActiveChannel" (see [Figure 16, page 49](#)) in **syncAXISConfig.xml**. **SpotDistance** ensures that the laser spots are equidistant: in this example, however, the "Contour-dependent speed calculation" is *not* switched on (default setting, as well as by **slsc_cfg_set_contour_dependent_speed_control_2d** or **slsc_list_set_contour_dependent_speed_control_2d** with **Direction=0** or **SpotRadius=0**). Therefore, syncAXIS control calculates the evenness of the laser spot distances in relation to the middle of the contour line. The contour line must (applies to syncAXIS control $\geq V0.11$) consist of several **slsc_list_[multi_para/para]*** functions. The velocity vector derives from the **Trajectory planning** and results from the defined marking pattern.

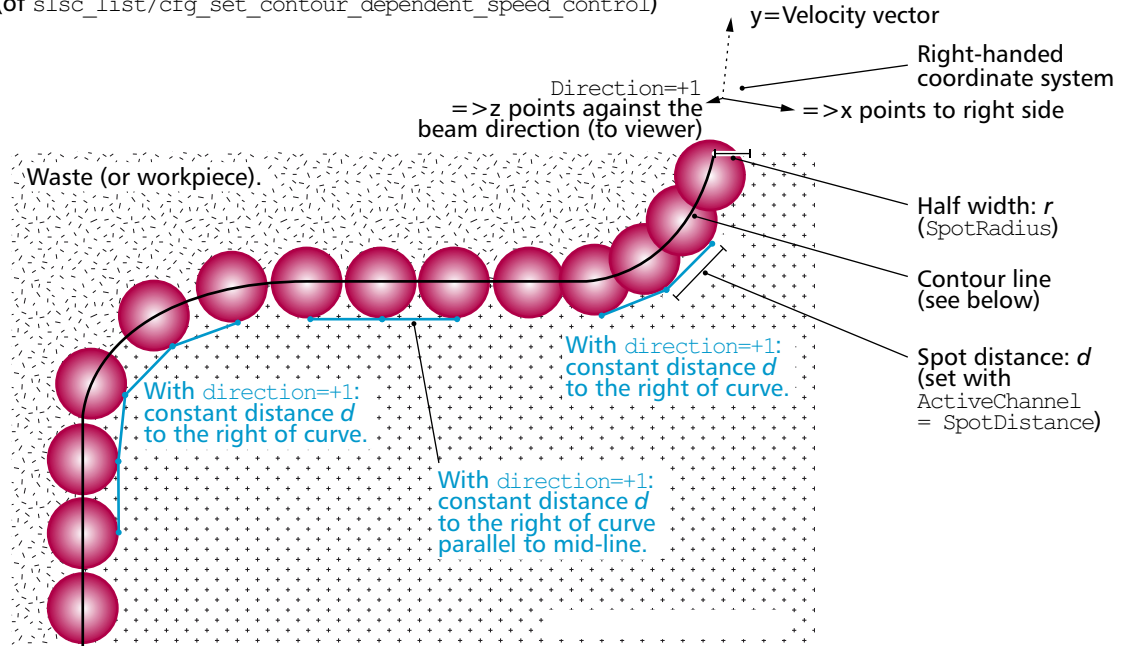
Note: In the case shown here, burn-ins in the area marked with "⚠" – despite equidistant laser spots and constant marking speed there – potentially may be observed at the designated area (which is why the "contour-dependent speed calculation" feature has been introduced for correction).

Automatic Laser Control = ON:

- ActiveChannel = SpotDistance

Contour-dependent speed calculation = ON:

- Direction=+1
(of slsc_list/cfg_set_contour_dependent_speed_control)



25

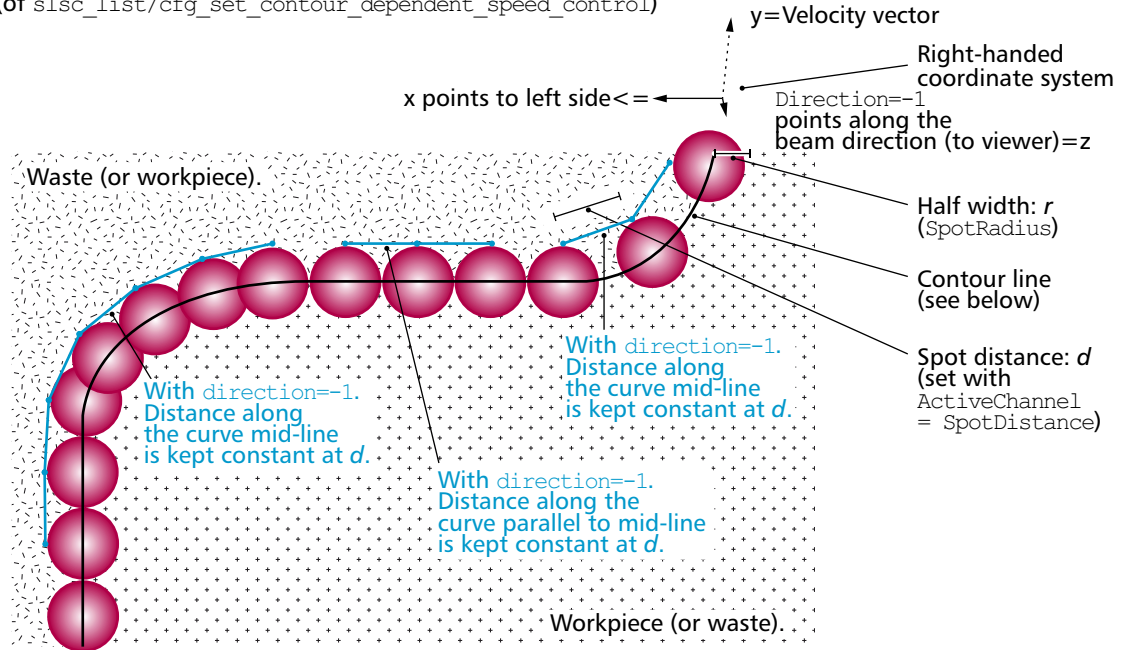
The "Automatic Laser Control" is on, because `SpotDistance` is set as "ActiveChannel" (see Figure 16, page 49) in `syncAXISConfig.xml`. `SpotDistance` ensures that the laser spots are equidistant: in this example, the "Contour-dependent speed calculation" is switched on, because `Direction=+1` is specified with `slsc_cfg_set_contour_dependent_speed_control_2d` or `slsc_list_set_contour_dependent_speed_control_2d`. Therefore, syncAXIS control calculates the evenness of the laser spot distances in relation to the "right" of the contour line. The contour line must (applies to syncAXIS control $\geq V0.11$) consist of several `slsc_list_[multi_para/para]*` functions. The velocity vector derives from the Trajectory planning and results from the defined marking pattern.

Automatic Laser Control = ON:

- ActiveChannel = SpotDistance

Contour-dependent speed calculation = ON:

- Direction=-1
(of slsc_list/cfg_set_contour_dependent_speed_control)



26

The "Automatic Laser Control" is on, because **SpotDistance** is set as "ActiveChannel" (see [Figure 16, page 49](#)) in **syncAXISConfig.xml**. **SpotDistance** ensures that the laser spots are equidistant: in this example, the "Contour-dependent speed calculation" is *switched* on, because **Direction=-1** is specified with **slsc_cfg_set_contour_dependent_speed_control_2d** or **slsc_list_set_contour_dependent_speed_control_2d**. Therefore, syncAXIS control calculates the evenness of the laser spot distances in relation to the "left" of the contour line. The contour line must (applies to syncAXIS control \geq V0.11) consist of several **slsc_list_[multi_para/para]*** functions. The velocity vector derives from the **Trajectory planning** and results from the defined marking pattern.

2.10 About **Heuristic** and Characteristics for Speed Reductions

Heuristic

- Definition incl. prerequisite, **Segment** types (incl. how to restrict to **Jump Segments** only), see Glossary entry **Heuristic**

Speed Reduction Characteristics

- Use cases see **DynamicReductionFunction** (page 473)
- Non-use case see **DynamicReductionFunction** (page 473)
- Definition location: child tags of **DynamicReductionFunctions**
 - One characteristic = one **DynamicReductionFunction** tag. Its interpolation points = **DataPoint** child tags (x values spatial length, y values speed)
 - Example see **XML section example**, page 473
 - Allowed characteristics number see **DynamicReductionFunction**
 - Default-characteristic see **DynamicReductionFunction**
 - To change the characteristic (**slsc_cfg_select_heuristic**) see **DynamicReductionFunction**
 - No characteristic – no **Heuristic** possible (**DynamicReductionFunction** is not a mandatory tag)

2.11 About Working with “Modules”

As illustrated in [Figure 11, page 41](#), the user defines a **Job** by a series of **Job functions** (**slsc_list_***) as input for the **Trajectory planning** of the syncAXIS-DLL. Its result is loaded into the RTC6 list memory⁽¹⁾ in the form of RTC6 micro vector commands. After **Job** execution has been triggered, **Trajectory planning** and **Job** execution typically run in parallel.

With a very computationally intensive **Trajectory** (for example, many very short vectors are defined in the code at high marking speeds⁽²⁾), the RTC6 list memory may be processed faster than new micro vectors can be calculated.

As soon as the RTC6 list memory has been completely processed **Job** execution is aborted due to a **Buffer underrun** (see also [Chapter 2.7.1 “About the Buffers of the syncAXIS control Instances”, page 42](#)) and XL SCAN is put into an error state.

This problem can be avoided by **slsc_list_begin_module**. In simulation mode, this function can be used to get a **Job** calculated “in advance”. The result (“**Module**”) is recorded as a “**Module file**”, see [Section “Module file”, page 66](#).

At a later time, this **Module file** can be reused in other **Jobs** (“replaying the **Module**” omitting the computation-intensive **Trajectory planning**) by either

- **slsc_list_playback_module** (parameter values on **Ramps** are not applied)
- **slsc_list_para_playback_module** (parameter values on **Ramps** are applied, if there is a **slsc_list_para_enable** in advance)

After **Module file** import position data is available to the user program. These are inserted before the processing step “**Motion decomposition**”, see [Figure 11, page 41](#).

From this step on, the further processing is fast. This is why a **Buffer underrun** no longer occurs, even if input data had been in high definition originally.

Notes

- **Motion decomposition** (**FilterBandwidth**) occurs not until the **Module** is replayed (**slsc_list_playback_module**).
- See also [Section “Behavior on Module replay”, page 66](#).
- The dynamics of the set (target) **Trajectory cannot** be changed when replaying the **Module**.
- Nevertheless, the contained **Module** can be positioned and rotated anywhere in the space by **slsc_list_set_rot_and_offset_2d**.
- From the user **Module** (recorded **Job**) and a programming using the corresponding **Job functions** (**slsc_list_***) behave largely the same. For special aspects in particular at the beginning and end of the **Module**, see **slsc_list_begin_module** and **slsc_list_playback_module**.
- Code examples:
 - General procedure with **Modules**, see [Figure 27, page 67](#)
 - To record a **Module**, see [Figure 28, page 68](#)
 - To replay a **Module**, see [Figure 29, page 69](#)
 - See also [Chapter 10 “Appendix C: Application Note – Marking Texts by Using Modules”, page 344](#)
 - See also [Chapter 11 “Appendix D: Application Note – Avoiding Buffer Underruns by Using Modules”, page 347](#)

(1) No general statement can be made about the calculation time of a single RTC6 micro vector. This depends strongly on the set parameter values and the complexity of the **Trajectory**.

(2) For example, by **slsc_list_mark_abs**.

Module file

- Properties
 - File extension *.slm
 - Binary file
 - Cannot be opened in syncAXIS Viewer V1.5
 - Has been recorded by
slsc_list_begin_module
 - Is replayed by
slsc_list_playback_module
 - However, the following applies:

The Operation mode in which the Module has been recorded	Allowed Operation mode for replaying the Module
StageOnly	StageOnly
ScannerOnly	ScannerOnly ScannerAndStage
ScannerAndStage	ScannerOnly ScannerAndStage

- Content
 - A internal version number (compatibility information)
 - Position data per 10 μ s clock cycles (result of the **Trajectory planning**) before **Motion decomposition**⁽¹⁾, see **Figure 11**, **page 41**, along with laser switching time points (these, however, without **LaserPreTriggerTime** and **LaserSwitchOffsetTime** correction⁽²⁾)
 - At replay time, therefore, the dynamics of the recorded **Trajectory** *cannot* be changed
 - Parameter values on **Ramps**
 - Relative triggering times and parameter values (as specified by the user) of **"SIGNAL" functions**⁽³⁾
 - Various meta data, for example, on maximum speeds

- Behavior on **Module** replay

The behavior of the configuration parameter values when replaying the **Module** is shown in **Chapter 13.1 "xml-Structure Overview"**, **page 358**, that is, if/how parameter values of either

 - the **Module** or
 - the replaying **syncAXIS control instance** are applied:
 - Container

No behavior as it is a container tag
 - STANDARD

Standard behavior: The parameter value of the replaying **syncAXIS control instance** is applied. If you want to vary the parameter value, then you *do not* need to record a new **Module** each time for this. If supplementary information is available:
STANDARD***
 - NON-ST'D

Non-Standard behavior: The parameter value of the replaying **syncAXIS control instance** is *not* applied. If you want to vary the parameter value, then you need to record a new **Module** each time for this. If supplementary information is available:
NON-ST'D***

- (1) This is the set (target) **Trajectory** relative to the workpiece.
- (2) The laser switching time points are corrected by the values for **LaserPreTriggerTime** and **LaserSwitchOffsetTime** valid at replay time.
- (3) See **Chapter 2.7.2 "About the Point in Time when Output Signals are actually set"**, **page 45**.

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

// To use slsc_cfg_initialize_copy, to record and play back a Module.
// To initialize source-syncAXIS control instance in hardware mode.
size_t HardwareHandle = 0;
const char* XmlConfigFileName = "HardwareMode_syncAXISConfig.xml";
slsc_cfg_initialize_from_file(&HardwareHandle, XmlConfigFileName);

// To change configuration of the source-syncAXIS control instance (is in hardware mode).
double NewMarkSpeed = 500.0;
slsc_cfg_set_mark_speed(HardwareHandle, NewMarkSpeed);

// To initialize target-syncAXIS control instance in simulation mode with current configuration of
// source-syncAXIS control instance (hardware mode).
size_t SimulationHandle = 0;
slsc_cfg_initialize_copy(&SimulationHandle, HardwareHandle);

// To prepare Callback function to communicate end of Module recording (= end of Job planning)
// in the target-syncAXIS control instance (simulation mode).
slsc_ExecTimeCallback setModuleRecordingFinished = static_cast<slsc_ExecTimeCallback>([](size_t JobID, uint64_t,
Progress, double ExecTime, void* Context)
{
    bool* PtrToModuleRecordingFinished = static_cast<bool*>(Context);
    *PtrToModuleRecordingFinished = true;
});
bool ModuleRecordingFinished = false;
slsc_cfg_register_callback_job_end_planned(SimulationHandle, setModuleRecordingFinished,
&ModuleRecordingFinished);

// To record a Module file in the target-syncAXIS control instance (is in simulation mode).
size_t JobID_Sim = 0;
const char* ModuleFileName = "NewModule.slm";
double[2] StartPosition = {0.0, 0.0};
slsc_list_begin_module(SimulationHandle, &JobID_Sim, StartPosition, ModuleFileName);
double Target[2] = {1.0, 2.0};
slsc_list_mark_abs(SimulationHandle, Target);
slsc_list_end(SimulationHandle);

// Wait until recording has finished. Then delete target-syncAXIS control instance (is in simulation mode).
while(!ModuleRecordingFinished)
{
    std::this_thread::sleep_for(std::chrono::milliseconds(10));
}
slsc_cfg_delete(SimulationHandle);

// To replay the recorded Module file in the source-syncAXIS control instance (is in hardware mode).
size_t JobID_HW = 0;
slsc_list_begin(HardwareHandle, &JobID_HW);
slsc_list_playback_module(HardwareHandle, ModuleFileName);
slsc_list_end();

// To execute the Job. Then delete source-syncAXIS control instance (is in hardware mode).
slsc_ExecState State;
slsc_ctrl_get_exec_state(HardwareHandle, &State);
while (State != slsc_ExecState_ReadyForExecution)
{
    slsc_ctrl_get_exec_state(HardwareHandle, &State);
    std::this_thread::sleep_for(std::chrono::milliseconds(10));
}
slsc_ctrl_start_execution(HardwareHandle);
while ((State != slsc_ExecState_Idle) && (State != slsc_ExecState_NotInitOrError))
{
    slsc_ctrl_get_exec_state(HardwareHandle, &State);
    std::this_thread::sleep_for(std::chrono::milliseconds(10));
}
slsc_cfg_delete(HardwareHandle);
```

Simplified code structure for working with **Modules** – General procedure (`slsc_cfg_initialize_copy`, recording and replaying a **Module**, executing a **Job**).

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

// To record a Module.

// To initialize the source-syncAXIS control instance in simulation mode.
size_t SLHandle = 0;
const char* XmlConfigFileName = "InSimulationMode_syncAXISConfig.xml";
slsc_cfg_initialize_from_file(&SLHandle, XmlConfigFileName);

// To prepare Callback function to communicate end of Module recording (= end of Job planning).
slsc_ExecTimeCallback setModuleRecordingFinished = static_cast<slsc_ExecTimeCallback>([](size_t JobID, uint64_t
Progress, double ExecTime, void* Context)
{
    bool* PtrToModuleRecordingFinished = static_cast<bool*>(Context);
    *PtrToModuleRecordingFinished = true;
});
bool ModuleRecordingFinished = false;
slsc_cfg_register_callback_job_end_planned(SLHandle, setModuleRecordingFinished, &ModuleRecordingFinished);

// To start Module recording at start position (0,0).
size_t JobID = 0;
double StartPosition[2] = {0.0, 0.0};
const char* ModuleFileName = "NewModule.slm";
slsc_list_begin_module(SLHandle, &JobID, StartPosition, ModuleFileName);

// To define the to-be-recorded Job by a series of list commands.
// Note that this example just illustrates the mechanics of Module recording.
// It generally does not make much sense to record a Module of such a small Job.
double Target1[2] = {0.0, 5.0};
double Target2[2] = {5.0, 5.0};
uint16_t DigOutVal = 2;
double DigOutDelay = 0.0;
slsc_list_jump_abs(SLHandle, Target1);
slsc_list_write_digital_out(SLHandle, DigOutVal, DigOutDelay);
slsc_list_mark_abs(SLHandle, Target2);
slsc_list_end(SLHandle);

// To wait for Module recording to finish. Then delete syncAXIS control instance.
while(!ModuleRecordingFinished)
{
    std::this_thread::sleep_for(std::chrono::milliseconds(10));
}
slsc_cfg_delete(SLHandle);
```

Simplified code structure for recording a **Module**.

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

// To play back a Module.

// To initialize syncAXIS control instance.
size_t SLHandle = 0;
const char* XmlConfigFileName = "syncAXISConfig.xml";
slsc_cfg_initialize_from_file(&SLHandle, XmlConfigFileName);

// To define the Job in which you want to include a recorded Module. Imagine that the Module
// represents some marking pattern, that you now want to repeat a number of times on a regular grid.
// Each grid element is also to be rotated by some amount with respect to the original pattern.
size_t JobID = 0;
slsc_list_begin(SLHandle, &JobID);
int Nx = 15;
double DeltaX = 1.0;
int Ny = 10;
double DeltaY = 2.0;
const double Pi = 3.14159265359;
double DeltaAngle = 5.0 * Pi/180;
const char* ModuleFileName = "MyMarkingPattern.slm";
for (int i = 0; i < Nx; ++i)
{
    for (int j = 0; j < Ny; ++j)
    {
        double Angle = (i+j) * DeltaAngle;
        double Offset[2] = {i * DeltaX, j * DeltaY};
        slsc_list_set_rot_and_offset_2d(SLHandle, Angle, Offset);
        slsc_list_playback_module(SLHandle, ModuleFileName);
    }
}
slsc_list_end(SLHandle);

// To execute the Job. Then delete syncAXIS control instance.
slsc_ExecState State;
slsc_ctrl_get_exec_state(SLHandle, &State);
while (State != slsc_ExecState_ReadyForExecution)
{
    slsc_ctrl_get_exec_state(SLHandle, &State);
    std::this_thread::sleep_for(std::chrono::milliseconds(10));
}
slsc_ctrl_start_execution(SLHandle);
while ((State != slsc_ExecState_Idle) && (State != slsc_ExecState_NotInitOrError))
{
    slsc_ctrl_get_exec_state(SLHandle, &State);
    std::this_thread::sleep_for(std::chrono::milliseconds(10));
}
slsc_cfg_delete(SLHandle);
```

Simplified code structure for replaying a [Module](#).

2.12 About the Mode “Manual Positioning”

- To automatically put a **syncAXIS control instance** to Mode “Manual Positioning”⁽¹⁾, call:
 - **slsc_ctrl_unfollow**
- To terminate Mode “Manual Positioning”, call:
 - **slsc_ctrl_follow**
- The Mode “Manual Positioning” allows to adjust the positions of the scan device and positioning stage from the **syncAXIS control instance** as desired. As the **syncAXIS control instance** does not need to be destroyed (then positioned by an external software, for example, **ACS** software) and then recreated, there is a time saving.
- In Mode “Manual Positioning”:
 - The **Job queue** is not deleted
 - It is still possible to call Job functions (**slsc_list_***)
 - Trajectory calculations are not interrupted
 - The RTC6 remains acquired
 - The positioning stage can not only be controlled externally, but also from the **syncAXIS control instance**⁽²⁾ (by **slsc_ctrl_move_stage_abs**; the scan device also by **slsc_ctrl_move_scanner_abs**)
- However, in Mode “Manual Positioning” no **Job** starts can be triggered by **slsc_ctrl_start_execution**
- A **syncAXIS-DLL**-function can be
 - not
 - only
 - also
 permitted in Mode “Manual Positioning”, see Chapter 2.12.1 “Allowed/Not Allowed **syncAXIS control Functions**”, page 71.
- Mode “Manual Positioning” is *not* another **Operation mode** (see enum **slsc_OperationMode**). Therefore, it cannot be set in the **syncAXISConfig.xml** as the initial operating mode.

Use Case: Examination of Laser Beam Quality During Active Operation

- In Mode “Manual Positioning”, by **slsc_ctrl_move_scanner_abs** and **slsc_ctrl_move_stage_abs** (which are similar to the RTC6 command **goto_xy**), the galvanometer scanners in the scan device and the positioning stage are independently moved from each other so that the laser beam focus can hit a sensor. Then, the laser can be directly switched on/off by **slsc_ctrl_laser_signal_on** and **slsc_ctrl_laser_signal_off**.

Use Case: Temporarily Releasing the Positioning Stage and Changing the Target Positioning Stage

- See Chapter 2.12.2 “Example – Temporarily Releasing the Positioning Stage and Changing the Target Positioning Stage”, page 74.

(1) As of **syncAXIS control** \geq V1.1.0.

(2) It is therefore not necessary to use the **ACS** software to move the positioning stage to a desired position.

2.12.1 Allowed/Not Allowed syncAXIS control Functions

- If a function is not allowed in Mode “Manual Positioning”, then the return value indicates that **Bit #11** is set (**NotAllowedInCurrentMode**).

Function	Allowed in Mode “Manual Positioning”?
slsc_cfg_acquire_stage (deprecated)	No.
slsc_cfg_delete	Yes.
slsc_cfg_delete_trajectory_config	Yes.
slsc_cfg_get_calculation_dynamics_jump_scan_device	Yes.
slsc_cfg_get_calculation_dynamics_mark_scan_device	Yes.
slsc_cfg_get_calculation_dynamics_stage	Yes.
slsc_cfg_get_dynamic_limits_scan_device	Yes.
slsc_cfg_get_dynamic_limits_stage	Yes.
slsc_cfg_get_dynamic_violation_reaction	Yes.
slsc_cfg_get_field_limits_scan_device	Yes.
slsc_cfg_get_field_limits_stage	Yes.
slsc_cfg_get_jump_time	Yes.
slsc_cfg_get_mode	Yes.
slsc_cfg_get_operation_status	Yes.
slsc_cfg_get_scan_device_dynamic_monitoring_level	Yes.
slsc_cfg_get_simulation_setting	Yes.
slsc_cfg_get_stage_dynamic_monitoring_level	Yes.
slsc_cfg_get_sync_axis_version	Yes.
slsc_cfg_get_trajectory_config	Yes.
slsc_cfg_initialize_copy	No.
slsc_cfg_initialize_from_file	No.
slsc_cfg_register_callback_job_end_planned	Yes.
slsc_cfg_register_callback_job_finished_executing	Yes.
slsc_cfg_register_callback_job_is_executing	Yes.
slsc_cfg_register_callback_job_loaded_enough	Yes.
slsc_cfg_register_callback_job_progress_planned	Yes.
slsc_cfg_register_callback_job_start_planned	Yes.
slsc_cfg_reinitialize	Yes.
slsc_cfg_reinitialize_from_file	Yes.
slsc_cfg_release_stage (deprecated)	No.
slsc_cfg_select_heuristic	Yes.
slsc_cfg_select_stage	Yes.
slsc_cfg_set_bandwidth	Yes.

Function (cont'd)	Allowed in Mode "Manual Positioning"? (cont'd)
slsc_cfg_set_calculation_dynamics_jump_scan_device	Yes.
slsc_cfg_set_calculation_dynamics_mark_scan_device	Yes.
slsc_cfg_set_calculation_dynamics_stage	Yes.
slsc_cfg_set_contour_dependent_speed_control_2d	Yes.
slsc_cfg_set_dynamic_limits_scan_device	Yes.
slsc_cfg_set_dynamic_limits_stage	Yes.
slsc_cfg_set_dynamic_violation_reaction	Yes.
slsc_cfg_set_field_limits_scan_device	Yes.
slsc_cfg_set_field_limits_stage	Yes.
slsc_cfg_set_jump_speed	Yes.
slsc_cfg_set_list_handling_mode	Yes.
slsc_cfg_set_list_handling_mode_with_context	Yes.
slsc_cfg_set_mark_speed	Yes.
slsc_cfg_set_matrix_and_offset	Yes.
slsc_cfg_set_mode	No.
slsc_cfg_set_part_displacement	Yes.
slsc_cfg_set_rot_and_offset_2d	Yes.
slsc_cfg_set_scan_device_dynamic_monitoring_level	Yes.
slsc_cfg_set_simulation_setting	No.
slsc_cfg_set_stage_dynamic_monitoring_level	Yes.
slsc_cfg_set_trajectory_config	Yes.
slsc_ctrl_disable_laser	Yes.
slsc_ctrl_enable_laser	Yes.
slsc_ctrl_follow	Only allowed in Mode "Manual Positioning".
slsc_ctrl_get_error	Yes.
slsc_ctrl_get_error_count	Yes.
slsc_ctrl_get_exec_state	Yes.
slsc_ctrl_get_free_variable	Yes.
slsc_ctrl_get_job_characteristic	Yes.
slsc_ctrl_get_scan_device_position	Yes.
slsc_ctrl_get_stage_position	Yes.
slsc_ctrl_get_syncaxis_simulation_filename	Yes.
slsc_ctrl_get_value	Yes.
slsc_ctrl_is_list_input_buffer_full	Yes.
slsc_ctrl_laser_signal_off	Only allowed in Mode "Manual Positioning".
slsc_ctrl_laser_signal_on	Only allowed in Mode "Manual Positioning".

Function (cont'd)	Allowed in Mode "Manual Positioning"? (cont'd)
<code>slsc_ctrl_move_scanner_abs</code>	Only allowed in Mode "Manual Positioning".
<code>slsc_ctrl_move_stage_abs</code>	Only allowed in Mode "Manual Positioning".
<code>slsc_ctrl_refresh_correction_file</code>	Yes.
<code>slsc_ctrl_select_correction_file</code>	Yes.
<code>slsc_ctrl_set_free_variable</code>	Yes.
<code>slsc_ctrl_set_laser_pulses</code>	Yes.
<code>slsc_ctrl_start_execution</code>	No.
<code>slsc_ctrl_stop</code>	Yes.
<code>slsc_ctrl_stop_controlled</code>	No.
<code>slsc_ctrl_unfollow</code>	No.
<code>slsc_ctrl_write_analog_x</code>	Only allowed in Mode "Manual Positioning".
<code>slsc_ctrl_write_digital_out</code>	Only allowed in Mode "Manual Positioning".
<code>slsc_ctrl_write_digital_out_mask</code>	Only allowed in Mode "Manual Positioning".
<code>slsc_list_*</code> (see page 109)	Yes.

2.12.2 Example – Temporarily Releasing the Positioning Stage and Changing the Target Positioning Stage

Notice!

To be able to use **slsc_cfg_select_stage** a **Dongle** must be used which allows the use of several positioning stages (standard **Dongle** not sufficient)!













Otherwise, the return value indicates that Bit #31 is set (**InvalidOrMissingDongle**).













Notes

- For the use case “Temporarily releasing the positioning stage and changing the target-positioning stage (by **slsc_cfg_select_stage**)” the following syncAXIS-DLL functions are obsolete and are no longer to be used:
 - **slsc_cfg_release_stage** (deprecated)
 - **slsc_cfg_acquire_stage** (deprecated)

Operators of 2 positioning stage systems can combine Mode “Manual Positioning” (**slsc_ctrl_unfollow/slsc_ctrl_follow**) and **slsc_cfg_select_stage**.

As an example, the following table shows the processes in detail.

Time	 *.exe^(a) running?	 *.exe calls...	 syncAXIS control in instance existing?	 syncAXIS control in instance in Mode “Manual Positioning”?	 syncAXIS control in instance with queue?	 syncAXIS control in instance calculating for Job 1?	 syncAXIS control in instance calculating for Job 2?	 syncAXIS control in instance executes Job 1?	 syncAXIS control in instance executes Job 2?	 RTCC6 acquired?^(b)	 Positioning stage “1” acquired?^(b)	 Positioning stage “2” acquired?^(b)
0	No.	–	–	–	–	–	–	–	–	No.	No.	No. ^{(c)(d)}
1	Yes.	–	No.	–	–	–	–	–	–	No.	No.	No.
2	Yes.	slsc_cfg_initialize_from_file^(e)	No.	–	–	–	–	–	–	No.	No.	No.
3	Yes.	–	Yes.	No.	Yes.	–	–	–	–	Yes.	Yes.	No.
4	Yes.	slsc_list_begin	Yes.	No.	Yes.	No.	No.	No.	No.	Yes.	Yes.	No.
5	Yes.	slsc_list_*	Yes.	No.	Yes.	Yes.	No.	No.	No.	Yes.	Yes.	No.
6	Yes.	slsc_list_**	Yes.	No.	Yes.	Yes.	No.	No.	No.	Yes.	Yes.	No.
7	Yes.	slsc_ctrl_get_exec_state^(f)	Yes.	No.	Yes.	Yes.	No.	No.	No.	Yes.	Yes.	No.
8	Yes.	slsc_ctrl_start_execution	Yes.	No.	Yes.	Yes.	No.	No.	No.	Yes.	Yes.	No.
9	Yes.	–	Yes.	No.	Yes.	Yes.	No.	Yes.	No.	Yes.	Yes.	No.
10	Yes.	slsc_list_end	Yes.	No.	Yes.	Yes.	No.	Yes.	No.	Yes.	Yes.	No.
11	Yes.	–	Yes.	No.	Yes.	Yes.	No.	Yes.	No.	Yes.	Yes.	No.
12	Yes.	slsc_list_begin	Yes.	No.	Yes.	Yes.	No.	Yes.	No.	Yes.	Yes.	No.
13	Yes.	slsc_list_*	Yes.	No.	Yes.	Yes.	No.	Yes.	No.	Yes.	Yes.	No.
14	Yes.	slsc_list_**	Yes.	No.	Yes.	Yes.	No.	Yes.	No.	Yes.	Yes.	No.
15	Yes.	–	Yes.	No.	Yes.	Yes.	No.	Yes.	No.	Yes.	Yes.	No.
16	Yes.	–	Yes.	No.	Yes.	No. ^(g)	Yes.	Yes.	No.	Yes.	Yes.	No.
17	Yes.	–	Yes.	No.	Yes.	No.	Yes.	Yes.	No.	Yes.	Yes.	No.
18	Yes.	slsc_ctrl_get_exec_state	Yes.	No.	Yes.	No.	Yes.	No. ^(h)	No.	Yes.	Yes.	No.
19	Yes.	slsc_ctrl_get_exec_state^(f)	Yes.	No.	Yes.	No.	Yes.	No.	No.	Yes.	Yes.	No.

Time	 * .exe ^(a) running?	 * .exe calls...	 syncAXIS control instance existing?	 syncAXIS control instance in Mode "Manual Positioning"?	 syncAXIS control instance with queue?	 syncAXIS control instance calculating for Job 1?	 syncAXIS control instance calculating for Job 2?	 syncAXIS control instance executes Job 1?	 syncAXIS control instance executes Job 2?	 RTC6 acquired? ^(b)	 Positioning stage "1" acquired? ^(b)	 Positioning stage "2" acquired? ^(b)
20	Yes.	–	Yes.	No.	Yes.	No.	Yes.	No.	No.	Yes.	Yes.	No.
21	Yes.	slsc_ctrl_unfollow	Yes.	No.	Yes.	No.	Yes.	No.	No.	Yes.	Yes.	No.
22	Yes.	–	Yes.	Yes.	Yes.	No.	Yes.	No.	No.	Yes.	No. ^(d)	No.
23	Yes.	slsc_ctrl_move_scanner_abs	Yes.	Yes.	Yes.	No.	Yes.	No.	No.	Yes.	No. ^(d)	No.
24	Yes.	slsc_ctrl_move_stage_abs ⁽ⁱ⁾	Yes.	Yes.	Yes.	No.	Yes.	No.	No.	Yes.	No. ^(d)	No.
25	Yes.	–	Yes.	Yes.	Yes.	No.	Yes.	No.	No.	Yes.	No. ^(d)	No.
26	Yes.	slsc_cfg_select_stage(Stage = 2) ^(j)	Yes.	Yes.	Yes.	No.	Yes.	No.	No.	Yes.	No. ^(d)	No.
27	Yes.	–	Yes.	Yes.	Yes.	No.	Yes.	No.	No.	Yes.	No.	No. ^(d)
28	Yes.	slsc_ctrl_move_scanner_abs	Yes.	Yes.	Yes.	No.	Yes.	No.	No.	Yes.	No.	No. ^(d)
29	Yes.	slsc_ctrl_move_stage_abs ^(k)	Yes.	Yes.	Yes.	No.	Yes.	No.	No.	Yes.	No.	No. ^(d)
30	Yes.	–	Yes.	Yes.	Yes.	No.	Yes.	No.	No.	Yes.	No.	No. ^(d)
31	Yes.	slsc_ctrl_follow	Yes.	Yes.	Yes.	No.	Yes.	No.	No.	Yes.	No.	No. ^(d)
32	Yes.	–	Yes.	No.	Yes.	No.	Yes.	No.	No.	Yes.	No.	Yes.
33	Yes.	slsc_ctrl_start_execution	Yes.	No.	Yes.	No.	Yes.	No.	No.	Yes.	No.	Yes.
34	Yes.	–	Yes.	No.	Yes.	No.	Yes.	No.	Yes.	Yes.	No.	Yes.
35	Yes.	slsc_list_end	Yes.	No.	Yes.	No.	Yes.	No.	Yes.	Yes.	No.	Yes.
36	Yes.	–	Yes.	No.	Yes.	No.	Yes.	No.	Yes.	Yes.	No.	Yes.
37	Yes.	–	Yes.	No.	Yes.	No.	Yes.	No.	Yes.	Yes.	No.	Yes.
38	Yes.	–	Yes.	No.	Yes.	No.	No. ^(l)	No.	Yes.	Yes.	No.	Yes.
39	Yes.	–	Yes.	No.	Yes.	No.	No.	No.	Yes.	Yes.	No.	Yes.
40	Yes.	–	Yes.	No.	Yes.	No.	No.	No.	Yes.	Yes.	No.	Yes.
41	Yes.	slsc_ctrl_get_exec_state	Yes.	No.	Yes.	No.	No.	No.	No. ^(m)	Yes.	No.	Yes.
42	Yes.	–	Yes.	No.	Yes.	No.	No.	No.	No.	Yes.	No.	Yes.
43	Yes.	slsc_cfg_delete	Yes.	No.	Yes.	No.	No.	No.	No.	Yes.	No.	Yes.
44	Yes.	–	No.	No.	No.	No.	No.	No.	No.	No.	No.	No.
45	No.	–	No.	No.	No.	No.	No.	No.	No.	No.	No.	No.

(a) syncAXIS-DLL-based user program, for example, a GUI.

(b) By the syncAXIS control instance.

(c) "2" positioning stage can be acquired by "another" user program.

(d) This positioning stage can be positioned by the syncAXIS-DLL. Here, an ACS point-to-point motion command is sent to ACS Motion Controller via TCP/IP. This can be used for preparations for subsequent workpiece processing, for example, setting up, moving, workpiece feeding, imaging, measuring, positioning.


(e) It is assumed, that positioning stage 1 and Operation mode "ScannerAndStage" are entered in syncAXISConfig.xml.

(f) slsc_ExecState_ReadyForExecution.

(g) Event job_end_planned, see Figure 12.

(h) Event job_finished_executing, see Figure 12.

(i) Sends an ACS point-to-point motion command for positioning stage 1 to ACS Motion Controller via TCP/IP.

(j)  Requires a correspondingly configured Dongle!

(k) Sends an ACS point-to-point motion command for positioning stage 1 to ACS Motion Controller via TCP/IP.

(l) Event job_end_planned, see Figure 12.

(m) Event job_finished_executing, see Figure 12.

3 Functions Available in the API

3.1 Functional Overview

In this chapter:

- [Configuration Functions \(slsc_cfg_*\), page 76](#)
- [Job Functions \(slsc_list_*\), page 85](#)
- [Control Functions \(slsc_ctrl_*\), page 96](#)
- [Utility Functions \(slsc_util_*\), page 101](#)

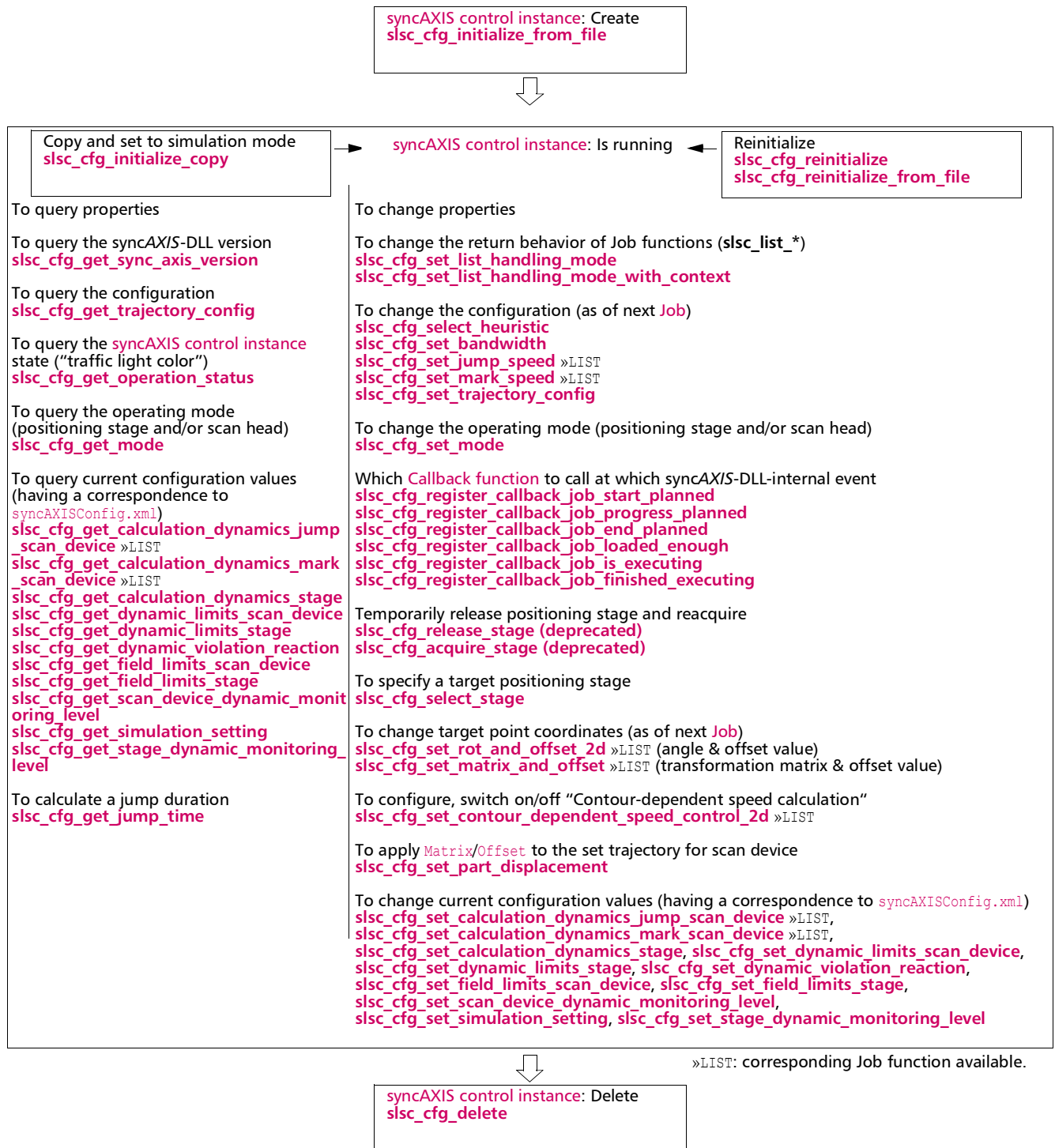
3.1.1 Configuration Functions (slsc_cfg_*)

Configuration function (prefix `slsc_cfg_`) allow creating/deleting/reinitializing `syncAXIS control instances` and querying and setting their properties ("configuration").

Furthermore, the positioning stage can be temporarily released (for example, to be controlled "externally") and subsequently acquired back on again. Optionally, during the temporal release, a different target positioning stage can be specified (which is then going to be acquired subsequently). That way it is possible to work alternately with several positioning stages. For further information, see [Chapter 2.12.2 "Example – Temporarily Releasing the Positioning Stage and Changing the Target Positioning Stage", page 74](#).

For a graphical overview, see [Figure 30, page 77](#).

Configuration Functions (slsc_cfg_*)



syncAXIS control instance-related Functions

- **slsc_cfg_acquire_stage (deprecated)** outdated
- **slsc_cfg_delete**
- **slsc_cfg_get_calculation_dynamics_jump_scan_device**
- **slsc_cfg_get_calculation_dynamics_mark_scan_device**
- **slsc_cfg_get_calculation_dynamics_stage**
- **slsc_cfg_get_dynamic_limits_scan_device**
- **slsc_cfg_get_dynamic_limits_stage**
- **slsc_cfg_get_dynamic_violation_reaction**
- **slsc_cfg_get_field_limits_scan_device**
- **slsc_cfg_get_field_limits_stage**
- **slsc_cfg_get_jump_time**
- **slsc_cfg_get_mode**
- **slsc_cfg_get_operation_status**
- **slsc_cfg_get_scan_device_dynamic_monitoring_level**
- **slsc_cfg_get_stage_dynamic_monitoring_level**
- **slsc_cfg_get_sync_axis_version**
- **slsc_cfg_get_trajectory_config**
- **slsc_cfg_initialize_copy**
- **slsc_cfg_initialize_from_file**
- **slsc_cfg_reinitialize**
- **slsc_cfg_reinitialize_from_file**
- **slsc_cfg_release_stage (deprecated)** outdated
- **slsc_cfg_select_stage**
- **slsc_cfg_set_list_handling_mode**
- **slsc_cfg_set_list_handling_mode_with_context**
- **slsc_cfg_set_matrix_and_offset**
- **slsc_cfg_set_mode**
- **slsc_cfg_set_trajectory_config**



Caution!

Make sure that laser safety is ensured in the entire system. In the safety concept of your system control, take into account that the RTC laser control signals are enabled by **slsc_cfg_initialize_from_file** and **slsc_ctrl_enable_laser**.

In order to make use of the syncAXIS-DLL, a function to create a **syncAXIS control instance** must be present at the beginning of the user program. This is done with **slsc_cfg_initialize_from_file** by specifying a valid XML configuration file⁽¹⁾⁽²⁾.

In the process, a unique **Handle** is assigned to the **syncAXIS control instance** through which it can be addressed (almost all functions require a **Handle** value to be specified). Furthermore, the hardware is acquired (RTC6 board and positioning stage). For further information see **Chapter 2.4 "About Initializing syncAXIS control-based User Programs"**, page 26.

A **syncAXIS control instance** can either be reinitialized by **slsc_cfg_reinitialize_from_file** (based on the **syncAXISConfig.xml**) or **slsc_cfg_reinitialize** (based on the current configuration state). Thereby the **Handle** value remains unchanged.

The current status ("traffic light color") of a **syncAXIS control instance** is queried by **slsc_cfg_get_operation_status**.

In order to query the **Trajectory planning** configuration values, **slsc_cfg_get_trajectory_config** is available (to change these, see **Section "Functions for Changing the Configuration of the Present syncAXIS-DLL Instance"**, page 80).

(1) XSD file (XML Schema Definition; **syncAXIS_1_8.xsd**) and XML configuration file templates are part of the delivery.

(2) In **syncAXISConfig.xml**, a number of **syncAXIS control instance** configuration parameters (suitable for the respective system or application) are to be specified. You can find a description of these parameters as inline comments in the **syncAXIS_1_8.xsd** as well as in **Chapter 13 "Appendix F: Reference of syncAXISConfig.xml Tags"**, page 358. Some of the parameter values can be changed by syncAXIS-DLL functions in the user program at runtime.

To query current values of the **syncAXIS control instance** for which there are equivalents/decided tags in the **syncAXISConfig.xml**, are available:

- **slsc_cfg_get_calculation_dynamics_jump_scan_device**
- **slsc_cfg_get_calculation_dynamics_mark_scan_device**
- **slsc_cfg_get_calculation_dynamics_stage**
- **slsc_cfg_get_dynamic_limits_scan_device**
- **slsc_cfg_get_dynamic_limits_stage**
- **slsc_cfg_get_dynamic_violation_reaction**
- **slsc_cfg_get_field_limits_scan_device**
- **slsc_cfg_get_field_limits_stage**
- **slsc_cfg_get_scan_device_dynamic_monitoring_level**
- **slsc_cfg_get_stage_dynamic_monitoring_level**

The **syncAXIS-DLL** version can be queried by **slsc_cfg_get_sync_axis_version**.

slsc_cfg_set_mode sets the **Operation mode** ("StageOnly"/"ScannerOnly"/"ScannerAndStage"; see enum **slsc_OperationMode**) the **syncAXIS control instance** shall work. *In the process, the syncAXIS control instance is reinitialized!*

slsc_cfg_initialize_copy puts a copy of a **syncAXIS control instance** to simulation mode without changing its configuration.

slsc_cfg_initialize_copy can be used, for example, in the context of recording **Module files**. For this, **slsc_cfg_initialize_copy** is called prior to **slsc_list_begin_module** (this function requires the simulation mode), see also **Section "Functions for "Modules""**, page 95.

The currently set **Operation mode** of the **syncAXIS control instance** is queried by **slsc_cfg_get_mode**.

For **slsc_cfg_select_stage**, see **Chapter 2.12.2 "Example – Temporarily Releasing the Positioning Stage and Changing the Target Positioning Stage"**, page 74. Note that **slsc_cfg_release_stage** (deprecated)/**slsc_cfg_acquire_stage** (deprecated) are outdated and should no longer be used for this use case.

By **slsc_cfg_get_simulation_setting**, the current **Simulation Setting** can be queried, see also **Chapter 2.5 "About the syncAXIS control Simulation Mode"**, page 31. For changing, **slsc_cfg_set_simulation_setting** is available. *In the process, the syncAXIS control instance is reinitialized!*

By **slsc_cfg_set_list_handling_mode** the return behavior of the Job functions (**slsc_list_***) can be modified. With **slsc_cfg_set_list_handling_mode_with_context**, a context can be specified in addition.

To be able to change target point coordinates (see page 268) of **slsc_list_arc_abs**, **slsc_list_circle_2d_abs**, **slsc_list_jump_abs**, **slsc_list_mark_abs** and their corresponding **slsc_list_[para/multi_para]*** functions for a **syncAXIS control instance**, **slsc_cfg_set_rot_and_offset_2d** and **slsc_cfg_set_matrix_and_offset** are available. With **slsc_cfg_set_rot_and_offset_2d** an angle and offset value can be specified, with **slsc_cfg_set_matrix_and_offset** a transformation matrix and an offset value.

For both there are corresponding Job functions (**slsc_list_***) which are **slsc_list_set_rot_and_offset_2d** and **slsc_list_set_matrix_and_offset**, see also **Section "Functions for Changing Target Point Coordinates"**, page 92.

The "Contour-dependent speed calculation" can be configured as well as switched on and off by **slsc_cfg_set_contour_dependent_speed_control_2d**. For prerequisites and further information, see **Chapter 2.9.5 "About the "Contour-dependent speed calculation""**, page 60. For **slsc_cfg_set_contour_dependent_speed_control_2d** there is the corresponding Job function (**slsc_list_***) **slsc_list_set_contour_dependent_speed_control_2d**.

By **slsc_cfg_get_jump_time** a **Job** can be analyzed and optimized - without having to simulate it as a whole. See also **Chapter 2.2.4 "Simulating and Improving Jobs"**, page 24.

At the end of the user program, **slsc_cfg_delete** must be inserted in order to erase the **syncAXIS control instance** and to release the hardware.

Functions for Changing the Configuration of the Present syncAXIS-DLL Instance

- `slsc_cfg_select_heuristic`
- `slsc_cfg_set_bandwidth`
- `slsc_cfg_set_calculation_dynamics_jump_scan_device`
- `slsc_cfg_set_calculation_dynamics_mark_scan_device`
- `slsc_cfg_set_calculation_dynamics_stage`
- `slsc_cfg_set_dynamic_limits_scan_device`
- `slsc_cfg_set_dynamic_limits_stage`
- `slsc_cfg_set_dynamic_violation_reaction`
- `slsc_cfg_set_field_limits_scan_device`
- `slsc_cfg_set_field_limits_stage`
- `slsc_cfg_set_jump_speed`
- `slsc_cfg_set_mark_speed`
- `slsc_cfg_set_part_displacement`
- `slsc_cfg_set_scan_device_dynamic_monitoring_level`
- `slsc_cfg_set_stage_dynamic_monitoring_level`
- `slsc_cfg_set_trajectory_config` (with `slsc_cfg_delete_trajectory_config`)

These functions have in common:

- There is *no reinitialization* of the syncAXIS control instance.
- The specified configuration changes become effective only after `slsc_list_begin*`. These configuration changes are valid as of the next **Job** for all **Jobs** that follow.

Notes

- For `slsc_cfg_set_jump_speed` there is the corresponding Job function (`slsc_list_*`) `slsc_list_set_jump_speed`.
- For `slsc_cfg_set_mark_speed` there is the corresponding Job function (`slsc_list_*`) `slsc_list_set_mark_speed`.
- `slsc_cfg_set_trajectory_config` sets the Trajectory planning configuration values for the specified syncAXIS control instance.
- `slsc_cfg_delete_trajectory_config` is an auxiliary function for software development. With it, the trajectory configuration object (which is created by `slsc_cfg_set_trajectory_config`) can be deleted again once it is no longer needed (in order to avoid memory leaks). `slsc_cfg_delete_trajectory_config` does not change any configuration parameter values.
- For further information on `slsc_cfg_set_part_displacement`, see Chapter 8.3 "About Transformations in syncAXIS control V1.2.4 and Higher", page 332.
- Different **Jobs** each have different "optimal" `FilterBandwidth` values. Besides the possibility to change the `FilterBandwidth` value in the user program, `slsc_cfg_set_bandwidth` is also provided to be able to implement optimization algorithms.
- `slsc_cfg_select_heuristic` sets the desired characteristic for the speed reduction (`DynamicReductionFunction`).
- To change current values of the syncAXIS control instance for which there are equivalents/decided tags in the `syncAXISConfig.xml`, are available:
 - `slsc_cfg_set_calculation_dynamics_jump_scan_device`
 - `slsc_cfg_set_calculation_dynamics_mark_scan_device`
 - `slsc_cfg_set_calculation_dynamics_stage`
 - `slsc_cfg_set_dynamic_limits_scan_device`
 - `slsc_cfg_set_dynamic_limits_stage`
 - `slsc_cfg_set_dynamic_violation_reaction`
 - `slsc_cfg_set_field_limits_scan_device`
 - `slsc_cfg_set_field_limits_stage`
 - `slsc_cfg_set_scan_device_dynamic_monitoring_level`
 - `slsc_cfg_set_stage_dynamic_monitoring_level`

Functions for Registering “Callback Events”

- `slsc_cfg_register_callback_job_start_planned`
- `slsc_cfg_register_callback_job_progress_planned`
- `slsc_cfg_register_callback_job_end_planned`
- `slsc_cfg_register_callback_job_finished_executing`
- `slsc_cfg_register_callback_job_is_executing`
- `slsc_cfg_register_callback_job_loaded_enough`

As shown in Figure 12, page 43, and following Table, different “Callback event” occur `syncAXIS control instance`-internally several times for each Job. A corresponding Function for registering “Callback events” is available for each of these “Callback events”, for example, `slsc_cfg_register_callback_job_end_planned` for “`job_finished_executing`”.

These functions allow to configure the `syncAXIS control instance` so that the specified user-defined function (= “Callback function”) is called as soon as the associated “Callback event” occurs.

Each “Callback function” must meet the predefined convention (return types, calling conventions, arguments), see the both `Callback function` signatures `slsc_ExecTimeCallback` and `slsc_JobCallback`.

Name of the “Callback event”	Frequency per Job	Corresponding “Function for registering “Callback events””	Mandatory signature for the “Callback function”
<code>job_start_planned</code>	1 ×	<code>slsc_cfg_register_callback_job_start_planned</code>	<code>slsc_JobCallback</code>
<code>job_progress_planned</code>	May occur several times ^(a)	<code>slsc_cfg_register_callback_job_progress_planned</code>	<code>slsc_ExecTimeCallback</code>
<code>job_end_planned</code>	1 ×	<code>slsc_cfg_register_callback_job_end_planned</code>	<code>slsc_ExecTimeCallback</code>
<code>job_loaded_enough</code>	1 ×	<code>slsc_cfg_register_callback_job_loaded_enough</code>	<code>slsc_JobCallback</code>
<code>job_is_executing</code>	May occur several times	<code>slsc_cfg_register_callback_job_is_executing</code>	<code>slsc_ExecTimeCallback</code>
<code>job_finished_executing</code>	1 ×	<code>slsc_cfg_register_callback_job_finished_executing</code>	<code>slsc_ExecTimeCallback</code>

(a) With very short Jobs: even possibly not at all.

Notes

- Simple usage examples are:
 - External (= by a GUI) monitoring of the **Job** progress
 - A “red traffic light” in a GUI as long as a **Job** is executed
 - A GUI message once a **Job** is finished
 - Code examples:
 - Exemplary implementation using function pointers, see [Figure 31, page 83](#).
 - Exemplary implementation using C++11 lambda functions (anonymous functions), see [Figure 32, page 84](#).
 - The “Callback function” specified with **slsc_cfg_register_callback_job_start_planned** is executed once the **Trajectory planning** has been started for the **Job**. This means **slsc_list_begin[*]** has been called and processed and the first Job function (like **slsc_list_jump_abs** etc.) has been called and successfully processed through the validation check. **Buffers** have to be filled sufficiently for the **Trajectory planning** to start and the motion for scan head and positioning stage can be decomposed.
 - The “Callback function” specified in **slsc_cfg_register_callback_job_progress_planned** is executed repeatedly while the **Job** planning is in progress. This means the Callback function specified in **slsc_cfg_register_callback_job_start_planned** already has been executed. With very short **Jobs**, the “Callback event” of type “**job_progress_planned**” might not occur at all.
 - The “Callback function” specified in **slsc_cfg_register_callback_job_loaded_enough** is executed once the **Job** planning processed enough so the **Job** could be started. This means that the information transfer to the RTC6 board has been started and some RTC6 commands have been written to the RTC6 list memory.
- Around this moment, the execution state (**slsc_ExecState**) of the **Job** also transverses to **slsc_ExecState_ReadyForExecution** (not necessarily exactly the same moment due to internal information polling. To check if the **Job** execution can be started, SCANLAB recommends not to rely on the “Callback function” but to use **slsc_ctrl_get_exec_state** instead. From this moment on, the **Job** execution can be started by **slsc_ctrl_start_execution**.
- The “Callback function” specified in **slsc_cfg_register_callback_job_end_planned** is executed after the **Job** planning has been finished. This means that all the calculation has been done and the calculation of the next **Job** will start, if another **Job** planning has been started by the user. syncAXIS control will continue transferring the remaining information to the RTC6 board. From this moment on, users can query the **Job** characteristics (“key”) by **slsc_ctrl_get_job_characteristic**, if desired.
 - The “Callback function” specified in **slsc_cfg_register_callback_job_is_executing** is executed repeatedly while the **Job** execution is in progress. This means the “**job_loaded_enough**” “Callback event” has already occurred and the **Job** execution has actually been started by **slsc_ctrl_start_execution**. At this moment, the execution state (**slsc_ExecState**) of the **Job** transverses to **slsc_ExecState_Executing**. With very short **Jobs**, the “Callback event” of type “**job_is_executing**” might not occur at all.
 - The “Callback function” specified in **slsc_cfg_register_callback_job_finished_executing** is executed once the **Job** execution finished. This means that the scan head and positioning stage reached their final position and stopped their movement. At this moment, the execution state (**slsc_ExecState**) of the **Job** transverses to **slsc_ExecState_Idle**. No more “Callback event” regarding this **Job** will occur.

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.
// Functions and concepts are utilized that are declared and defined in the "Installation_Project"
// of the syncAXIS control-software package, see there.

struct Content
{
    int* Counter;
    std::string Action;
};

void execCallbackFunction(size_t JobID, uint64_t Progress, double ExecTime, void* Context)
{
    Content* CallbackContent = static_cast<Content*>(Context);
    std::cout << "Step number " << (*CallbackContent->Counter)++ << " " << CallbackContent->Action << ". Execution up to now took: "
    << ExecTime << std::endl; return;
};

void jobCallbackFunction(size_t JobID, void* Context)
{
    Content* CallbackContent = static_cast<Content*>(Context);
    std::cout << "Step number " << (*CallbackContent->Counter)++ << " " << CallbackContent->Action << "." << std::endl;
    return;
};

size_t SLHandle = 0;
// Initialize Instance, e.g.
slsc_cfg_initialize_from_file(&SLHandle, "syncAXISConfig.xml");

int ProcessCounter = 0;

Content ContentIsPlanning = { &ProcessCounter, "is planning" };
Content ContentIsExecuting = { &ProcessCounter, "is executing" };
Content ContentPlanningStarted = { &ProcessCounter, "started the planning" };
Content ContentJobFinished = { &ProcessCounter, "finished the execution" };
Content ContentPlanningFinished = { &ProcessCounter, "finished the planning" };
Content ContentJobReady = { &ProcessCounter, "loaded enough information in the buffers to start the execution." };

slsc_cfg_register_callback_job_finished_executing(SLHandle, static_cast<slsc_ExecTimeCallback>(&execCallbackFunction), &ContentJobFinished);
slsc_cfg_register_callback_job_end_planned(SLHandle, static_cast<slsc_ExecTimeCallback>(&execCallbackFunction), &ContentPlanningFinished);
slsc_cfg_register_callback_job_loaded_enough(SLHandle, static_cast<slsc_JobCallback>(&jobCallbackFunction), &ContentJobReady);
slsc_cfg_register_callback_job_is_executing(SLHandle, static_cast<slsc_ExecTimeCallback>(&execCallbackFunction), &ContentIsExecuting);
slsc_cfg_register_callback_job_progress_planned(SLHandle, static_cast<slsc_ExecTimeCallback>(&execCallbackFunction), &ContentIsPlanning);
slsc_cfg_register_callback_job_start_planned(SLHandle, static_cast<slsc_JobCallback>(&jobCallbackFunction), &ContentPlanningStarted);

// Execute random job, e.g. (from Installation Project)
{
    double ScalingFactor = 10;
    auto ListFilling = std::async(std::launch::async, [&]()
    {
        return writeMarkingPattern1(SLHandle, RetVal, ScalingFactor, Transformation, Offset);
    });
}

startJob(SLHandle, RetVal, true);

// Delete Instance, e.g.
slsc_cfg_delete(SLHandle);
```

Code example: implementation of the functions for registering "Callback events" (see [page 12](#)) and "Callback functions". Other than in [Figure 32, page 84](#), here, function pointers are used.

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.
// Functions and concepts are utilized that are declared and defined in the "Installation_Project"
// of the syncAXIS control-software package, see there.

struct Content
{
    int* Counter;
    std::string Action;
};

size_t SLHandle = 0;
//Initialize Instance, e.g.
slsc_cfg_initialize_from_file(&SLHandle, "syncAXISConfig.xml");

int ProcessCounter = 0;

Content ContentIsPlanning= { &ProcessCounter, "is planning" };
Content ContentIsExecuting = { &ProcessCounter, "is executing" };
Content ContentPlanningStarted = { &ProcessCounter, "started the planning" };
Content ContentJobFinished = { &ProcessCounter, "finished the execution" };
Content ContentPlanningFinished = { &ProcessCounter, "finished the planning" };
Content ContentJobReady = { &ProcessCounter, "loaded enough information in the buffers to start the execution." };

slsc_ExecTimeCallback ExecCallback = [](size_t JobID, uint64_t Progress, double ExecTime, void* Context)
{
    Content* CallbackContent = static_cast<Content*>(Context);
    std::cout << "Step number " << (*CallbackContent->Counter)++ << " " << CallbackContent->Action << ". Execution up to now took: "
    << ExecTime << std::endl; return;
};

slsc_JobCallback JobCallback = [](size_t JobID, void* Context)
{
    Content* CallbackContent = static_cast<Content*>(Context);
    std::cout << "Step number " << (*CallbackContent->Counter)++ << " " << CallbackContent->Action << "." << std::endl;
    return;
};

slsc_cfg_register_callback_job_finished_executing(SLHandle, ExecCallback, &ContentJobFinished);
slsc_cfg_register_callback_job_end_planned(SLHandle, ExecCallback, &ContentPlanningFinished);
slsc_cfg_register_callback_job_loaded_enough(SLHandle, JobCallback, &ContentJobReady);
slsc_cfg_register_callback_job_is_executing(SLHandle, ExecCallback, &ContentIsExecuting);
slsc_cfg_register_callback_job_progress_planned(SLHandle, ExecCallback, &ContentIsPlanning);
slsc_cfg_register_callback_job_start_planned(SLHandle, JobCallback, &ContentPlanningStarted);

// execute random job, e.g. (from Installation Project)
{
    double ScalingFactor = 10;
    auto ListFilling = std::async(std::launch::async, [&]()
    {
        return writeMarkingPattern1(SLHandle, RetVal, ScalingFactor, Transformation, Offset);
    });
}

startJob(SLHandle, RetVal, true);

// Delete Instance, e.g.
slsc_cfg_delete(SLHandle);
```

Code example: implementation of the functions for registering "[Callback events](#)" (see [page 12](#)) and "[Callback functions](#)". Other than in [Figure 31](#), [page 83](#), here, C++ 11 Lambda functions (Anonymous functions) are used.

3.1.2 Job Functions (**slsc_list_***)

Job functions (prefix **slsc_list_**) allow defining **Jobs**, that is, they are the individual elements of **Jobs**. Particularly important are the functions for markings and jumps, as well as for switching signals at output ports (apart from the mandatory functions for the begin and end of **Jobs**).

For a graphical overview, see [Figure 33, page 86](#).

Notes

- The following applies to all Job functions (**slsc_list_***): upon submission from the user program to the **syncAXIS control instance**, a consistency check is carried out. In case of an error, the return value indicates that **Bit #07** is set (**JobStructureNotValid**).
- See also [Section "Structure to Comply with when Defining Jobs", page 25](#).
- Job functions (prefix **slsc_list_**) are technically totally different from "RTC list commands (suffix **_list**)" which are described in the [RTC6 Manual](#). Therefore, they are not designated as such in this document. Among other things, the execution times of Job functions are not exactly predictable from a user's point of view (unlike RTC list commands). Furthermore, there are Job functions which generate several RTC6 list commands for the RTC6 board.
- The very last Job function in a **Job** must be **slsc_list_end**. It calculates a completion for the **Trajectory**. After that (as with **slsc_cfg_initialize_from_file** and **slsc_cfg_reinitialize_from_file**)
 - the scan head mirrors are in the zero position and
 - the positioning stage (not relevant with **Operation mode "ScannerOnly"**) remains at the last set jump position or marking position.

Functions for Defining Job-Beginnings/Ends

- **slsc_list_begin**
- **slsc_list_begin_absolute**
- **slsc_list_begin_relative**
- **slsc_list_end**

The first function of a **Job** must be **slsc_list_begin**, (alternatively **slsc_list_begin_absolute** or **slsc_list_begin_relative**, see their reference tables). Otherwise, the return value indicates that **Bit #07** is set (**JobStructureNotValid**).

None of the functions **slsc_list_begin**, **slsc_list_begin_absolute** and **slsc_list_begin_relative** must be followed by **slsc_list_begin**, **slsc_list_begin_absolute** or **slsc_list_begin_relative**.

Otherwise, the return value indicates that **Bit #07** is set (**JobStructureNotValid**).

slsc_list_end must be the last Job function in a **Job**, see list bullet on the left.

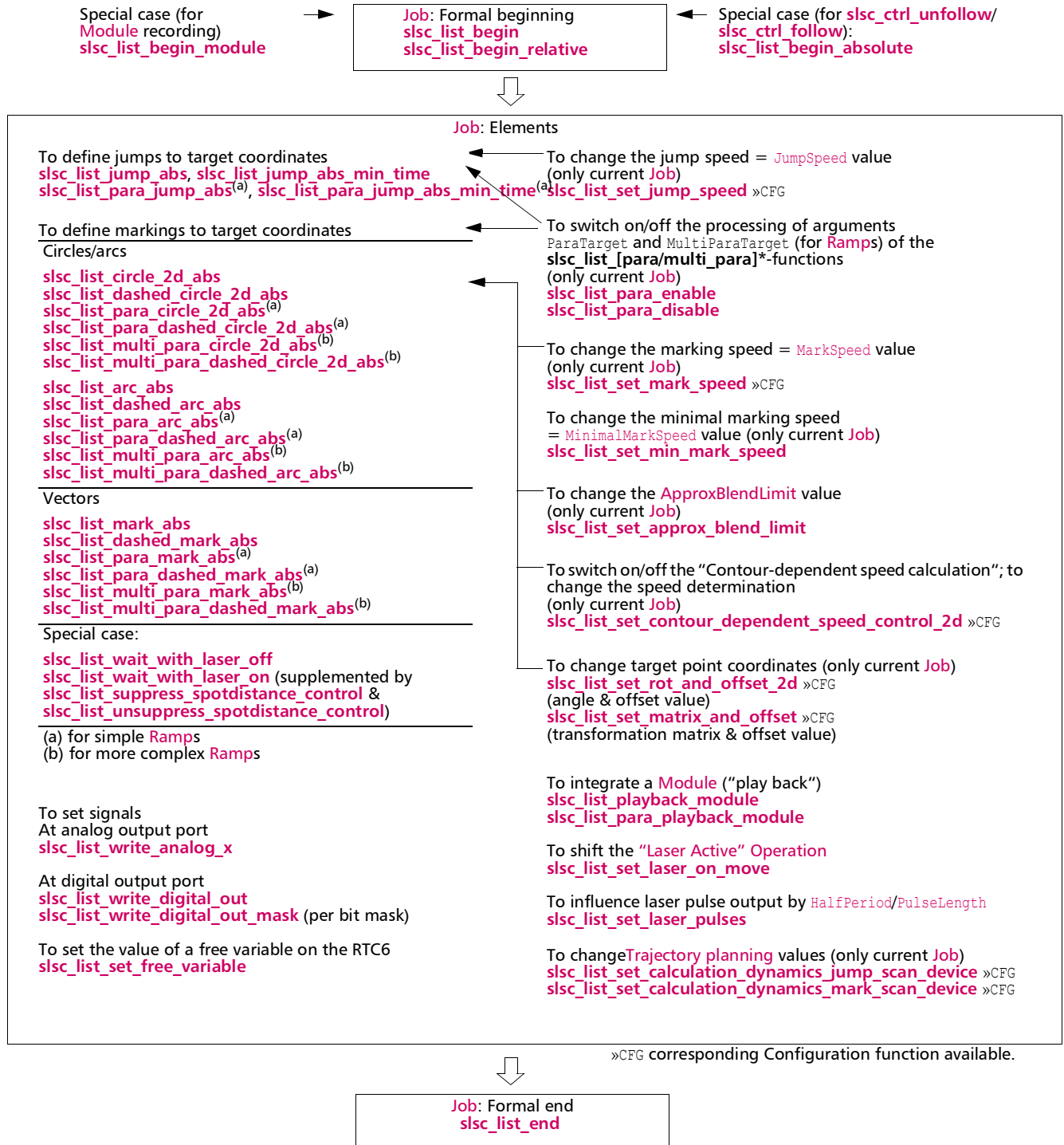
Functions for Defining Jumps

- **slsc_list_jump_abs**
- **slsc_list_jump_abs_min_time**

By **slsc_list_jump_abs_min_time** a jump can be defined – as by **slsc_list_jump_abs** – but additionally allows to specify its minimum duration.

The corresponding **slsc_list_para*** function of **slsc_list_jump_abs** is **slsc_list_para_jump_abs**, that of **slsc_list_jump_abs_min_time** is **slsc_list_para_jump_abs_min_time**.

Job Functions (slsc_list_*)



Functions for Defining Markings

- **slsc_list_arc_abs**
- **slsc_list_circle_2d_abs**
- **slsc_list_mark_abs**
- **slsc_list_wait_with_laser_off**
- **slsc_list_wait_with_laser_on**
(with supplementing **slsc_list_suppress_spotdistance_control** and **slsc_list_unsuppress_spotdistance_control**)
- **slsc_list_set_laser_on_move**

slsc_list_wait_with_laser_on behaves like a **slsc_list_mark_abs** with velocity 0 (that is, for a specified time the mirrors remain in their last position while the laser is switched on) and can be used, for example, if the quality of the laser spot is to be measured by external sensors.

slsc_list_wait_with_laser_on is supplemented by **slsc_list_wait_with_laser_off**. The only difference between the two functions is that the laser is switched off with **slsc_list_wait_with_laser_off**.

Special Case: **SpotDistance** as an “ActiveChannel”

Only Sky Writings who (in **syncAXISConfig.xml**) have set **SpotDistance** as “ActiveChannel”, see [Chapter 2.9.2 “Definition of the Channels and ActiveChannel”](#), page 48, must call

slsc_list_suppress_spotdistance_control prior to **slsc_list_wait_with_laser_on**. Otherwise, no pulses are initiated with triggerable lasers.

Background: With **SpotDistance** as temporal pulse spacing (that is, the half-period for the LASER1 signals and LASER2 signals) is adjusted during **Job** execution so that the spot spacing is ultimately equidistant. However, the syncAXIS-DLL-internal calculation for this is based on the current marking speed. The lower it is, the larger is the pulse distance. Finally, in the border case “marking speed = 0” (which is the case with **slsc_list_wait_with_laser_on**, see above), no more pulse is triggered from the laser.

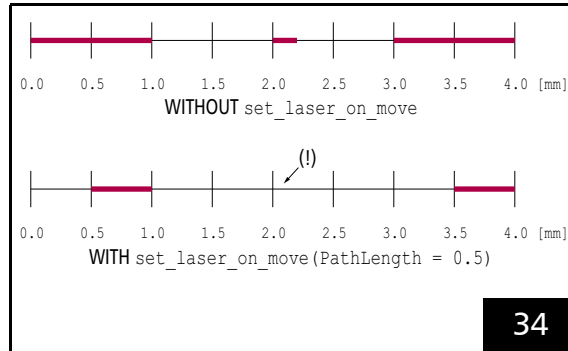
A **slsc_list_suppress_spotdistance_control** call (prior to **slsc_list_wait_with_laser_on**) suppresses the functionality to achieve equidistant spot distances until this state is cancelled by **slsc_list_unsuppress_spotdistance_control**.

As long as the suppression persists the following is used to trigger laser pulses:

- the **HalfPeriod** value of **slsc_ctrl_set_laser_pulses**, or
- the **HalfPeriod** value of **slsc_list_set_laser_pulses**, or
- the **HalfPeriod** attribute value from the **syncAXISConfig.xml** tag **LaserOutput**

About `slsc_list_set_laser_on_move`

`slsc_list_set_laser_on_move` delays the “Laser Active” Operation by exactly the amount of time needed to travel the specified path length (`PathLength`) on the current marking section, see Figure 34.



`slsc_list_set_laser_on_move`. See text for explanations, page 87.

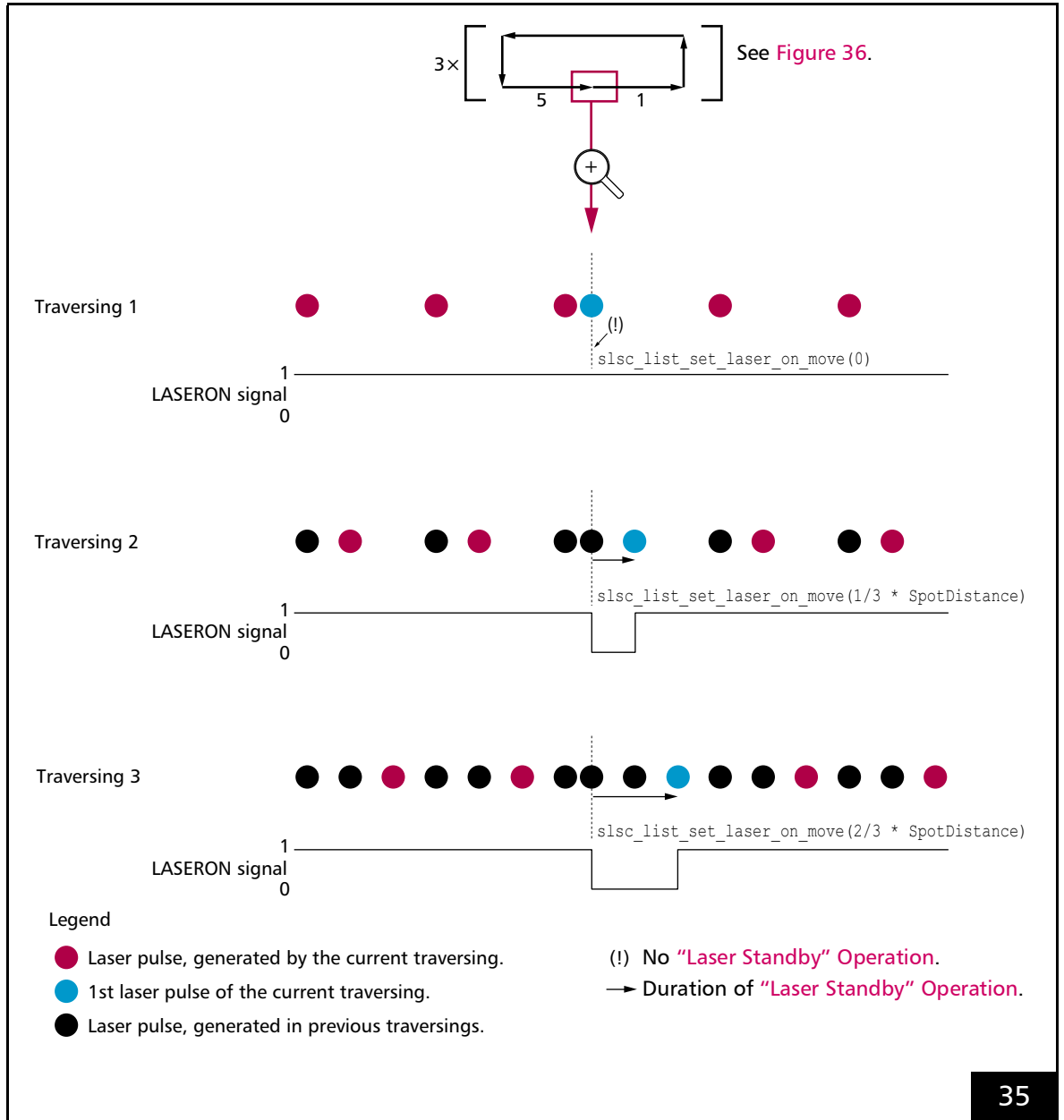
A use case is a marking pattern that is to be marked several times with equidistant laser spot distances (see `SpotDistance` and Chapter 2.9.5 “About the “Contour-dependent speed calculation””, page 60) (“multiple traversing”). For each repetition, the “Laser Active” Operation can be delayed (that is, the first laser pulse is shifted locally) by `slsc_list_set_laser_on_move`. Thus, the laser pulses of the individual traversings do not overlap, see Figure 35.

The following applies to a `slsc_list_set_laser_on_move` call between two marking pattern sections with blending curve:

- The blending curve is replaced by a Sky Writing-like motion

The following applies to a `slsc_list_set_laser_on_move` call between two marking pattern sections with “normal” traversing (= neither a Sky Writing-like motion nor a blending curve is inserted):

- At the transition of the two marking pattern sections, it is switched to “Laser Standby” Operation (see Figure 35, falling edge of the LASERON signal)
- After the `PathLength`, it is switched back to “Laser Active” Operation (see Figure 35, rising edge at the arrow ends). If necessary, a Sky Writing-like motion is inserted, so that the `LaserMinOffTime` is not undercut



slsc_list_set_laser_on_move. See text for explanations, [page 87](#).

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

size_t JobID;
slsc_list_begin(SLHandle, &JobID);

// target points: 2 x 1 mm rectangle
const std::array<double, 2> LowerMid = { 0.0, 0.0 };
const std::array<double, 2> LowerRightCorner = { 1.0, 0.0 };
const std::array<double, 2> UpperRightCorner = { 1.0, 1.0 };
const std::array<double, 2> UpperLeftCorner = { -1.0, 1.0 };
const std::array<double, 2> LowerLeftCorner = { -1.0, 0.0 };

// Say spot distance is 10 µm
const double SpotDistance = 0.01; // mm

// Jump to start position
slsc_list_jump_abs(SLHandle, LowerMid.data());

// Traversing 1
slsc_list_set_laser_on_move(SLHandle, 0.0);

slsc_list_mark_abs(SLHandle, LowerRightCorner.data());
slsc_list_mark_abs(SLHandle, UpperRightCorner.data());
slsc_list_mark_abs(SLHandle, UpperLeftCorner.data());
slsc_list_mark_abs(SLHandle, LowerLeftCorner.data());
slsc_list_mark_abs(SLHandle, LowerMid.data());

// Traversing 2
slsc_list_set_laser_on_move(SLHandle, 1.0 / 3.0 * SpotDistance);

slsc_list_mark_abs(SLHandle, LowerRightCorner.data());
slsc_list_mark_abs(SLHandle, UpperRightCorner.data());
slsc_list_mark_abs(SLHandle, UpperLeftCorner.data());
slsc_list_mark_abs(SLHandle, LowerLeftCorner.data());
slsc_list_mark_abs(SLHandle, LowerMid.data());

// Traversing 3
slsc_list_set_laser_on_move(SLHandle, 2.0 / 3.0 * SpotDistance);


slsc_list_mark_abs(SLHandle, LowerRightCorner.data());
slsc_list_mark_abs(SLHandle, UpperRightCorner.data());
slsc_list_mark_abs(SLHandle, UpperLeftCorner.data());
slsc_list_mark_abs(SLHandle, LowerLeftCorner.data());
slsc_list_mark_abs(SLHandle, LowerMid.data());

slsc_list_end(SLHandle);
```

Code example: [slsc_list_set_laser_on_move](#).

[*]dashed[*] Functions

- `slsc_list_dashed_arc_abs`
- `slsc_list_dashed_circle_2d_abs`
- `slsc_list_dashed_mark_abs`
- `slsc_list_multi_para_dashed_arc_abs`
- `slsc_list_multi_para_dashed_circle_2d_abs`
- `slsc_list_multi_para_dashed_mark_abs`
- `slsc_list_para_dashed_arc_abs`
- `slsc_list_para_dashed_circle_2d_abs`
- `slsc_list_para_dashed_mark_abs`

[*]dashed[*] Functions are specially designed to enable/disable the laser spatially frequent along marking pattern sections (see **Mark functions**). Use cases are, for example, marking patterns having hatchings from short lines .

To all [*]dashed[*] Functions, the following applies:

- They behave like their corresponding (without “_dashed” in the name) non-[*]dashed[*] Functions
- They provide additionally 2 arguments each for switching the laser. These behave the same with all [*]dashed[*] Functions:
 - `NSwitches`
 - `LaserSwitches`
- `NSwitches` is the size of the `LaserSwitches` array. It specifies how often the laser is to be switched on/off along the marking pattern section. Minimum value: 1.
- `LaserSwitches` is an array of `double` values. The array specifies at which arc length values (in mm) a switching of “Laser Standby” Operation and “Laser Active” Operation has to occur.

The first switching:

- Is defined to be a switch to “Laser Active” Operation
 - Is allowed for an arc length of 0.0 mm, that is, the laser is switched on immediately at the start of the marking pattern section

The `LaserSwitches` values must meet the following requirements:

- Ascending order
- ≥ 0.0
- \leq Total arc length of the marking pattern section
- Duration of “Laser Standby” Operation⁽¹⁾:
 $\geq [\text{LaserMinOffTime} + \text{LaserPreTriggerTime}]$
- Time between two switchings s of similar type⁽²⁾:
 $\geq 1 \mu\text{s}$
- If one of the above requirements is not met, the return values of the calling [*]dashed[*] Functions indicate that Bit #06 is set (`UnplausibleOrUnknownParameter`).
- Important: Regardless of the currently set `BlendMode`, marking pattern sections defined by [*]dashed[*] Functions are *never* part of a blending curve.

Notes

- Marking patterns suitable for [*]dashed[*] Functions could alternatively be achieved by a sequence of short **Mark functions** and **Jump commands**.
- However, there is a risk that the calculation time will increase significantly. The reason for this would be the geometrical calculations (see **Figure 11**), which have to be performed for each of these **Mark functions** and **Jump commands** separately. Their calculation durations are independent of the spatial extension of the marking pattern sections. In contrast, with [*]dashed[*] Functions, these calculations are performed only 1 ×, independent of the specified `NSwitches` value.
- In contrast, sequences of **Mark functions** and **Jump commands**:
 - Are suitable if the laser is to be switched on/off only relatively infrequently
 - Even must be used, if blending curves are desired

(1) The time differences result from the specified arc lengths and the marking speed.

(2) “Laser Standby” Operation -> “Laser Standby” Operation
 “Laser Active” Operation -> “Laser Active” Operation

Functions for Changing Target Point Coordinates

- **slsc_list_set_matrix_and_offset**
- **slsc_list_set_rot_and_offset_2d**

To be able to change target point coordinates (see page 268) of subsequent **slsc_list_arc_abs**, **slsc_list_circle_2d_abs**, **slsc_list_jump_abs**, **slsc_list_mark_abs** and their corresponding **slsc_list_[para/multi_para]*** functions as of the insert position only until the end of the current **Job**, **slsc_list_set_rot_and_offset_2d** and **slsc_list_set_matrix_and_offset** are available. With **slsc_list_set_rot_and_offset_2d** an angle and offset value can be specified, with **slsc_list_set_matrix_and_offset** a transformation matrix and an offset value. For both there are corresponding Configuration functions (**slsc_cfg_***) which are **slsc_cfg_set_rot_and_offset_2d** and **slsc_cfg_set_matrix_and_offset**.

The transformation matrix to be specified with **slsc_list_set_matrix_and_offset/slsc_cfg_set_matrix_and_offset** can, for example, be used for rotating (mathematically positive direction of rotation, that is, positive angles produce counterclockwise rotation), scaling, or flipping marking results. Some examples are shown in the following table.

Rotation by the angle α	$\begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix}$
Scaling by the factors k_x and k_y	$\begin{bmatrix} k_x & 0 \\ 0 & k_y \end{bmatrix}$
Mirroring around the y axis (flipping in the x direction)	$\begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$

Functions for Defining **Ramps** (**slsc_list_[para/multi_para]***-Functions)

- **slsc_list_multi_para_arc_abs**
- **slsc_list_multi_para_circle_2d_abs**
- **slsc_list_multi_para_dashed_arc_abs⁽¹⁾**
- **slsc_list_multi_para_dashed_circle_2d_abs⁽¹⁾**
- **slsc_list_multi_para_dashed_mark_abs⁽¹⁾**
- **slsc_list_multi_para_mark_abs**
- **slsc_list_para_arc_abs**
- **slsc_list_para_circle_2d_abs**
- **slsc_list_para_dashed_arc_abs⁽¹⁾**
- **slsc_list_para_dashed_circle_2d_abs⁽¹⁾**
- **slsc_list_para_dashed_mark_abs⁽¹⁾**
- **slsc_list_para_disable**
- **slsc_list_para_enable**
- **slsc_list_para_jump_abs**
- **slsc_list_para_jump_abs_min_time**
- **slsc_list_para_mark_abs**

slsc_list_para* functions (**slsc_list_para_arc_abs**, **slsc_list_para_circle_2d_abs**, **slsc_list_para_jump_abs**, **slsc_list_para_jump_abs_min_time**, **slsc_list_para_mark_abs**) offer the argument **ParaTarget** additionally (compared to their corresponding **slsc_list*** functions **slsc_list_arc_abs**, **slsc_list_circle_2d_abs**, **slsc_list_jump_abs**, **slsc_list_jump_abs_min_time**, **slsc_list_mark_abs**). By **ParaTarget**, a (simple) **Ramp** is defined (in the working field, the value/s of one/two "ActiveChannel" is/are varied linearly). In order that the argument **ParaTarget** is being evaluated, **slsc_list_para_enable** must have been called before. Otherwise, **slsc_list_para*** functions work as their corresponding **slsc_list*** functions.

Each **slsc_list_para*** function works as its corresponding **slsc_list*** function, if no "ActiveChannel" has been entered, see Section "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48, or **slsc_list_para_disable** has been called before.

(1) See Section "[*]dashed[*] Functions", page 91.

slsc_list_multi_para* functions

([slsc_list_multi_para_arc_abs](#), [slsc_list_multi_para_circle_2d_abs](#), [slsc_list_multi_para_mark_abs](#)) additionally offer the argument `MultiParaTarget` (compared to their corresponding `slsc_list*` functions [slsc_list_arc_abs](#), [slsc_list_circle_2d_abs](#), [slsc_list_mark_abs](#)). By `MultiParaTarget`, a [Ramp](#) (per “ActiveChannel”) can be defined which consist of several sections, which makes it also possible to define sawtooth- or square-shaped [Ramps](#), see also [Section “About Ramps”](#), [page 53](#).

For the argument `MultiParaTarget` to be evaluated, [slsc_list_para_enable](#) must have been called before. Otherwise, `slsc_list_multi_para*` functions work as their corresponding `slsc_list*` functions.

Each `slsc_list_multi_para*` function works as its corresponding `slsc_list*` function, if no “ActiveChannel” has been defined, see [Section “About Automatically Controlling the Laser by syncAXIS control \(“Automatic Laser Control”\)](#)”, [page 48](#), or [slsc_list_para_disable](#) has been called before.

Functions for Setting Signals

- [slsc_list_write_analog_x](#)
- [slsc_list_write_digital_out](#)
- [slsc_list_write_digital_out_mask](#)

With these functions, the signals are set at the specified output ports in between two Job functions (`slsc_list_*`; referring to the marking result in the image field), see also [Chapter 2.7.2 “About the Point in Time when Output Signals are actually set”](#), [page 45](#). They remain set beyond the [Job](#) end.

The most recently set signals are still outputted, even if the [syncAXIS control instance](#) is deleted by [slsc_cfg_delete](#).

Functions for Changing Speeds

- [slsc_list_set_jump_speed](#)
- [slsc_list_set_mark_speed](#)

A change by [slsc_list_set_jump_speed](#) and [slsc_list_set_mark_speed](#) applies to all (as of the insert position) following Job functions (`slsc_list_*`), but only until the end of the currently running [Job](#).

For both there are corresponding Configuration functions (`slsc_cfg_*`) which are [slsc_cfg_set_jump_speed](#) and [slsc_cfg_set_mark_speed](#).

Function for Changing Minimum Speeds

- [slsc_list_set_min_mark_speed](#)

A change by [slsc_list_set_min_mark_speed](#) applies to all (as of the insert position) following Job functions (`slsc_list_*`), but only until the end of the currently running [Job](#).

For [slsc_list_set_min_mark_speed](#), there is no corresponding Configuration function (`slsc_cfg_*`).

Functions for Changing Trajectory planning Values

- `slsc_list_set_calculation_dynamics_jump_scan_device`
- `slsc_list_set_calculation_dynamics_mark_scan_device`

A change by these functions applies to all (as of the insert position) following Job functions (`slsc_list_*`), but only until the end of the currently running Job.

With these functions, the values for acceleration and jerk used for planning trajectories⁽¹⁾ (for Operation mode “ScannerOnly” as well as “ScannerAndStage”) can be changed “locally” (= at a specific point in the Job).

Possible application scenarios:

- (1) Very high accuracy requirements are imposed on the marking result. When optimizing, such a “local” reduction of the scan head acceleration can further improve the marking result because it reduces the already low control error of the scan head even more.
- (2) For selective optimization of the Motion decomposition, when:
 - With the (Job-wide valid) Motion decomposition settings (= `FilterBandwidth` value, `DynamicReductionFunction` value), a relatively good result is already achieved overall
 - However, in a few places the positioning stage dynamic limits are still exceeded
 In these few places, ≥ V1.6 allows to specifically reduce `CalculationDynamics` values (as a simple workaround alternatively to changing the Motion decomposition parameter values again, see above).
 As a fix (with ≥ V1.6), `CalculationDynamics` values⁽²⁾ can be specifically reduced in these few places (instead of having to change the Motion decomposition settings, see above).

Functions for Changing the Behavior of Blending Curves

- `slsc_list_set_approx_blend_limit`

By `slsc_list_set_approx_blend_limit`, the `ApproxBlendLimit` value can be changed, see also Figure 39, page 292 in `slsc_GeometryConfig`.

A change by `slsc_list_set_approx_blend_limit` applies to all (as of the insert position) following Job functions (`slsc_list_*`), but only until the end of the currently running Job.

Function for the “Contour-dependent Speed Calculation”

- `slsc_list_set_contour_dependent_speed_control_2d`

The “Contour-dependent speed calculation” can be switched on and off as well as changed by `slsc_list_set_contour_dependent_speed_control_2d`. For prerequisites and further information, see Chapter 2.9.5 “About the “Contour-dependent speed calculation””, page 60.

A change by `slsc_list_set_contour_dependent_speed_control_2d` applies to all (as of the insert position) following Job functions (`slsc_list_*`), but only until the end of the currently running Job.

For `slsc_list_set_contour_dependent_speed_control_2d` there is the corresponding Configuration function (`slsc_cfg_*`) `slsc_cfg_set_contour_dependent_speed_control_2d`.

(1) See Trajectory, page 15.

(2) `<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:CalculationDynamics>`



Function for Setting the Value of a Free Variable on the RTC6

- **slsc_list_set_free_variable**

See [Section "Functions for Managing the Value of a Free Variable on the RTC6"](#), page 99.

Function for Influencing the Laser Pulse Output by HalfPeriod/PulseLength

- **slsc_list_set_laser_pulses**

See [Section "Function for Influencing the Laser Pulse Output by HalfPeriod/PulseLength"](#), page 100.

Functions for "Modules"

- **slsc_list_begin_module**
- **slsc_list_para_playback_module**
- **slsc_list_playback_module**

See [Chapter 2.11 "About Working with "Modules""](#), page 65.

3.1.3 Control Functions (slsc_ctrl_*)

Control functions (prefix **slsc_ctrl_**) serve to control the user program flow. These functions primarily serve the purpose of controlling the user program flow:

- To release/not release laser signals
- To query whether the syncAXIS-DLL-internal **Input buffer** has free capacity
- To query the execution state of the RTC6 board
- To query errors
- To query measured signals
- To start and to cancel a **Job** execution

For a graphical overview, see **Figure 37, page 98**.

Notes

- Control functions are technically totally different than “RTC control commands” which are described in the **RTC6 Manual**. Therefore, they are not designated as such in this document. Among other things, the **syncAXIS control instance** ensures that a Control function may/can be actually executed.
- Control functions (**slsc_ctrl_***) are always accepted, if the operation status is “green” (see **slsc_cfg_get_operation_status**).
- Some Control functions (**slsc_ctrl_***) have **Operation mode (slsc_OperationMode)** restrictions, see **Figure 37, page 98**.

Laser-related Functions

- **slsc_ctrl_disable_laser**
- **slsc_ctrl_enable_laser**



Warning!

Risk of injury due to laser radiation!
slsc_cfg_initialize_from_file can lead to undefined states of the RTC6 board(s) in which the laser could be switched on unexpectedly! Make sure that the laser is switched off before calling **slsc_cfg_initialize_from_file**!



Caution!

Make sure that laser safety is ensured in the entire system. In the safety concept of your system control, take into account that the RTC laser control signals are enabled by **slsc_cfg_initialize_from_file** and **slsc_ctrl_enable_laser**.

Among other things, **slsc_cfg_initialize_from_file** activates (arms) the laser control on the RTC6 board (internally the RTC6 control command **set_laser_control** is used for this purpose). Furthermore, **slsc_cfg_initialize_from_file** automatically executes **slsc_ctrl_enable_laser** among other things. That is, the laser control signals⁽¹⁾ LASERON, LASER1 and LASER2 are already actually outputted on the RTC6 board. This output can be inhibited by **slsc_ctrl_disable_laser**.

(1) See **RTC6 Manual**.

Execution-related Functions

To check whether the syncAXIS-DLL **Input buffer** is full and, therefore, cannot accept an additional Job function at the moment:

- **slsc_ctrl_is_list_input_buffer_full**

To query the execution status⁽¹⁾ of the RTC6:

- **slsc_ctrl_get_exec_state**

To start and to cancel a **Job**:

- **slsc_ctrl_start_execution**
- **slsc_ctrl_stop_controlled** or **slsc_ctrl_stop**, see Section "Comparison of **slsc_ctrl_stop_controlled** and **slsc_ctrl_stop**", page 97



Caution!

Make sure that laser safety is ensured in the entire system. In the safety concept of your system control, take into account that the laser is on during **Job** execution.



Caution!

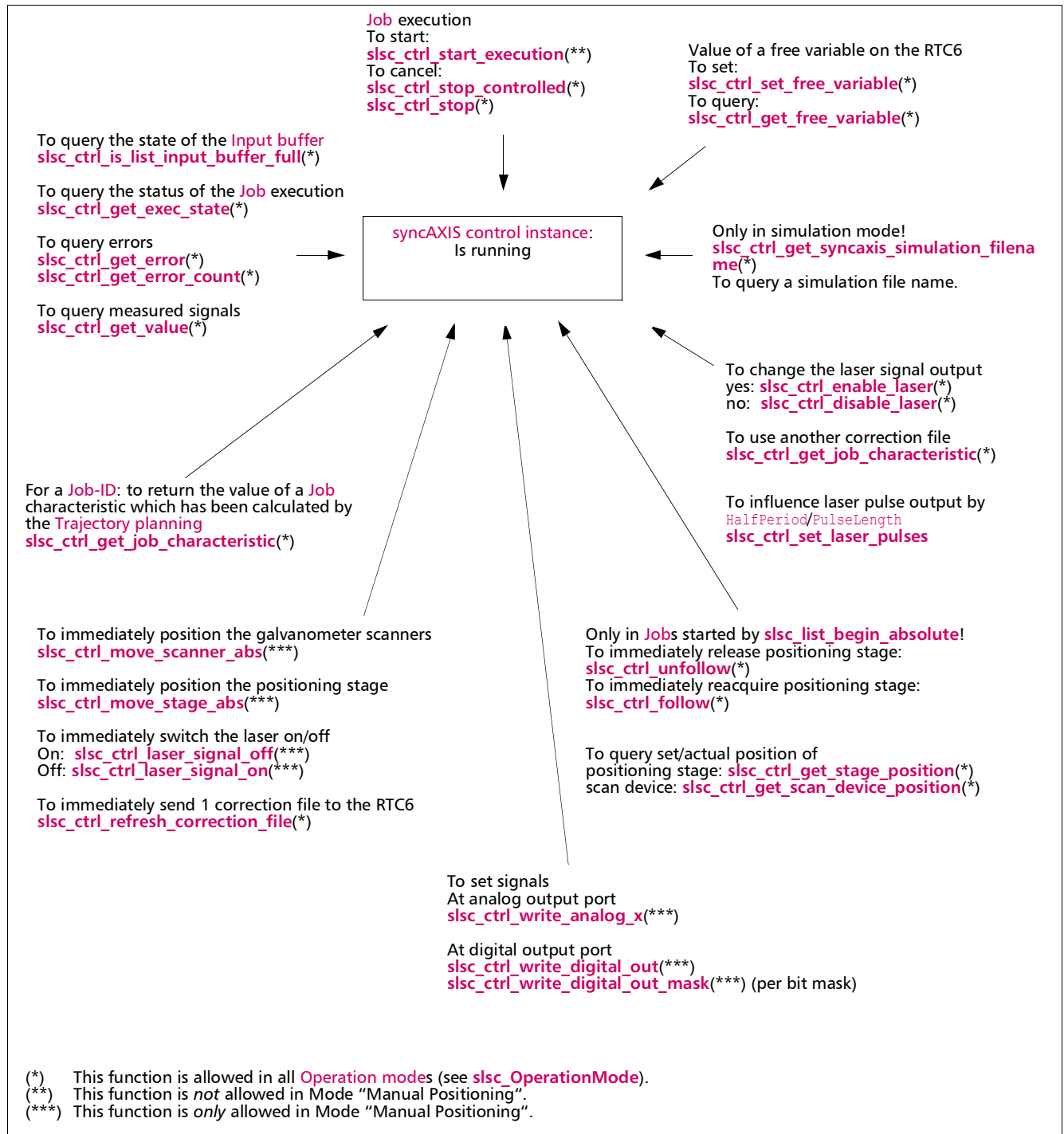
A moving positioning stage poses mechanical hazards. There are risks of injuries to fingers and hands from crushing. In the safety concept of your system control, take into account that **slsc_ctrl_start_execution** can move the positioning stage (possibly with a certain delay). Make sure that all bystanders keep sufficient distance to the appliance during execution.

Comparison of **slsc_ctrl_stop_controlled** and **slsc_ctrl_stop**

slsc_ctrl_stop_controlled	slsc_ctrl_stop
With compensation movement for deceleration	"Emergency stop"
Does not ensure the fastest possible standstill of the system, but maintains all specified dynamic limits	<ul style="list-style-type: none"> • Immediately turns off the laser • Brings mirrors and positioning stage to a standstill as quickly as possible • Wear may occur with frequent use
Subsequently, the syncAXIS control instance must be re-initialized	Subsequently, the syncAXIS control instance must be re-initialized

(1) **slsc_ExecState_Idle**,
slsc_ExecState_ReadyForExecution,
slsc_ExecState_Executing,
slsc_ExecState_NotInitOrError.

Control Functions (slsc_ctrl_*)



Correction File-related Functions

- **slsc_ctrl_refresh_correction_file**
- **slsc_ctrl_select_correction_file**

In a **syncAXISConfig.xml** for syncAXIS control \geq V1.1.0, up to 4 correction files can be entered⁽¹⁾. Upon creating the **syncAXIS control instance** these are transferred to the RTC6 and saved there.

By **slsc_ctrl_select_correction_file** it can be set (by specifying the corresponding index 0...3 of the desired correction file) which of these correction files is to be immediately used (for example, after having switched to another positioning stage-scan head combination which is now to be used for marking). So **slsc_ctrl_select_correction_file** selects (similar to the RTC6 command **select_cor_table**) a correction file which is already present on the RTC6.

In contrast, **slsc_ctrl_refresh_correction_file** (similar to the RTC6 command **load_correction_file**) immediately transfers a correction file to the RTC6 (the corresponding index 0...3 is to be specified, as with **slsc_ctrl_select_correction_file**). Example of an application scenario: a **syncAXIS control instance** is running which means that the correction files specified in **syncAXISConfig.xml** are saved on the RTC6. The user now generates an optimized correction file for the laser scan system and overwrites the corresponding correction file which is currently in use (in the file system). By **slsc_ctrl_refresh_correction_file** (without the need to destroy/recreate the **syncAXIS control instance**), the optimized correction file is then sent to the RTC6.

Error-related Functions

- **slsc_ctrl_get_error**
- **slsc_ctrl_get_error_count**

First of all, the number *n* of present errors must be determined by **slsc_ctrl_get_error_count**. Then a meaningful value for **ErrorNr** can be specified with **slsc_ctrl_get_error** in order to query details about the respective error. The first occurred error is the oldest detected error. It has the number 0.

Functions for Querying Measured Values

- **slsc_ctrl_get_value**

By **slsc_ctrl_get_value** a broad range of measured signals related to the 4 axes can be queried for the purpose of diagnosis and monitoring.

Functions Only for Mode "Manual Positioning"

- **slsc_ctrl_laser_signal_off**
- **slsc_ctrl_laser_signal_on**
- **slsc_ctrl_move_scanner_abs**
- **slsc_ctrl_move_stage_abs**

See Chapter 2.12 "About the Mode "Manual Positioning"", page 70.

Functions for Managing the Value of a Free Variable on the RTC6

- **slsc_ctrl_get_free_variable**
- **slsc_ctrl_set_free_variable**

The functions for free variables **slsc_ctrl_set_free_variable**, **slsc_ctrl_get_free_variable** as well as the supplementary Job function (**slsc_list**) **slsc_list_set_free_variable** each make the directly corresponding RTC6 command available in syncAXIS control. They can be used, for example, to determine and count increments (within **Jobs**). However, **slsc_ctrl_get_free_variable**, **slsc_ctrl_set_free_variable**, as well as **slsc_list_set_free_variable** have no effect in simulation mode.

For further information on free variables, refer to the RTC6 Manual, Chapter 6.9.1 "Free Variables", page 134.

(1) **syncAXIS_1_8.xsd** allows up to 4 **CorrectionFilePath** as child tags below **CorrectionFileList**.

Functions for Optimizing Parameter Values

- **slsc_ctrl_get_job_characteristic**

slsc_ctrl_get_job_characteristic is primarily intended to evaluate the effect of parameter value permutations (in simulation mode) by algorithms (that is, automating parameter value optimization).

slsc_ctrl_get_job_characteristic returns – for a specified **Job-ID** – the value of *one certain Job* characteristic (“**Key**”, see enum **slsc_JobCharacteristic**) which has been calculated by the **Trajectory planning**. The last 10 calculated **Jobs** (by **Job-ID**, see **slsc_list_begin**) can be queried. To query all characteristics of a **Job**, **slsc_ctrl_get_job_characteristic** must be called correspondingly several times.

However, **slsc_ctrl_get_job_characteristic** requires that the status of the specified **Job-ID** is at least “Calculation: Finished” (see **Figure 12**, page 43). Otherwise, the return value indicates that **Bit #06** is set (**UnplausibleOrUnknownParameter**). Therefore, with short **Job-ID**s (not quantifiable more precisely), **slsc_ctrl_get_job_characteristic** can also be used to implement confirmation messages in a GUI, that is, if the **Trajectory planning** calculation results (for example, max control values for the positioning stage) exceed defined limits (for example, “...do you really want to execute Job...”).

Functions for Starting/Ending the Mode “Manual Positioning”

- **slsc_ctrl_follow**
- **slsc_ctrl_unfollow**

See **Chapter 2.12 “About the Mode “Manual Positioning”**”, page 70.

Functions for Querying Positions

- **slsc_ctrl_get_scan_device_position**
- **slsc_ctrl_get_stage_position**

slsc_ctrl_get_scan_device_position returns the set position or actual position of the specified scan device. **slsc_ctrl_get_stage_position** does the same for the positioning stage.

Simulation Setting-related Function

- **slsc_ctrl_get_syncaxis_simulation_filename**

slsc_ctrl_get_syncaxis_simulation_filename replaces **slsc_ctrl_get_simulation_filename** because simulation files in V1.3 are no longer generated scan device-specific.

slsc_ctrl_get_syncaxis_simulation_filename requires the simulation mode. This function can be used to determine the simulation file name (for a specific **Job-ID**). This facilitates the realization of user programs that automatically import simulation files (for example, to evaluate or graphically display them).

Functions for Setting Signals

- **slsc_ctrl_write_analog_x**
- **slsc_ctrl_write_digital_out**
- **slsc_ctrl_write_digital_out_mask**

These functions are only allowed in Mode “Manual Positioning”. They are not accepted, when a **Job** is currently being executed. Each of these functions is always sent to all RTC6 boards and the signals are set promptly.

The most recently set signals are still outputted, even if the **syncAXIS control instance** is deleted by **slsc_cfg_delete**.

Function for Influencing the Laser Pulse Output by HalfPeriod/PulseLength

- **slsc_ctrl_set_laser_pulses**

slsc_ctrl_set_laser_pulses and **slsc_list_set_laser_pulses** are provided for those Sky Writings who (due to the laser they use) cannot use the “Automatic Laser Control” to achieve equidistant spot distances and instead want to influence the pulse output via **HalfPeriod** and **PulseLength**.

However, **slsc_ctrl_set_laser_pulses** as well as **slsc_list_set_laser_pulses** have no effect in simulation mode.

3.1.4 Utility Functions (slsc_util_*)

Utility Functions (prefix `slsc_util_`)⁽¹⁾⁽²⁾ essentially replace external utility programs and must meet certain preconditions. They must not be called during the regular syncAXIS control operation.
For a graphical overview, see [Figure 38, page 101](#).

RTC6 board-related Function

- `slsc_util_reset_pcie`



Warning!

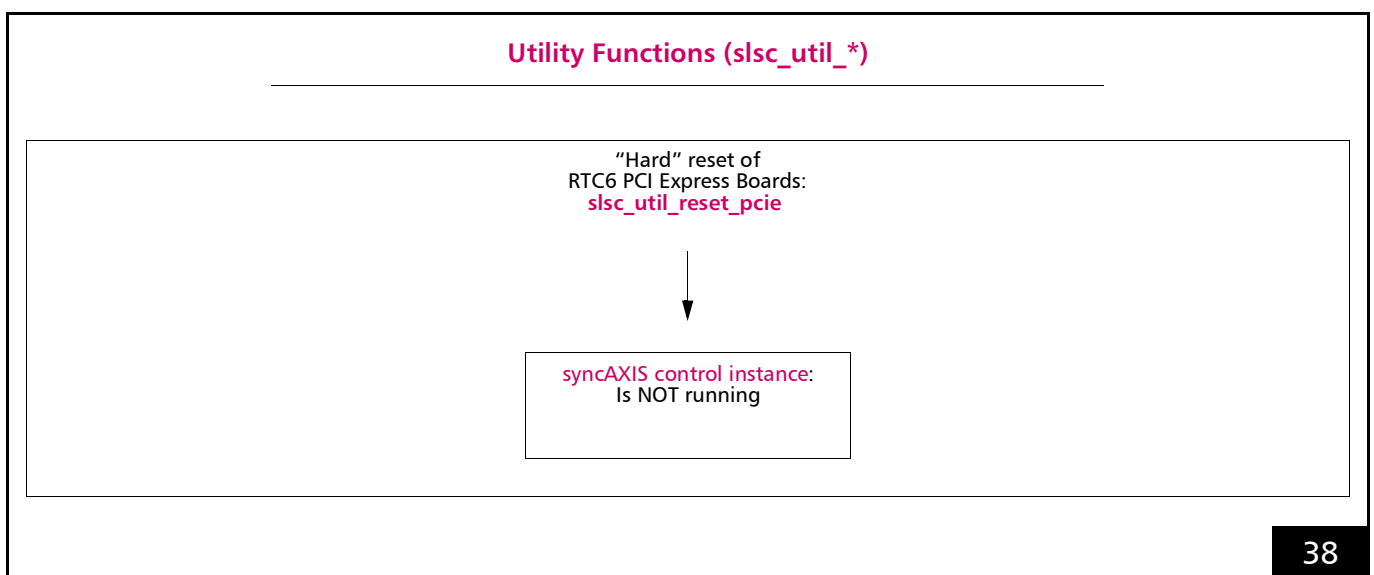
Risk of injury due to laser radiation!
`slsc_util_reset_pcie` can lead to undefined states of the RTC6 board(s) in which the laser could be switched on unexpectedly! Make sure that the laser is switched off before calling `slsc_util_reset_pcie`!

Apart from the preceding safety notice, make sure that no syncAXIS control instance is running before `slsc_util_reset_pcie` is called. `slsc_util_reset_pcie` is a “hard” reset function, which can be used if one of the RTC6 PCI Express Boards is in a state that users perceive as “strange”⁽³⁾.

(1) Available in syncAXIS control \geq V1.3.0.

(2) To date (V1.3.0), only one.

(3) In this situation it is not advisable to use `iSCANcfg.exe`, because mostly RTC6 files are loaded which do not originate from the syncAXIS control-software package which results in unpredictable side effects.



Utility Functions (`slsc_util_*`): graphical overview.

3.2 Alphabetical Overview

In this Chapter:

- Configuration Functions (slsc_cfg_*), page 102
- Control Functions (slsc_ctrl_*), page 107
- Job Functions (slsc_list_*), page 109
- Utility Functions (slsc_util_*), page 113

Configuration Functions (slsc_cfg_*)	Purpose
slsc_cfg_acquire_stage (deprecated)	Deprecated.
slsc_cfg_delete	Destroys the specified syncAXIS control instance. In the process, the resources (RTC6 board, positioning stage, ...) are released.
slsc_cfg_delete_trajectory_config	Auxiliary function for software development: deletes the trajectory configuration object (in order to avoid memory leaks), see Code example.
slsc_cfg_get_calculation_dynamics_stage	Returns the current setting of the specified syncAXIS control instance for: The maximum dynamic capabilities ("dynamic limits") of the intended positioning stage type. The values are used only in Trajectory planning calculations of the positioning stage motion.
slsc_cfg_get_dynamic_limits_scan_device	Returns the current setting of the specified syncAXIS control instance for: The maximum dynamic capabilities ("dynamic limits") of the intended scan device type.
slsc_cfg_get_dynamic_limits_stage	Returns the current setting of the specified syncAXIS control instance for: The dynamic limits of the intended positioning stage type.
slsc_cfg_get_dynamic_violation_reaction	Returns the current setting of the specified syncAXIS control instance for: The reaction when a limit value exceedance occurs.
slsc_cfg_get_field_limits_scan_device	Returns the current setting of the specified syncAXIS control instance for: The working field limits of the intended scan device type.
slsc_cfg_get_field_limits_stage	Returns the current setting of the specified syncAXIS control instance for: The working field limits of the intended positioning stage type.
Alphabetical Overview, page 102	

Configuration Functions (slsc_cfg_*) (cont'd.)	Purpose (cont'd.)
slsc_cfg_get_calculation_dynamics_jump_scan_device	Returns the current setting of the specified syncAXIS control instance for: The maximum acceleration and jerk value of the intended scan device type. The values are used only in Trajectory planning calculations of the scan device motion – however, only for jumps but not markings.
slsc_cfg_get_calculation_dynamics_mark_scan_device	Returns the current setting of the specified syncAXIS control instance for: The maximum acceleration and jerk value of the intended scan device type. The values are used only in Trajectory planning calculations of the scan device motion – however, only for markings but not jumps.
slsc_cfg_get_jump_time	Calculates the duration of a jump (outside “regular” Job): <ul style="list-style-type: none"> Based on the current setting of the specified syncAXIS control instance and Depending on the specified values for start dynamic and end dynamic
slsc_cfg_get_mode	Returns the current setting of the specified syncAXIS control instance for: The Operation mode (ScannerOnly, StageOnly, ScannerAndStage).
slsc_cfg_get_operation_status	Returns the current setting of the specified syncAXIS control instance for: The operation status (“traffic light color”).
slsc_cfg_get_scan_device_dynamic_monitoring_level	Returns the current setting of the specified syncAXIS control instance for: The criterion for which the scan devices are to be monitored (for example, slsc_DynamicsMonitoringLevel_Velocity).
slsc_cfg_get_simulation_setting	Returns the current setting of the specified syncAXIS control instance for: The Simulation Setting.
slsc_cfg_get_stage_dynamic_monitoring_level	Returns the current setting of the specified syncAXIS control instance for: The criterion for which the positioning stages are to be monitored (for example, slsc_DynamicsMonitoringLevel_Velocity).
slsc_cfg_get_sync_axis_version	Returns version info on the currently running syncAXIS-DLL.
slsc_cfg_get_trajectory_config	Returns the current setting of the specified syncAXIS control instance for: The Trajectory planning configuration.
slsc_cfg_initialize_copy	Initialization function: Creates a new (target-)syncAXIS control instance in simulation mode with the current configuration of the specified (source-)syncAXIS control instance (in either hardware mode or simulation mode) and assigns it a unique Handle value.
Alphabetical Overview, page 102	

Configuration Functions (slsc_cfg_*) (cont'd.)	Purpose (cont'd.)
slsc_cfg_initialize_from_file	Initialization function: Creates (by using the specified XML configuration file) a new syncAXIS control instance and assigns a unique <code>Handle</code> value to it.
slsc_cfg_register_callback_job_end_planned	Sets up that the specified "Callback function" is called when a "Callback event" of type "job_end_planned" occurs.
slsc_cfg_register_callback_job_finished_executing	Sets up that the specified "Callback function" is called when a "Callback event" of type "job_finished_executing" occurs.
slsc_cfg_register_callback_job_is_executing	Sets up that the specified "Callback function" is called when a "Callback event" of type "job_is_executing" occurs.
slsc_cfg_register_callback_job_loaded_enough	Sets up that the specified "Callback function" is called when a "Callback event" of type "job_loaded_enough" occurs.
slsc_cfg_register_callback_job_progress_planned	Sets up that the specified "Callback function" is called when a "Callback event" of type "job_progress_planned" occurs.
slsc_cfg_register_callback_job_start_planned	Sets up that the specified "Callback function" is called when a "Callback event" of type "job_start_planned" occurs.
slsc_cfg_reinitialize	Initialization function: Destroys the specified (by the <code>Handle</code>) syncAXIS control instance and creates it again (by using the momentary configuration settings and values that have been previously read out). In the process, the <code>Handle</code> value remains unchanged.
slsc_cfg_reinitialize_from_file	Initialization function: Destroys the specified (by the <code>Handle</code>) syncAXIS control instance and creates it again (by using the specified <code>syncAXISConfig.xml</code>). In the process, the <code>Handle</code> value remains unchanged.
slsc_cfg_release_stage (deprecated)	Deprecated.
slsc_cfg_select_heuristic	For specifying the speed reduction characteristic (<code>DynamicReductionFunction</code>).
slsc_cfg_select_stage	For specifying the target positioning stage ("positioning stage change"). As of syncAXIS-DLL ≥ V1.2.0, <code>slsc_cfg_select_stage</code> replaces <code>slsc_cfg_select_stage_axis</code> (deprecated).
slsc_cfg_select_stage_axis (deprecated)	Deprecated.
slsc_cfg_set_bandwidth	Changes the <code>FilterBandwidth</code> value of the specified syncAXIS control instance.
Alphabetical Overview, page 102	

Configuration Functions (slsc_cfg_*) (cont'd.)	Purpose (cont'd.)
slsc_cfg_set_calculation_dynamics_jump_scan_device	Changes the setting of the specified syncAXIS control instance for: The maximum acceleration and jerk value of the intended scan device type. The values are used only in Trajectory planning calculations of the scan device motion – however, only for jumps but not markings.
slsc_cfg_set_calculation_dynamics_mark_scan_device	Changes the setting of the specified syncAXIS control instance for: The maximum acceleration and jerk value of the intended scan device type. The values are used only in Trajectory planning calculations of the scan device motion – however, only for markings but not jumps.
slsc_cfg_set_calculation_dynamics_stage	Changes the setting of the specified syncAXIS control instance for: The maximum dynamic capabilities (“dynamic limits”) of the intended positioning stage type. The values are used only in Trajectory planning calculations of the positioning stage motion.
slsc_cfg_set_contour_dependent_speed_control_2d	Switch on/off the “Contour-dependent speed calculation”. Furthermore, it is configured how the syncAXIS control instance internally determines speeds along curves (“left” or “right” of the curve mid-line; distance to it). Once the “Automatic Laser Control” is activated, these results are used to correspondingly set, for example, the laser spot distances equidistant.
slsc_cfg_set_dynamic_limits_scan_device	Changes the setting of the specified syncAXIS control instance for: The maximum dynamic capabilities (“dynamic limits”) of the intended scan device type.
slsc_cfg_set_dynamic_limits_stage	Changes the setting of the specified syncAXIS control instance for: The dynamic limits of the intended positioning stage type.
slsc_cfg_set_dynamic_violation_reaction	Changes the setting of the specified syncAXIS control instance for: The reaction when a limit value exceedance occurs.
slsc_cfg_set_field_limits_scan_device	Changes the setting of the specified syncAXIS control instance for: The working field limits of the intended scan device type.
slsc_cfg_set_field_limits_stage	Changes the setting of the specified syncAXIS control instance for: The working field limits of the intended positioning stage type.
slsc_cfg_set_jump_speed	Changes the setting of the specified syncAXIS control instance for: The jump speed.
Alphabetical Overview, page 102	

Configuration Functions (slsc_cfg_*) (cont'd.)	Purpose (cont'd.)
slsc_cfg_set_list_handling_mode	Sets the handling and the return behavior of the Job functions (slsc_list_*).
slsc_cfg_set_list_handling_mode_with_context	Like slsc_cfg_set_list_handling_mode. But in addition, a context can be specified. Sets the handling and the return behavior of the Job functions (slsc_list_*).
slsc_cfg_set_mark_speed	Changes the setting of the specified syncAXIS control instance for: The marking speed.
slsc_cfg_set_matrix_and_offset	Changes the setting of the specified syncAXIS control instance for: Target point coordinates according to a transformation matrix and an offset value.
slsc_cfg_set_mode	Changes the setting of the specified syncAXIS control instance for: The Operation mode (ScannerOnly, StageOnly, ScannerAndStage).
slsc_cfg_set_part_displacement	Applies a Matrix and an Offset to the set trajectory for the specified scan device (scan head). See Chapter 8.3 "About Transformations in syncAXIS control V1.2.4 and Higher", page 332.
slsc_cfg_set_rot_and_offset_2d	Changes the setting of the specified syncAXIS control instance for: Target point coordinates by an angle and an offset value.
slsc_cfg_set_scan_device_dynamic_monitoring_level	Changes the setting of the specified syncAXIS control instance for: The criterion for which the scan devices are to be monitored (for example, slsc_DynamicsMonitoringLevel_Velocity).
slsc_cfg_set_simulation_setting	Changes the setting of the specified syncAXIS control instance for: The Simulation Setting.
slsc_cfg_set_stage_dynamic_monitoring_level	Changes the setting of the specified syncAXIS control instance for: The criterion for which the positioning stages are to be monitored (for example, slsc_DynamicsMonitoringLevel_Velocity).
slsc_cfg_set_trajectory_config	Changes the setting of the specified syncAXIS control instance for: The Trajectory planning configuration.
Alphabetical Overview, page 102	

Control Functions (slsc_ctrl_*)	Purpose
slsc_ctrl_disable_laser	Inhibits that the laser control signals LASERON, LASER1 and LASER2 (see RTC6 Manual) are outputted at the RTC6 board.
slsc_ctrl_enable_laser	Releases the laser control signals LASERON, LASER1 and LASER2 (see RTC6 Manual) at the RTC6 board.
slsc_ctrl_follow	To re-acquire the positioning stage after a slsc_ctrl_unfollow.
slsc_ctrl_get_error	Returns information on an error that occurred (error number ErrorNr).
slsc_ctrl_get_error_count	Returns the number of present errors.
slsc_ctrl_get_exec_state	Returns the state of the Execution Layer.
slsc_ctrl_get_free_variable	Returns the current value of a free variable of the RTC6.
slsc_ctrl_get_job_characteristic	Returns – for a specified Job-ID – the value of a Job characteristic ("key", see enum slsc_JobCharacteristic) which has been calculated by the Trajectory planning.
slsc_ctrl_get_scan_device_position	Returns the set position or actual position of the specified scan device (scan head).
slsc_ctrl_get_simulation_filename	Deprecated.
slsc_ctrl_get_stage_position	Returns the set position or actual position of the positioning stage.
slsc_ctrl_get_syncaxis_simulation_filename	Only in simulation mode! Returns the corresponding simulation file name for a specified Job-ID.
slsc_ctrl_get_value	Returns the present value of the specified signals at the specified axis.
slsc_ctrl_is_list_input_buffer_full	Checks whether the syncAXIS-DLL Input buffer is full (and therefore, cannot accept an additional Job function (slsc_list_*) at the moment).
slsc_ctrl_laser_signal_off	Only in Mode "Manual Positioning": Switches the laser off immediately.
slsc_ctrl_laser_signal_on	Only in Mode "Manual Positioning": Switches the laser on immediately.
Alphabetical Overview, page 102	

Control Functions (slsc_ctrl_*) (cont'd.)	Purpose (cont'd.)
slsc_ctrl_move_scanner_abs	Only in Mode "Manual Positioning": Moves all scan devices to the specified position with jump speed (starting from the current position).
slsc_ctrl_move_stage_abs	Only in Mode "Manual Positioning": Moves the positioning stage to the specified position with the dynamics (acceleration and jerk) set by the ACS API (starting from the current position).
slsc_ctrl_refresh_correction_file	Immediately transfers a correction file to the RTC6 board.
slsc_ctrl_select_correction_file	To specify a correction file, which is to be used immediately.
slsc_ctrl_set_free_variable	Sets the value of a free variable on the RTC6.
slsc_ctrl_set_laser_pulses	Defines the output period and the pulse lengths for the laser signals LASER1 and LASER2 for "laser active" operation of the RTC6 board.
slsc_ctrl_start_execution	Tries to start the execution of a Job by the Execution Layer.
slsc_ctrl_stop	Cancels the execution of the current Job uncontrolled and immediately by a direct access to the RTC6 board ("Emergency stop").
slsc_ctrl_stop_controlled	Cancels the execution of the current Job controlled and inserts a compensation movement for deceleration.
slsc_ctrl_unfollow	The specified syncAXIS control instance temporarily releases the positioning stage. Then, it can be controlled externally (for example, by a non-syncAXIS control-based user program).
slsc_ctrl_write_analog_x	Only in Mode "Manual Positioning": Writes a output value to the 12-Bit-analog output port ANALOG OUT1 or ANALOG OUT2 of all RTC6 boards.
slsc_ctrl_write_digital_out	Only in Mode "Manual Positioning": Writes a 16-bit output value to the 16-bit digital output port DIGITAL OUT 0...DIGITAL OUT 15 of all RTC6 boards.
slsc_ctrl_write_digital_out_mask	Only in Mode "Manual Positioning"! Writes only those bits of the Value-values to the 16-bit digital output port of all RTC6 boards, which are specified in the user-defined bit mask (Mask parameter).
Alphabetical Overview, page 102	

Job Functions (slsc_list_*)	Purpose
slsc_list_arc_abs	Defines a to-be-marked circular arc (not: elliptical arc) by absolute coordinate values.
slsc_list_begin	Defines the beginning of a Job. Is 1 of 2 mandatory structure elements of a Job.
slsc_list_begin_absolute	Defines (alternatively to slsc_list_begin, slsc_list_begin_relative) the beginning of a Job to compensate a position change (performed in Mode "Manual Positioning") of the positioning stage caused by slsc_ctrl_move_stage_abs. Important: slsc_list_begin_absolute may cause the user program to crash when starting Jobs, if the end position of the preceding Job and the position specified with slsc_list_begin_absolute do not match!
slsc_list_begin_module	Only allowed in simulation mode. To "precalculate a Job". Defines the beginning of a to-be-recorded Job (Module) which is closed as usual by slsc_list_end.
slsc_list_begin_relative	Defines (alternatively to slsc_list_begin) the beginning of a Job. Is 1 of 2 mandatory structure elements of a Job.
slsc_list_circle_2d_abs	Defines a circle (not: ellipse) by the absolute coordinate value of the circle center. The parameter <i>Angle</i> determines the marking direction as well as the number of rotations (for example, $3,25 \times 2\pi$).
slsc_list_dashed_arc_abs	Like slsc_list_arc_abs, but the corresponding [*]dashed[*] Function. Therefore, offers the arguments <i>NSwitches</i> and <i>LaserSwitches</i> in addition, see Section "[*]dashed[*] Functions", page 91.
slsc_list_dashed_circle_2d_abs	Like slsc_list_circle_2d_abs, but the corresponding [*]dashed[*] Function. Therefore, offers the arguments <i>NSwitches</i> and <i>LaserSwitches</i> in addition, see Section "[*]dashed[*] Functions", page 91.
slsc_list_dashed_mark_abs	Like slsc_list_mark_abs, but the corresponding [*]dashed[*] Function. Therefore, offers the arguments <i>NSwitches</i> and <i>LaserSwitches</i> in addition, see Section "[*]dashed[*] Functions", page 91.
slsc_list_end	Defines the end of a Job. Is 1 of 2 mandatory structure elements of a Job.
slsc_list_jump_abs	Defines a jump by absolute coordinate values.
slsc_list_jump_abs_min_time	Like slsc_list_jump_abs. But additionally allows to specify a minimum duration for the jump.
slsc_list_mark_abs	Defines a mark vector by absolute coordinate values.
Alphabetical Overview, page 102	

Job Functions (slsc_list_*) (cont'd.)	Purpose (cont'd.)
<code>slsc_list_multi_para_arc_abs</code>	Like <code>slsc_list_arc_abs</code> . But offers the argument <code>MultiParaTarget</code> additionally, by which (per "ActiveChannel") a Ramp consisting of several sections is defined.
<code>slsc_list_multi_para_circle_2d_abs</code>	Like <code>slsc_list_circle_2d_abs</code> . But offers the argument <code>MultiParaTarget</code> additionally, by which (per "ActiveChannel") a Ramp consisting of several sections is defined.
<code>slsc_list_multi_para_dashed_arc_abs</code>	Like <code>slsc_list_multi_para_arc_abs</code> , but the corresponding <code>[*]dashed[*]</code> Function. Therefore, offers the arguments <code>NSwitches</code> and <code>LaserSwitches</code> in addition, see Section "[*]dashed[*] Functions", page 91.
<code>slsc_list_multi_para_dashed_circle_2d_abs</code>	Like <code>slsc_list_multi_para_circle_2d_abs</code> , but the corresponding <code>[*]dashed[*]</code> Function. Therefore, offers the arguments <code>NSwitches</code> and <code>LaserSwitches</code> in addition, see Section "[*]dashed[*] Functions", page 91.
<code>slsc_list_multi_para_dashed_mark_abs</code>	Like <code>slsc_list_multi_para_mark_abs</code> , but the corresponding <code>[*]dashed[*]</code> Function. Therefore, offers the arguments <code>NSwitches</code> and <code>LaserSwitches</code> in addition, see Section "[*]dashed[*] Functions", page 91.
<code>slsc_list_multi_para_mark_abs</code>	Like <code>slsc_list_mark_abs</code> . But offers the argument <code>MultiParaTarget</code> additionally, by which (per "ActiveChannel") a Ramp consisting of several sections is defined.
<code>slsc_list_para_arc_abs</code>	Like <code>slsc_list_arc_abs</code> . But offers the argument <code>ParaTarget</code> additionally, by which a Ramp is defined (in the working field, the value/s of one/two "ActiveChannel" is/are varied linearly).
<code>slsc_list_para_circle_2d_abs</code>	Like <code>slsc_list_circle_2d_abs</code> . But offers the argument <code>ParaTarget</code> additionally, by which a Ramp is defined (in the working field, the value/s of one/two "ActiveChannel" is/are varied linearly).
<code>slsc_list_para_dashed_arc_abs</code>	Like <code>slsc_list_para_arc_abs</code> , but the corresponding <code>[*]dashed[*]</code> Function. Therefore, offers the arguments <code>NSwitches</code> and <code>LaserSwitches</code> in addition, see Section "[*]dashed[*] Functions", page 91.
<code>slsc_list_para_dashed_circle_2d_abs</code>	Like <code>slsc_list_para_circle_2d_abs</code> , but the corresponding <code>[*]dashed[*]</code> Function. Therefore, offers the arguments <code>NSwitches</code> and <code>LaserSwitches</code> in addition, see Section "[*]dashed[*] Functions", page 91.
<code>slsc_list_para_dashed_mark_abs</code>	Like <code>slsc_list_para_dashed_mark_abs</code> , but the corresponding <code>[*]dashed[*]</code> Function. Therefore, offers the arguments <code>NSwitches</code> and <code>LaserSwitches</code> in addition, see Section "[*]dashed[*] Functions", page 91.
Alphabetical Overview, page 102	

Job Functions (slsc_list_*) (cont'd.)	Purpose (cont'd.)
slsc_list_para_disable	Switches the processing of the arguments ParaTarget (of slsc_list_para* functions) and MultiParaTarget (of slsc_list_multi_para* functions) off.
slsc_list_para_enable	Switches the processing of the arguments ParaTarget (of slsc_list_para* functions) and MultiParaTarget (of slsc_list_multi_para* functions) on.
slsc_list_para_jump_abs	Like slsc_list_jump_abs . But offers the argument ParaTarget additionally, by which a Ramp is defined (in the working field, the value/s of one/two "ActiveChannel" is/are varied linearly).
slsc_list_para_jump_abs_min_time	Like slsc_list_jump_abs_min_time . But offers the argument ParaTarget additionally, by which a Ramp is defined (in the working field, the value/s of one/two "ActiveChannel" is/are varied linearly).
slsc_list_para_mark_abs	Like slsc_list_jump_abs_min_time . But offers the argument ParaTarget additionally, by which a Ramp is defined (in the working field, the value/s of one/two "ActiveChannel" is/are varied linearly).
slsc_list_para_playback_module	Like slsc_list_mark_abs . But offers the argument ParaTarget additionally, by which a Ramp is defined (in the working field, the value/s of one/two "ActiveChannel" is/are varied linearly).
slsc_list_playback_module	Like slsc_list_playback_module , however, parameter values on Ramps are applied, if there is a slsc_list_para_enable in advance.
slsc_list_set_approx_blend_limit	Changes the ApproxBlendLimit value, which is specified in the configuration of the Trajectory planning (see below). This change applies to all following Job functions (slsc_list_*) but only until the end of the Job.
slsc_list_set_calculation_dynamics_jump_scan_device	Changes: The maximum acceleration and jerk value of the intended scan device type. The values are used only in Trajectory planning calculations of the scan device motion – however, only for jumps but not markings
slsc_list_set_calculation_dynamics_mark_scan_device	Changes: The maximum acceleration and jerk value of the intended scan device type. The values are used only in Trajectory planning calculations of the scan device motion – however, only for markings but not jumps.
Alphabetical Overview, page 102	

Job Functions (slsc_list_*) (cont'd.)	Purpose (cont'd.)
slsc_list_set_contour_dependent_speed_control_2d	Switches on/off the “Contour-dependent speed calculation”. Furthermore, it can be changed how the syncAXIS control instance internally determines speeds along curves (“left” or “right” of the curve mid-line; distance to it). Once the “Automatic Laser Control” is activated, these results are used to correspondingly set, for example, the laser spot distances equidistant. This change applies to all following Job functions (slsc_list_*) but only until the end of the Job.
slsc_list_set_free_variable	Like slsc_ctrl_set_free_variable.
slsc_list_set_jump_speed	Changes the jump speed. This change applies to all following Job functions (slsc_list_*) but only until the end of the Job.
slsc_list_set_laser_on_move	Delays the “Laser Active” Operation by exactly the amount of time needed to travel the specified path length (PathLength) on the current marking section. This change applies to all following Job functions (slsc_list_*) but only until the end of the Job.
slsc_list_set_laser_pulses	Like slsc_ctrl_set_laser_pulses.
slsc_list_set_mark_speed	Changes the marking speed. This change applies to all following Job functions (slsc_list_*) but only until the end of the Job.
slsc_list_set_matrix_and_offset	Changes target point coordinates according to a transformation matrix and an offset value. This change applies to all following Job functions (slsc_list_*) but only until the end of the Job.
slsc_list_set_min_mark_speed	Changes the minimal marking speed, see MinimalMarkSpeed. This change applies to all following Job functions (slsc_list_*) but only until the end of the Job.
slsc_list_set_rot_and_offset_2d	Changes target point coordinates by an angle and an offset value. This change applies to all following Job functions (slsc_list_*) but only until the end of the Job.
slsc_list_suppress_spotdistance_control	Only if “Automatic Laser Control” is active with SpotDistance as an “ActiveChannel”: supplementary function that must precede slsc_list_wait_with_laser_on.
slsc_list_unsuppress_spotdistance_control	Cancels the effect of slsc_list_suppress_spotdistance_control.
Alphabetical Overview, page 102	

Job Functions (slsc_list_*) (cont'd.)	Purpose (cont'd.)
slsc_list_wait_with_laser_off	Like slsc_list_wait_with_laser_on, but the laser is switched off.
slsc_list_wait_with_laser_on	Defines a waiting time with which the laser spot is to wait at the last defined target point with the laser switched on.
slsc_list_write_analog_x	Writes a output value to the 12-Bit-analog output port ANALOG OUT1 or ANALOG OUT2 of all RTC6 boards.
slsc_list_write_digital_out	Writes a 16-bit output value to the 16-bit digital output port DIGITAL OUT 0...DIGITAL OUT 15 of all RTC6 boards.
slsc_list_write_digital_out_mask	Writes only those bits of the Value-values to the 16-bit digital output port of all RTC6, which are specified in the user-defined bit mask (Mask parameter).
Alphabetical Overview, page 102	

Utility Functions (slsc_util_*)	Purpose
slsc_util_reset_pcie	Carries out a “hard” reset of all found RTC6 PCI Express Boards.
Alphabetical Overview, page 102	

3.3 Function Reference

In this chapter:

- [Chapter 3.3.1 "General Structure of the Reference Tables", page 114](#)
- [Chapter 3.3.2 "Data Types of the syncAXIS-DLL Functions", page 115](#)
- [Chapter 3.3.3 "Reference Tables", page 117](#)

3.3.1 General Structure of the Reference Tables

Name of the function	<p>prefix_name</p> <p>The prefix indicates the category of the function:</p> <p>slsc_cfg_ – Configuration function</p> <p>slsc_ctrl_ – Control function</p> <p>slsc_list_ – Job function</p>
Purpose	Short description describing the purpose of the function.
Function signature	<pre>datatype prefix_name(datatype A, datatype* B, datatype C);</pre> <p> > Line Argument(s) C</p> <p> > Line Argument(s) B^(a)</p> <p> > Line Argument(s) A</p> <p> > Lines Name of the function, Purpose</p> <p>> Line Return value</p> <p>Example: <code>uint32_t slsc_list_arc_abs(size_t Handle, const double* Mid, const double* Target);</code></p>
Argument(s)	<p>A Data type. Short text.</p>
	<p>B Data type. Short text.</p>
	<p>C Data type. Short text.</p>
Return value	Reference to a description of the return value, for example, "See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions", page 279 ".
Comment(s)	<ul style="list-style-type: none"> • Additional information on this and similar functions. • References to other chapters and publications.
Code example	<p>Exemplary source code snippet (not compilable).</p> <pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16.</pre>
Version info	States the syncAXIS-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.

3.3.2 Data Types of the syncAXIS-DLL Functions

C programming language	Data format
bool	boolean value (true, false).
bool*	Pointer to a boolean value (true, false).
char	A presentable character of 1 byte = 8 bit.
char*	Pointer to a null-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.
size_t	As defined in <code>stddef.h</code> . In general, <code>uint32_t</code> for Win32 Executables.
size_t*	Pointer to a size_t value.
slsc_ExecTimeCallback	<p>Auxiliary data type. Dictates the signature for (the to-be-supplied by the user) the “Callback function”, which needs to be specified with:</p> <ul style="list-style-type: none"> • <code>slsc_cfg_register_callback_job_end_planned</code> • <code>slsc_cfg_register_callback_job_finished_executing</code> • <code>slsc_cfg_register_callback_job_is_executing</code> • <code>slsc_cfg_register_callback_job_progress_planned</code> <p>Function signature:</p> <pre>typedef void(*slsc_ExecTimeCallback)(size_t JobID, uint64_t Progress, double ExecTime, void* Context);</pre> <p>Argument(s):</p> <p>JobID Job-ID.</p> <p>Progress Reserved.</p> <p>ExecTime Execution time of this Job-ID up to here in ms. This information about the duration of planned or executed motions can be used for process evaluation and optimization. Note that the measured times will fluctuate because MS Windows is not a real time system.</p> <p>Context Pointer to the object which has been referenced in the respective <code>cfg_register_callback</code> function.</p> <p>Comment(s):</p> <ul style="list-style-type: none"> • <code>slsc_ExecTimeCallback</code> can be used, for example, as cast for function pointers or as type for lambda functions.

C programming language	Data format
<code>slsc_JobCallback</code>	<p>Auxiliary data type. Dictates the signature for (the to-be-supplied by the user) the “Callback function”, which needs to be specified with:</p> <ul style="list-style-type: none"> • <code>slsc_cfg_register_callback_job_loaded_enough</code> • <code>slsc_cfg_register_callback_job_start_planned</code> <p>Function signature:</p> <pre>typedef void (*slsc_JobCallback) (size_t JobID, void* Context);</pre> <p>Argument(s):</p> <p>JobID Job-ID.</p> <p>Context Pointer to the object which has been referenced in the respective <code>cfg_register_callback</code> function.</p> <p>Comment(s):</p> <ul style="list-style-type: none"> • <code>slsc_JobCallback</code> can be used, for example, as cast for function pointers or as type for lambda functions.
<code>uint16_t</code>	Synonym: unsigned short. Unsigned 16-bit value: $[0 \dots + (2^{16} - 1)]$.
<code>uint32_t</code>	Synonym: unsigned int. Unsigned 32-bit value: $[0 \dots + (2^{32} - 1)]$.
<code>uint64_t</code>	Synonym: unsigned long long. Unsigned 64-bit value: $[0 \dots + (2^{64} - 1)]$.
<code>uint32_t*</code>	Pointer to a unsigned 32-bit value : $[0 \dots + (2^{32} - 1)]$.
<code>uint64_t*</code>	Pointer to a unsigned 64-bit value : $[0 \dots + (2^{64} - 1)]$.

Notes

- ******
means pointer to a pointer, for example, with **`slsc_cfg_get_trajectory_config`**.
- **const**
(for example, with `const double* Target`) 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.

3.3.3 Reference Tables

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

Name of the function	slsc_cfg_acquire_stage (deprecated)
Purpose	<p>Deprecated.</p> <p>Use slsc_ctrl_follow/slsc_ctrl_unfollow instead.</p> <p>Resets the specified syncAXIS control instance (after slsc_cfg_release_stage (deprecated)) to a fully functional state. RTC6 board and positioning stage are acquired back again.</p>
Function signature	<code>uint32_t slsc_cfg_acquire_stage (deprecated) (size_t Handle);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • Prior to slsc_cfg_acquire_stage (deprecated), slsc_cfg_release_stage (deprecated) should have been called. • slsc_cfg_acquire_stage (deprecated) resets the specified syncAXIS control instance (after slsc_cfg_release_stage (deprecated)) to a fully functional state. In the process, RTC6 board and positioning stage are acquired back again. As a consequence, the syncAXIS control instance can be used without restrictions (see page 150) again (= <code>slsc_OperationStatus_Green</code>). The process is typically completed in less than 0.1 s. • Prior to slsc_cfg_acquire_stage (deprecated), slsc_cfg_select_stage can be called in order to specify a different positioning stage than the previous one. In this case, slsc_cfg_acquire_stage (deprecated) acquires a different positioning stage than has been released by slsc_cfg_release_stage (deprecated). • See also Comment(s) of slsc_cfg_release_stage (deprecated), page 150. • See also Chapter 2.12.2 "Example – Temporarily Releasing the Positioning Stage and Changing the Target Positioning Stage", page 74. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	See slsc_cfg_release_stage (deprecated) .
Version info	Available as of syncAXIS-DLL V0.11.0. Deprecated as of syncAXIS-DLL V1.0.7.
References	slsc_cfg_release_stage (deprecated)

Name of the function	<code>slsc_cfg_delete</code>
Purpose	Destroys the specified syncAXIS control instance . In the process, the resources (RTC6 board, positioning stage, ...) are released.
Function signature	<code>uint32_t slsc_cfg_delete(size_t Handle);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • slsc_cfg_delete is not executed, if the specified Handle value does not exist. Then, the return value indicates that Bit #02 is set (<code>NotAllowedWithoutInitialization</code>). • slsc_cfg_delete also deletes the Handle value of the specified syncAXIS control instance. This must be taken into account in particular if you are managing Handle values in your user program (see comment in Code example at slsc_cfg_initialize_from_file). • During the destruction (see page 26 for the build-up) of a syncAXIS control instance in hardware mode, the following processes take place: <ul style="list-style-type: none"> – Scan head: is released. The scan head mirror position remains as is (unchanged). – RTC6: is released. The RTC6 laser control is no longer active. – Positioning stage: is released. The positioning stage position remains as is (unchanged). – Output signals: the most recently set signals are continued to be outputted. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	slsc_cfg_initialize_from_file

Name of the function	slsc_cfg_delete_trajectory_config
Purpose	Auxiliary function for software development: deletes the trajectory configuration object (in order to avoid memory leaks), see Code example .
Function signature	<code>uint32_t slsc_cfg_delete_trajectory_config(slsc_TrajectoryConfig** TrajConfig);</code>
Argument(s)	TrajConfig See structure slsc_TrajectoryConfig .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_delete_trajectory_config does not change any configuration parameter values. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning" ", page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. slsc_TrajectoryConfig* TrajConfig = 0; // Handle: see Code example at slsc_cfg_initialize_from_file // get configuration parameters slsc_cfg_get_trajectory_config(Handle, &TrajConfig); // change configuration parameters TrajConfig->GeometryConfig.BlendMode = slsc_BlendModes::slsc_BlendModes_Deactivated; slsc_cfg_set_trajectory_config(Handle, TrajConfig); // delete configuration object slsc_TrajectoryConfig // (does not change the configuration parameters) slsc_cfg_delete_trajectory_config(&TrajConfig);</pre>
Version info	Available as of syncAXIS-DLL V0.11.0.
References	slsc_cfg_get_trajectory_config , slsc_cfg_set_trajectory_config

Name of the function	<code>slsc_cfg_get_calculation_dynamics_jump_scan_device</code>
Purpose	<p>Returns the current setting of the specified syncAXIS control instance for:</p> <ul style="list-style-type: none"> The maximum acceleration and jerk value of the intended scan device type. The values are used only in Trajectory planning calculations of the scan device motion – however, only for jumps but not markings
Function signature	<pre>uint32_t slsc_cfg_get_calculation_dynamics_jump_scan_device(size_t Handle, double* JumpAngularAcc, double* JumpAngularJerk);</pre>
Argument(s)	<p>Handle Handle to a syncAXIS control instance.</p>
	<p>JumpAngularAcc Like JumpAngularAcc.</p>
	<p>JumpAngularJerk Like JumpAngularJerk.</p>
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> Available to change the setting is: <ul style="list-style-type: none"> <code>slsc_cfg_set_calculation_dynamics_jump_scan_device</code> <code>slsc_list_set_calculation_dynamics_jump_scan_device</code> For <code>slsc_cfg_get_calculation_dynamics_jump_scan_device</code>, there is: <ul style="list-style-type: none"> A corresponding Job function (<code>slsc_list_*</code>) <code>slsc_list_set_calculation_dynamics_jump_scan_device</code> No corresponding Control function (<code>slsc_ctrl_*</code>) On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.6.0.
References	<code>slsc_cfg_set_calculation_dynamics_jump_scan_device</code> , <code>slsc_list_set_calculation_dynamics_jump_scan_device</code>

Name of the function	slsc_cfg_get_calculation_dynamics_mark_scan_device
Purpose	Returns the current setting of the specified syncAXIS control instance for: <ul style="list-style-type: none">The maximum acceleration and jerk value of the intended scan device type. The values are used only in Trajectory planning calculations of the scan device motion – however, only for markings but not jumps
Function signature	uint32_t slsc_cfg_get_calculation_dynamics_mark_scan_device(size_t Handle, double* MarkAngularAcc, double* MarkAngularJerk);
Argument(s)	Handle

Name of the function	<code>slsc_cfg_get_calculation_dynamics_stage</code>
Purpose	Returns the current setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The maximum dynamic capabilities ("dynamic limits") of the intended positioning stage type. The values are used only in Trajectory planning calculations of the positioning stage motion
Function signature	<code>uint32_t slsc_cfg_get_calculation_dynamics_stage(size_t Handle, slsc_Stage Stage, double* StageVel, double* StageAcc, double* StageJerk);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Stage Returned parameter value: pointer. See enum <code>slsc_Stage</code>.
	StageVel Like <code>StageVel</code>.
	StageAcc Like <code>StageAcc</code>.
	StageJerk Like <code>StageJerk</code>.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> Available to change the setting is: <ul style="list-style-type: none"> <code>slsc_cfg_set_calculation_dynamics_stage</code> For <code>slsc_cfg_get_calculation_dynamics_stage</code>, there is: <ul style="list-style-type: none"> No corresponding Job function (<code>slsc_list_*</code>) No corresponding Control function (<code>slsc_ctrl_*</code>) On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	<code>slsc_cfg_set_calculation_dynamics_stage</code>

Name of the function	<code>slsc_cfg_get_dynamic_limits_scan_device</code>
Purpose	Returns the current setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The maximum dynamic capabilities ("dynamic limits") of the intended scan device type
Function signature	<code>uint32_t slsc_cfg_get_dynamic_limits_scan_device(size_t Handle, double* AngularVel, double* AngularAcc, double* AngularJerk);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	AngularVel Like AngularVel.
	AngularAcc Like AngularAcc.
	AngularJerk Like AngularJerk.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> Available to change the setting is: <ul style="list-style-type: none"> <code>slsc_cfg_set_dynamic_limits_scan_device</code> For <code>slsc_cfg_get_dynamic_limits_scan_device</code>, there is: <ul style="list-style-type: none"> No corresponding Job function (<code>slsc_list_*</code>) No corresponding Control function (<code>slsc_ctrl_*</code>) On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	<code>slsc_cfg_set_dynamic_limits_scan_device</code>

Name of the function	<code>slsc_cfg_get_dynamic_limits_stage</code>
Purpose	Returns the current setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The dynamic limits of the intended positioning stage type
Function signature	<code>uint32_t slsc_cfg_get_dynamic_limits_stage(size_t Handle, slsc_Stage Stage, double* StageVel, double* StageAcc, double* StageJerk);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Stage See enum <code>slsc_Stage</code>.
	StageVel Like <code>StageVel</code>.
	StageAcc Like <code>StageAcc</code>.
	StageJerk Like <code>StageJerk</code>.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> Available to change the setting is: <ul style="list-style-type: none"> <code>slsc_cfg_set_dynamic_limits_stage</code> For <code>slsc_cfg_get_dynamic_limits_stage</code>, there is: <ul style="list-style-type: none"> No corresponding Job function (<code>slsc_list_*</code>) No corresponding Control function (<code>slsc_ctrl_*</code>) On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. double StageVel; double StageAcc; double StageJerk; slsc_cfg_get_dynamic_limits_stage(Handle, slsc_Stage1, &StageVel, &StageAcc, &StageJerk);</pre>
Version info	Available as of syncAXIS-DLL V1.5.0.
References	<code>slsc_cfg_set_dynamic_limits_stage</code>

Name of the function	<code>slsc_cfg_get_dynamic_violation_reaction</code>
Purpose	Returns the current setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The reaction when a limit value exceedance occurs
Function signature	<code>uint32_t slsc_cfg_get_dynamic_violation_reaction(size_t Handle, slsc_DynamicViolationReaction* DynamicViolationReaction);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	DynamicViolationReaction Returned parameter value: pointer. See enum slsc_DynamicViolationReaction .
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> Available to change the setting is: <ul style="list-style-type: none"> slsc_cfg_set_dynamic_violation_reaction For slsc_cfg_get_dynamic_violation_reaction, there is: <ul style="list-style-type: none"> No corresponding Job function (slsc_list_*) No corresponding Control function (slsc_ctrl_*) On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. slsc_DynamicViolationReaction DynamicViolationReaction; // Handle: see Code example at slsc_cfg_initialize_from_file slsc_cfg_get_dynamic_violation_reaction(Handle, &DynamicViolationReaction);</pre>
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_cfg_set_dynamic_violation_reaction

Name of the function	<code>slsc_cfg_get_field_limits_scan_device</code>
Purpose	Returns the current setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The working field limits of the intended scan device type
Function signature	<code>uint32_t slsc_cfg_get_field_limits_scan_device(size_t Handle, double* FieldLimitsMin, double* FieldLimitsMax);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	FieldLimitsMin Like <code>FieldLimitsMin</code> .
	FieldLimitsMax Like <code>FieldLimitsMax</code> .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> Available to change the setting is: <ul style="list-style-type: none"> <code>slsc_cfg_set_field_limits_scan_device</code> For <code>slsc_cfg_get_field_limits_scan_device</code>, there is: <ul style="list-style-type: none"> No corresponding Job function (<code>slsc_list_*</code>) No corresponding Control function (<code>slsc_ctrl_*</code>) On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	<code>slsc_cfg_set_field_limits_scan_device</code>

Name of the function	<code>slsc_cfg_get_field_limits_stage</code>
Purpose	Returns the current setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The working field limits of the intended positioning stage type
Function signature	<code>uint32_t slsc_cfg_get_field_limits_stage(size_t Handle, slsc_Stage Stage, double* FieldLimitsMin, double* FieldLimitsMax);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Stage See enum slsc_Stage .
	FieldLimitsMin Like FieldLimitsMin .
	FieldLimitsMax Like FieldLimitsMax .
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> Available to change the setting is: <ul style="list-style-type: none"> slsc_cfg_set_field_limits_stage For slsc_cfg_get_field_limits_stage, there is: <ul style="list-style-type: none"> No corresponding Job function (slsc_list_*) No corresponding Control function (slsc_ctrl_*) On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_cfg_set_field_limits_stage

Name of the function	<code>slsc_cfg_get_jump_time</code>																		
Purpose	<p>Calculates the duration of a jump (outside “regular” Job):</p> <ul style="list-style-type: none"> Based on the current setting of the specified syncAXIS control instance and Depending on the specified values for start dynamic and end dynamic 																		
Function signature	<pre>uint32_t slsc_cfg_get_jump_time(const size_t Handle, const double* SStart, const double* VStart, const double* AStart, const double* SEnd, const double* VEnd, const double* AEnd, double MinimalJumpTime, double* JumpTime);</pre>																		
Argument(s)	<table> <tr> <td>Handle</td><td>Handle to a syncAXIS control instance.</td></tr> <tr> <td>SStart</td><td>Pointer to an array of dimension 2. Coordinates of starting point. In mm.</td></tr> <tr> <td>VStart</td><td>Pointer to an array of dimension 2. Velocity at starting point. In mm/s.</td></tr> <tr> <td>AStart</td><td>Pointer to an array of dimension 2. Acceleration at starting point. In mm/s².</td></tr> <tr> <td>SEnd</td><td>Pointer to an array of dimension 2. Coordinates of target point. In mm.</td></tr> <tr> <td>VEnd</td><td>Pointer to an array of dimension 2. Velocity im target point. In mm/s.</td></tr> <tr> <td>AEnd</td><td>Pointer to an array of dimension 2. Acceleration im target point. In mm/s².</td></tr> <tr> <td>MinimalJumpTime</td><td>Minimum duration for the jump. Allowed values: ≥ 0. In s.</td></tr> <tr> <td>JumpTime</td><td>Returned parameter value: pointer. Duration of the jump. In s.</td></tr> </table>	Handle	Handle to a syncAXIS control instance .	SStart	Pointer to an array of dimension 2. Coordinates of starting point. In mm.	VStart	Pointer to an array of dimension 2. Velocity at starting point. In mm/s.	AStart	Pointer to an array of dimension 2. Acceleration at starting point. In mm/s ² .	SEnd	Pointer to an array of dimension 2. Coordinates of target point. In mm.	VEnd	Pointer to an array of dimension 2. Velocity im target point. In mm/s.	AEnd	Pointer to an array of dimension 2. Acceleration im target point. In mm/s ² .	MinimalJumpTime	Minimum duration for the jump. Allowed values: ≥ 0 . In s.	JumpTime	Returned parameter value: pointer. Duration of the jump. In s.
Handle	Handle to a syncAXIS control instance .																		
SStart	Pointer to an array of dimension 2. Coordinates of starting point. In mm.																		
VStart	Pointer to an array of dimension 2. Velocity at starting point. In mm/s.																		
AStart	Pointer to an array of dimension 2. Acceleration at starting point. In mm/s ² .																		
SEnd	Pointer to an array of dimension 2. Coordinates of target point. In mm.																		
VEnd	Pointer to an array of dimension 2. Velocity im target point. In mm/s.																		
AEnd	Pointer to an array of dimension 2. Acceleration im target point. In mm/s ² .																		
MinimalJumpTime	Minimum duration for the jump. Allowed values: ≥ 0 . In s.																		
JumpTime	Returned parameter value: pointer. Duration of the jump. In s.																		
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.																		
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_get_jump_time is intended for isolated use in the context of Job pre-analyses for optimization (that is, slsc_cfg_get_jump_time is not supposed to be called in the “regular” Job). See also Chapter 2.2.4 “Simulating and Improving Jobs”, page 24. Calculating the duration of a jump during Job execution and slsc_cfg_get_jump_time are identical. However, slsc_cfg_get_jump_time never communicates with the RTC6 board. slsc_cfg_get_jump_time: <ul style="list-style-type: none"> Calculates the jump duration with current setting of the specified syncAXIS control instance (like values for dynamics, <code>LaserMinOffTime</code>, <code>LaserPreTriggerTime</code>, <code>MotionDecompositionConfig</code>). Takes into account changes by Configuration functions (<code>slsc_cfg_*</code>) (for example, <code>slsc_cfg_set_jump_speed</code>) after initialization Does not consider changes by Job functions (<code>slsc_list_*</code>) (for example, <code>slsc_list_set_jump_speed</code>) during Job execution 																		

Name of the function	slsc_cfg_get_jump_time
Comment(s) (cont'd)	<ul style="list-style-type: none"> MinimalJumpTime corresponds to MinimalJumpTime from slsc_list_jump_abs_min_time. If you want to calculate the duration of a common jump commanded by slsc_list_jump_abs, you need to set the MinimalJumpTime value to 0.0. For slsc_cfg_get_jump_time there is: <ul style="list-style-type: none"> No corresponding Job function (slsc_list_*) No corresponding Control function (slsc_ctrl_*) slsc_cfg_get_jump_time is allowed in any Operation mode. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.7.0.
References	–

Name of the function	<code>slsc_cfg_get_mode</code>
Purpose	Returns the current setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The Operation mode (<code>ScannerOnly</code>, <code>StageOnly</code>, <code>ScannerAndStage</code>)
Function signature	<code>uint32_t slsc_cfg_get_mode(size_t Handle, slsc_OperationMode* Mode);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Mode Returned parameter value: pointer. See enum slsc_OperationMode .
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> The Operation mode of the syncAXIS control instance is set by slsc_cfg_set_mode. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. slsc_OperationMode Mode; // Handle: see Code example at slsc_cfg_initialize_from_file slsc_cfg_get_mode(Handle, &Mode);</pre>
Version info	Available as of syncAXIS-DLL V0.11.0.
References	slsc_cfg_set_mode

Name of the function	<code>slsc_cfg_get_operation_status</code>
Purpose	Returns the current setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The operation status ("traffic light color")
Function signature	<code>uint32_t slsc_cfg_get_operation_status(size_t Handle, slsc_OperationStatus* State);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	State Returned parameter value: pointer. See enum slsc_OperationStatus .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_get_operation_status does not provide information on the status of Jobs. For example, operation status "green" (syncAXIS control instance is running, no errors occurred) does not mean that a Job is currently in execution. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. slsc_OperationStatus State; // Handle: see Code example at slsc_cfg_initialize_from_file slsc_cfg_get_operation_status(Handle, &State);</pre>
Version info	Available as of syncAXIS-DLL V0.11.0.
References	slsc_cfg_initialize_from_file

Name of the function	<code>slsc_cfg_get_scan_device_dynamic_monitoring_level</code>
Purpose	Returns the current setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The criterion for which the scan devices are to be monitored (for example, <code>slsc_DynamicsMonitoringLevel_Velocity</code>)
Function signature	<code>uint32_t slsc_cfg_get_scan_device_dynamic_monitoring_level(size_t Handle, slsc_DynamicsMonitoringLevel* DynamicMonitoringLevel);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	DynamicMonitoringLevel Returned parameter value: pointer. See enum slsc_DynamicsMonitoringLevel .
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> Available to change the setting is: <ul style="list-style-type: none"> <code>slsc_cfg_set_scan_device_dynamic_monitoring_level</code> For <code>slsc_cfg_get_scan_device_dynamic_monitoring_level</code>, there is: <ul style="list-style-type: none"> No corresponding Job function (<code>slsc_list_*</code>) No corresponding Control function (<code>slsc_ctrl_*</code>) On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. slsc_DynamicsMonitoringLevel DynamicMonitoringLevel; // Handle: see Code example at slsc_cfg_initialize_from_file slsc_cfg_get_scan_device_dynamic_monitoring_level(Handle, &slsc_DynamicsMonitoringLevel);</pre>
Version info	Available as of syncAXIS-DLL V1.5.0.
References	<code>slsc_cfg_set_scan_device_dynamic_monitoring_level</code>

Name of the function	<code>slsc_cfg_get_simulation_setting</code>
Purpose	Returns the current setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The Simulation Setting
Function signature	<code>uint32_t slsc_cfg_get_simulation_setting(size_t Handle, slsc_SimulationSetting* SimulationSetting);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	SimulationSetting Returned parameter value: pointer. See enum slsc_SimulationSetting .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> The Simulation Setting of the syncAXIS control instance is changed by slsc_cfg_set_simulation_setting. See also Chapter 2.5 "About the syncAXIS control Simulation Mode", page 31. For slsc_cfg_get_simulation_setting, there is: <ul style="list-style-type: none"> No corresponding Job function (slsc_list_*) No corresponding Control function (slsc_ctrl_*) On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. slsc_SimulationSetting SimulationSetting; // Handle: see Code example at slsc_cfg_initialize_from_file slsc_cfg_get_simulation_setting(Handle, &SimulationSetting);</pre>
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_cfg_set_simulation_setting

Name of the function	<code>slsc_cfg_get_stage_dynamic_monitoring_level</code>
Purpose	Returns the current setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The criterion for which the positioning stages are to be monitored (for example, <code>slsc_DynamicsMonitoringLevel_Velocity</code>)
Function signature	<code>uint32_t slsc_cfg_get_stage_dynamic_monitoring_level(size_t Handle, slsc_DynamicsMonitoringLevel* DynamicMonitoringLevel);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	DynamicMonitoringLevel Returned parameter value: pointer. See enum slsc_DynamicsMonitoringLevel .
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> Available to change the setting is: <ul style="list-style-type: none"> slsc_cfg_set_stage_dynamic_monitoring_level For slsc_cfg_get_stage_dynamic_monitoring_level, there is: <ul style="list-style-type: none"> No corresponding Job function (<code>slsc_list_*</code>) No corresponding Control function (<code>slsc_ctrl_*</code>) On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. slsc_DynamicsMonitoringLevel DynamicMonitoringLevel; // Handle: see Code example at slsc_cfg_initialize_from_file slsc_cfg_get_stage_dynamic_monitoring_level(Handle, &DynamicMonitoringLevel);</pre>
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_cfg_set_stage_dynamic_monitoring_level

Name of the function	slsc_cfg_get_sync_axis_version
Purpose	Returns version info on the currently running syncAXIS-DLL.
Function signature	<code>VersionInfo slsc_cfg_get_sync_axis_version(void);</code>
Argument(s)	This function has no arguments.
Return value	See structure VersionInfo .
Comment(s)	<ul style="list-style-type: none"> On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	–

Name of the function	slsc_cfg_get_trajectory_config
Purpose	Returns the current setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The Trajectory planning configuration
Function signature	<code>uint32_t slsc_cfg_get_trajectory_config(size_t Handle, slsc_TrajectoryConfig** TrajConfig);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	TrajConfig Returned parameter value: pointer to a pointer. See structure slsc_TrajectoryConfig .
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279 .
Comment(s)	<ul style="list-style-type: none"> On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0.
References	slsc_cfg_delete_trajectory_config, slsc_cfg_set_trajectory_config

Name of the function	slsc_cfg_initialize_copy
Purpose	Initialization function: Creates a new (target-)syncAXIS control instance in simulation mode with the current configuration of the specified (source-)syncAXIS control instance (in either hardware mode or simulation mode) and assigns it a unique Handle value.
Function signature	<code>uint32_t slsc_cfg_initialize_copy(size_t* Handle, size_t OriginalHandle);</code>
Argument(s)	Handle Returned parameter value: pointer. Handle to the newly created syncAXIS control instance (which is configured for the simulation mode).
	OriginalHandle Handle to a syncAXIS control instance.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions", page 279.
Comment(s)	<ul style="list-style-type: none"> • slsc_cfg_initialize_copy and slsc_cfg_initialize_from_file are the same, except: <ul style="list-style-type: none"> – slsc_cfg_initialize_copy uses the configuration of an existing (source-)syncAXIS control instance instead of reading it from a <code>syncAXISConfig.xml</code>. – slsc_cfg_initialize_copy always builds the new (target-)syncAXIS control instance in simulation mode only. • slsc_cfg_initialize_copy is primarily designed to be used in the context of recording Module files, see Code example below: slsc_cfg_initialize_copy precedes slsc_list_begin_module to quickly create a syncAXIS control instance configured for the simulation mode (because slsc_list_begin_module is only allowed in simulation mode). The module is then replayed the "original" (source-)syncAXIS control instance. • See also Section "Functions for "Modules"", page 95. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	See Figure 27, page 67 and Chapter 11 "Appendix D: Application Note – Avoiding Buffer Underruns by Using Modules", page 347.
Version info	Available as of syncAXIS-DLL V1.3.0.
References	slsc_list_begin_module

Name of the function	slsc_cfg_initialize_from_file	
Purpose	Initialization function: Creates (by using the specified XML configuration file) a new syncAXIS control instance and assigns a unique Handle value to it.	
Function signature	uint32_t slsc_cfg_initialize_from_file(size_t* Handle, const char* XmlConfigFileName);	
Argument(s)	Handle	Returned parameter value: pointer. Handle to a syncAXIS control instance .
	XmlConfigFileName	Name of the XML configuration file. Pointer to a \0-terminated ANSI string, 1 byte per char.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.	
Comment(s)	<ul style="list-style-type: none"> ⚠ Warning! Risk of injury due to laser radiation! slsc_cfg_initialize_from_file can lead to states of the RTC6 board(s) in which the laser could emit unexpectedly! Make sure that the laser is switched off before calling slsc_cfg_initialize_from_file! slsc_cfg_initialize_from_file is not executed, if the RTC6 board specified in the XML configuration file is already acquired. Example: slsc_cfg_initialize_from_file is called a second time with the same XML configuration file (and there BySerialNumber is entered as BoardIdentificationMethod). Then, the return value indicates that Bit #06 is set (UnplausibleOrUnknownParameter). slsc_cfg_initialize_from_file fails, if the FilterBandwidth value is smaller than 0.23. Then, the return value indicates that Bit #14 is set (XmlLoadError). Among other things, slsc_cfg_initialize_from_file activates (arms) the laser control on the RTC6 board (internally the RTC control command set_laser_control is used for this purpose). Furthermore, slsc_cfg_initialize_from_file automatically executes slsc_ctrl_enable_laser among other things. That is, the laser control signals LASERON, LASER1 and LASER2 are actually already outputted on the RTC6 board. ⚠ Caution! Make sure that laser safety is ensured in the entire system. slsc_cfg_initialize_from_file as well as the very last Job function (slsc_list_*) <ul style="list-style-type: none"> – takes the scan head mirrors to the zero position – but leaves the position of the positioning stage unchanged For more information about the operations which take place during the syncAXIS control instance initialization, see page 26. The syncAXIS control-software package includes an XSD file (with inline comments for user assistance) and sample XML configuration files. The XSD file defines the scheme of the XML configuration file which must be provided with slsc_cfg_initialize_from_file in order to initialize a syncAXIS control instance. 	

Name of the function	<code>slsc_cfg_initialize_from_file</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> • <code>slsc_cfg_reinitialize_from_file</code> and <code>slsc_cfg_initialize_from_file</code> are the same, except: <ul style="list-style-type: none"> – <code>slsc_cfg_initialize_from_file</code> assigns a unique Handle value to a syncAXIS control instance – With <code>slsc_cfg_reinitialize_from_file</code> a Handle value is specified (which remains unchanged). • syncAXIS control-software package does not offer any possibility to check whether and which Handles are already present “out-of-the-box”. Therefore, users should insert code for managing Handles. • On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. size_t Handle = 0; const char* XmlConfigFileName = "syncAXISConfig.xml"; slsc_cfg_initialize_from_file(&Handle, XmlConfigFileName); // This Handle value must be memorized within the user program // in order to access exactly this instance again. // That is, Handle values cannot be queried "out-of-the-box".</pre>
Version info	Available as of syncAXIS-DLL V0.9.0.
References	<code>slsc_cfg_delete</code> , <code>slsc_ctrl_enable_laser</code> , <code>slsc_cfg_get_operation_status</code> , <code>slsc_cfg_reinitialize_from_file</code>

Name of the function	<code>slsc_cfg_register_callback_job_end_planned</code>
Purpose	Sets up that the specified “Callback function” is called when a “Callback event” of type “job_end_planned” occurs.
Function signature	<code>uint32_t slsc_cfg_register_callback_job_end_planned(size_t Handle, slsc_ExecTimeCallback Callback, void* Context);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Callback Function pointer to the “Callback function” (= user-supplied function complying to signature <code>slsc_ExecTimeCallback</code>).
	Context Pointer to a user-supplied object. In the function <code>Callback</code> this object can be accessed.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> As soon as an “Callback event” of type “job_end_planned” (see also Figure 12, page 43) occurs inside the syncAXIS control instance the specified “Callback function” is called automatically. See also Section “Functions for Registering “Callback Events””, page 81. <code>slsc_cfg_register_callback_job_end_planned</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime at the time of the call. It is kept until the next <code>slsc_cfg_delete</code> or <code>slsc_cfg_reinitialize_from_file</code>. In the <code>syncAXISConfig.xml</code> there are no default values for the arguments of <code>slsc_cfg_register_callback_job_end_planned</code> to initialize the syncAXIS control instance. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	See <code>slsc_cfg_register_callback_job_finished_executing</code> .
Version info	Available as of syncAXIS-DLL V0.11.0.
References	–

Name of the function	<code>slsc_cfg_register_callback_job_finished_executing</code>
Purpose	Sets up that the specified “ Callback function ” is called when a “ Callback event ” of type “ job_finished_executing ” occurs.
Function signature	<code>uint32_t slsc_cfg_register_callback_job_finished_executing(size_t Handle, slsc_ExecTimeCallback Callback, void* Context);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Callback Function pointer to the “ Callback function ” (= user-supplied function complying to signature <code>slsc_ExecTimeCallback</code>).
	Context Pointer to a user-supplied object. In the function <code>Callback</code> this object can be accessed.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> As soon as an “Callback event” of type “job_finished_executing” (see also Figure 12, page 43) occurs inside the syncAXIS control instance the specified “Callback function” is called automatically. See also Section “Functions for Registering “Callback Events””, page 81. <code>slsc_cfg_register_callback_job_finished_executing</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime at the time of the call. It is kept until the next <code>slsc_cfg_delete</code> or <code>slsc_cfg_reinitialize_from_file</code>. In the <code>syncAXISConfig.xml</code> there are <i>no</i> default values for the arguments of <code>slsc_cfg_register_callback_job_finished_executing</code> to initialize the syncAXIS control instance. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. int Counter; slsc_ExecTimeCallback Callback = [](size_t JobID, uint64_t Progress, double ExecTime, void* Context) { int* Counter = static_cast<int*>(Context); Counter++; }; // Handle: see Code example at slsc_cfg_initialize_from_file slsc_cfg_register_callback_job_finished_executing(Handle, Callback, &Counter);</pre>
Version info	Available as of syncAXIS-DLL V0.11.0 .
References	—

Name of the function	<code>slsc_cfg_register_callback_job_is_executing</code>
Purpose	Sets up that the specified “ Callback function ” is called when a “ Callback event ” of type “ job_is_executing ” occurs.
Function signature	<code>uint32_t slsc_cfg_register_callback_job_is_executing(size_t Handle, slsc_ExecTimeCallback Callback, void* Context);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
	Callback Function pointer to the “ Callback function ” (= user-supplied function complying to signature <code>slsc_ExecTimeCallback</code>).
	Context Pointer to a user-supplied object. In the function <code>Callback</code> this object can be accessed.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> As soon as an “Callback event” of type “job_is_executing” (see also Figure 12, page 43) occurs inside the syncAXIS control instance the specified “Callback function” is called automatically. See also Section “Functions for Registering “Callback Events””, page 81. <code>slsc_cfg_register_callback_job_is_executing</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime at the time of the call. It is kept until the next <code>slsc_cfg_delete</code> or <code>slsc_cfg_reinitialize_from_file</code>. In the <code>syncAXISConfig.xml</code> there are <i>no</i> default values for the arguments of <code>slsc_cfg_register_callback_job_is_executing</code> to initialize the syncAXIS control instance. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	See <code>slsc_cfg_register_callback_job_finished_executing</code> .
Version info	Available as of syncAXIS-DLL V0.11.0 .
References	–

Name of the function	<code>slsc_cfg_register_callback_job_loaded_enough</code>
Purpose	Sets up that the specified “Callback function” is called when a “Callback event” of type “ <code>job_loaded_enough</code> ” occurs.
Function signature	<code>uint32_t slsc_cfg_register_callback_job_loaded_enough(size_t Handle, slsc_JobCallback Callback, void* Context);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Callback Function pointer to the “Callback function” (= user-supplied function complying to signature <code>slsc_JobCallback</code>).
	Context Pointer to a user-supplied object. In the function <code>Callback</code> this object can be accessed.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> As soon as an “Callback event” of type “<code>job_loaded_enough</code>” (see also Figure 12, page 43) occurs inside the syncAXIS control instance the specified “Callback function” is called automatically. See also Section “Functions for Registering “Callback Events””, page 81. <code>slsc_cfg_register_callback_job_loaded_enough</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime at the time of the call. It is kept until the next <code>slsc_cfg_delete</code> or <code>slsc_cfg_reinitialize_from_file</code>. In the <code>syncAXISConfig.xml</code> there are no default values for the arguments of <code>slsc_cfg_register_callback_job_loaded_enough</code> to initialize the syncAXIS control instance. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	See <code>slsc_cfg_register_callback_job_finished_executing</code> .
Version info	Available as of syncAXIS-DLL V0.11.0.
References	–

Name of the function	<code>slsc_cfg_register_callback_job_progress_planned</code>
Purpose	Sets up that the specified “Callback function” is called when a “Callback event” of type “job_progress_planned” occurs.
Function signature	<code>uint32_t slsc_cfg_register_callback_job_progress_planned(size_t Handle, slsc_ExecTimeCallback Callback, void* Context);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Callback Function pointer to the “Callback function” (= user-supplied function complying to signature <code>slsc_ExecTimeCallback</code>).
	Context Pointer to a user-supplied object. In the function <code>Callback</code> this object can be accessed.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> As soon as an “Callback event” of type “job_progress_planned” (see also Figure 12, page 43) occurs inside the syncAXIS control instance the specified “Callback function” is called automatically. See also Section “Functions for Registering “Callback Events””, page 81. <code>slsc_cfg_register_callback_job_progress_planned</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime at the time of the call. It is kept until the next <code>slsc_cfg_delete</code> or <code>slsc_cfg_reinitialize_from_file</code>. In the <code>syncAXISConfig.xml</code> there are no default values for the arguments of <code>slsc_cfg_register_callback_job_progress_planned</code> to initialize the syncAXIS control instance. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	See <code>slsc_cfg_register_callback_job_finished_executing</code> .
Version info	Available as of syncAXIS-DLL V0.11.0.
References	–

Name of the function	<code>slsc_cfg_register_callback_job_start_planned</code>
Purpose	Sets up that the specified “ Callback function ” is called when a “ Callback event ” of type “ <code>job_start_planned</code> ” occurs.
Function signature	<code>uint32_t slsc_cfg_register_callback_job_start_planned(size_t Handle, slsc_JobCallback Callback, void* Context);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
	Callback Function pointer to the “ Callback function ” (= user-supplied function complying to signature <code>slsc_JobCallback</code>).
	Context Pointer to a user-supplied object. In the function <code>Callback</code> this object can be accessed.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> As soon as an “Callback event” of type “<code>job_start_planned</code>” (see also Figure 12, page 43) occurs inside the syncAXIS control instance the specified “Callback function” is called automatically. See also Section “Functions for Registering “Callback Events””, page 81. <code>slsc_cfg_register_callback_job_start_planned</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime at the time of the call. It is kept until the next <code>slsc_cfg_delete</code> or <code>slsc_cfg_reinitialize_from_file</code>. In the <code>syncAXISConfig.xml</code> there are <i>no</i> default values for the arguments of <code>slsc_cfg_register_callback_job_start_planned</code> to initialize the syncAXIS control instance. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	See <code>slsc_cfg_register_callback_job_finished_executing</code> .
Version info	Available as of syncAXIS-DLL V0.11.0 .
References	–

Name of the function	<code>slsc_cfg_reinitialize</code>
Purpose	Initialization function: Destroys the specified (by the Handle) syncAXIS control instance and creates it again (by using the momentary configuration settings and values that have been previously read out). In the process, the Handle value remains unchanged.
Function signature	<code>uint32_t slsc_cfg_reinitialize(size_t Handle);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • slsc_cfg_reinitialize is allowed in all Operation modes (see slsc_OperationMode). • slsc_cfg_reinitialize is not executed, if the specified syncAXIS control instance is currently executing a Job. Then, the return value indicates that Bit #03 is set (NotAllowedInExecuting). • slsc_cfg_reinitialize is not executed, if the specified Handle value does not exist. Then, the return value indicates that Bit #02 is set (NotAllowedWithoutInitialization). • Among other things, slsc_cfg_reinitialize activates (arms) the laser control on the RTC6 board (internally the RTC control command set_laser_control is used for this purpose). Furthermore, slsc_cfg_reinitialize automatically executes slsc_ctrl_enable_laser among other things. That is, the laser control signals LASERON, LASER1 and LASER2 are actually already outputted on the RTC6 board. • ⚠ Caution! Make sure that laser safety is ensured in the entire system. • slsc_cfg_reinitialize as well as the very last Job function (slsc_list_*) <ul style="list-style-type: none"> – takes the scan head mirrors to the zero position – but leaves the position of the positioning stage unchanged • For more information about the operations which take place during the syncAXIS control instance initialization, see page 26. • slsc_cfg_reinitialize, slsc_cfg_reinitialize_from_file and slsc_cfg_initialize_from_file are the same, except: <ul style="list-style-type: none"> – slsc_cfg_initialize_from_file assigns a unique Handle value to a syncAXIS control instance – With slsc_cfg_reinitialize and slsc_cfg_reinitialize_from_file a Handle value is specified (which remains unchanged). • Use case for slsc_cfg_reinitialize: The syncAXIS control instance is in an error state. The user must reinitialize to continue executing Jobs. By slsc_cfg_reinitialize, the user can simply continue to use the current configuration without having to repeat the previous configuration steps (import syncAXISConfig.xml, series of syncAXIS-DLL function calls).

Name of the function	slsc_cfg_reinitialize
Comment(s) (cont'd)	<ul style="list-style-type: none"> • See also Chapter 2.4 "About Initializing syncAXIS control-based User Programs", page 26 with Figure 3, page 27. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.4.0.
References	slsc_cfg_delete , slsc_ctrl_enable_laser , slsc_cfg_initialize_from_file

Name of the function	<code>slsc_cfg_reinitialize_from_file</code>
Purpose	Initialization function: Destroys the specified (by the Handle) syncAXIS control instance and creates it again (by using the specified <code>syncAXISConfig.xml</code>). In the process, the Handle value remains unchanged.
Function signature	<code>uint32_t slsc_cfg_reinitialize_from_file(size_t Handle, const char* XmlConfigFileName);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	XmlConfigFileName Name of the <code>syncAXISConfig.xml</code> . Pointer to a \0-terminated ANSI string, 1 byte per char.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • <code>slsc_cfg_reinitialize</code> is allowed in all Operation modes (see slsc_OperationMode). • <code>slsc_cfg_reinitialize_from_file</code> is not executed, if the specified syncAXIS control instance is currently executing a Job. Then, the return value indicates that Bit #03 is set (<code>NotAllowedInExecuting</code>). • <code>slsc_cfg_reinitialize_from_file</code> is not executed, if the specified Handle value does not exist. Then, the return value indicates that Bit #02 is set (<code>NotAllowedWithoutInitialization</code>). • Among other things, <code>slsc_cfg_reinitialize_from_file</code> activates (arms) the laser control on the RTC6 board (internally the RTC control command <code>set_laser_control</code> is used for this purpose). Furthermore, <code>slsc_cfg_reinitialize_from_file</code> automatically executes <code>slsc_ctrl_enable_laser</code> among other things. That is, the laser control signals LASERON, LASER1 and LASER2 are actually already outputted on the RTC6 board. • ⚠ Caution! Make sure that laser safety is ensured in the entire system. • <code>slsc_cfg_reinitialize_from_file</code> as well as the very last Job function (<code>slsc_list_*</code>) <ul style="list-style-type: none"> – takes the scan head mirrors to the zero position – but leaves the position of the positioning stage unchanged • For more information about the operations which take place during the syncAXIS control instance initialization, see page 26. • The syncAXIS control-software package includes an XSD file (with inline comments for user assistance) and a sample XML configuration file. The XSD file defines the scheme of the XML configuration file which must be provided with <code>slsc_cfg_reinitialize_from_file</code> in order to initialize a syncAXIS control instance. • <code>slsc_cfg_reinitialize</code>, <code>slsc_cfg_reinitialize_from_file</code> and <code>slsc_cfg_initialize_from_file</code> are the same, except: <ul style="list-style-type: none"> – <code>slsc_cfg_initialize_from_file</code> assigns a unique Handle value to a syncAXIS control instance – With <code>slsc_cfg_reinitialize</code> and <code>slsc_cfg_reinitialize_from_file</code> a Handle value is specified (which remains unchanged).

Name of the function	<code>slsc_cfg_reinitialize_from_file</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> • See also Chapter 2.4 "About Initializing syncAXIS control-based User Programs", page 26 with Figure 3, page 27. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	slsc_cfg_delete , slsc_ctrl_enable_laser , slsc_cfg_initialize_from_file , slsc_cfg_reinitialize



Name of the function	slsc_cfg_release_stage (deprecated)
Purpose	<p>Deprecated.</p> <p>Use slsc_ctrl_follow/slsc_ctrl_unfollow instead.</p> <p>The specified syncAXIS control instance temporarily releases the positioning stage. Then, it can be controlled externally (for example, by a non-syncAXIS control-based user program).</p>
Function signature	<code>uint32_t slsc_cfg_release_stage (deprecated) (size_t Handle);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.

Name of the function	slsc_cfg_release_stage (deprecated)
Comment(s)	<ul style="list-style-type: none"> After the successful creation of a syncAXIS control instance (configured in hardware mode; by slsc_cfg_initialize_from_file) the positioning stage is also acquired (besides the RTC6 board). By slsc_cfg_release_stage (deprecated), the positioning stage can be released <i>temporarily</i>. Releasing is typically performed in less than 0.02 s. The respective syncAXIS control instance does not need to be destroyed and subsequently created again (creating typically needs about 1 s). While the positioning stage is temporarily released an external user program is able to control it. The Handle value of the syncAXIS control instance is not changed by slsc_cfg_release_stage (deprecated). slsc_cfg_release_stage (deprecated) is not accepted, when a Job is currently being executed. Then, the return value indicates that Bit #03 is set (NotAllowedInExecuting). After slsc_cfg_release_stage (deprecated), all Jobs from the Job queue are deleted. After slsc_cfg_release_stage (deprecated), the syncAXIS control instance continues to exist. However, it is transiently (until slsc_cfg_acquire_stage (deprecated) has been called) no longer addressable by all functions: <ul style="list-style-type: none"> The operation status changes to "red" (see slsc_cfg_get_operation_status). Job functions (slsc_list_*) are not accepted (including slsc_list_begin – therefore, no Job can be started). Then, the return value indicates that Bit #02 is set (NotAllowedWithoutInitialization). Control functions (slsc_ctrl_*) not accepted. Then, the return value indicates that Bit #02 is set (NotAllowedWithoutInitialization). However, many Configuration functions (slsc_cfg_*) are accepted (for example, slsc_cfg_delete, slsc_cfg_acquire_stage (deprecated), slsc_cfg_select_stage). Not accepted Configuration functions are: slsc_cfg_get_mode, slsc_cfg_set_list_handling_mode, slsc_cfg_set_mode, as well as all functions for registering event callbacks (cfg_register_callback_ functions, see Section "Functions for Registering "Callback Events"", page 81). slsc_cfg_release_stage (deprecated) does <i>not</i> change the present laser control configuration (that is, for example, no slsc_ctrl_disable_laser is sent off). In order to return the syncAXIS control instance to its original state, slsc_cfg_acquire_stage (deprecated) must be called. In the process, RTC6 board and positioning stage are acquired back on again. This process is typically completed in less than 0.1 s.

Name of the function	slsc_cfg_release_stage (deprecated)
Comment(s) (cont'd)	<ul style="list-style-type: none"> • slsc_cfg_release_stage (deprecated) also technically releases the RTC6 board temporarily. Upon re-acquisition by slsc_cfg_acquire_stage (deprecated), its internal status must not have been changed. Therefore, (in contrast to the positioning stage) the RTC6 board must <i>not</i> be used by another user program (because it can change the internal status of the RTC6 board). • See also Chapter 2.12.2 “Example – Temporarily Releasing the Positioning Stage and Changing the Target Positioning Stage”, page 74. • On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. size_t Handle = 0; const char* XmlConfigFileName = "syncAXISConfig.xml"; slsc_cfg_initialize_from_file(&Handle, XmlConfigFileName); // typical duration: about 1 s // [optional] define and execute a Job slsc_cfg_release_stage (deprecated)(Handle); // typical duration: < 0.02 s // [optional] external control of stage slsc_cfg_acquire_stage (deprecated)(Handle); // typical duration: < 0.1 s // [optional] define and execute a Job // afterwards destroy instance slsc_cfg_delete(Handle);</pre>
Version info	Available as of syncAXIS-DLL V0.9.0. Deprecated as of syncAXIS-DLL V1.0.7.
References	slsc_cfg_acquire_stage (deprecated)

Name of the function	slsc_cfg_select_heuristic
Purpose	For specifying the speed reduction characteristic (DynamicReductionFunction).
Function signature	<code>uint32_t slsc_cfg_select_heuristic(size_t Handle, uint32_t HeuristicIndex);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	HeuristicIndex Index of the desired speed reduction characteristic (DynamicReductionFunction). Allowed value range: 0...(DynamicReductionFunction – 1).
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • slsc_cfg_select_heuristic requires the Operation mode "ScannerAndStage" to be active. Otherwise, the return value indicates that Bit #11 is set (NotAllowedInCurrentMode). • For HeuristicIndex value > (DynamicReductionFunction – 1) the return value indicates that Bit #06 is set (UnplausibleOrUnknownParameter). • slsc_cfg_select_heuristic changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime upon the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. • In the syncAXISConfig.xml there are default values for the arguments of slsc_cfg_select_heuristic to initialize the syncAXIS control instance: – <code><cfg:Configuration> → <cfg:MotionDecompositionConfig> → <cfg:HeuristicConfig> → <cfg:DynamicReductionFunctions> → <cfg:DynamicReductionFunction ...></code> (= the 1st DynamicReductionFunction tag; corresponds to slsc_cfg_select_heuristic(HeuristicIndex = 0)) • There is no corresponding Job function (slsc_list_*) for slsc_cfg_select_heuristic. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.4.0.
References	–

Name of the function	<code>slsc_cfg_select_stage</code>
Purpose	For specifying the target positioning stage ("positioning stage change"). As of <code>syncAXIS-DLL</code> \geq V1.2.0, <code>slsc_cfg_select_stage</code> replaces <code>slsc_cfg_select_stage_axis</code> (deprecated).
Function signature	<code>uint32_t slsc_cfg_select_stage(size_t Handle, slsc_Stage Stage, uint32_t CorrectionFileIndex);</code>
Argument(s)	Handle Handle to a <code>syncAXIS</code> control instance.
	Stage See enum <code>slsc_Stage</code> .
	CorrectionFileIndex Index of the correction file to be used (correction files are specified in the <code>syncAXISConfig.xml</code> ; see also Section "Correction File-related Functions", page 99). Allowed values: 0...3.
Return value	See Chapter 4 "Standard Return Values of the <code>syncAXIS-DLL</code> Functions", page 279.
Comment(s)	<ul style="list-style-type: none"> By <code>slsc_cfg_select_stage</code>, the positioning stage is specified which is to be moved by <code>slsc_ctrl_move_stage_abs</code>. To be able to use <code>slsc_cfg_select_stage</code> a <code>Dongle</code> must be used which allows the use of several positioning stages (standard <code>Dongle</code> not sufficient)! Otherwise, the return value indicates that Bit #31 is set (<code>InvalidOrMissingDongle</code>). <code>slsc_cfg_select_stage</code> is only possible in the Mode "Manual Positioning". <code>Jobs</code> can only be executed after the positioning stage has been re-acquired with <code>slsc_ctrl_follow</code>. The <code>Job</code> executions after a positioning stage change are executed with the positioning stage taken over at runtime, not with the positioning stage taken over at planning time. See also Chapter 2.12.2 "Example – Temporarily Releasing the Positioning Stage and Changing the Target Positioning Stage", page 74. <code>slsc_cfg_select_stage</code> changes the current configuration of the <code>syncAXIS</code> control instance. In the process, the <code>syncAXIS</code> control instance is not reinitialized. In the <code>syncAXISConfig.xml</code> there are default values for the arguments of <code>slsc_cfg_select_stage</code> to initialize the <code>syncAXIS</code> control instance. There is no corresponding Job function (<code>slsc_list_*</code>) for <code>slsc_cfg_select_stage</code>. On the permissibility of <code>syncAXIS</code> control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of <code>syncAXIS-DLL</code> V1.2.0.
References	<code>slsc_cfg_acquire_stage</code> (deprecated), <code>slsc_ctrl_follow</code> , <code>slsc_ctrl_unfollow</code>

Name of the function	<code>slsc_cfg_select_stage_axis</code> (deprecated)
Purpose	<p>Deprecated.</p> <p>Auxiliary function for specifying the target positioning stage.</p> <p>As of <code>syncAXIS-DLL</code> ≥ V1.2.0, <code>slsc_cfg_select_stage</code> is to be used!</p>
Function signature	<pre>uint32_t slsc_cfg_select_stage_axis (deprecated) (size_t Handle, uint32_t StageAxisX, uint32_t StageAxisY, uint32_t SlecEtherCATNodeID, uint32_t DriveEtherCATNodeID, uint32_t CorrectionFileIndex);</pre>
Argument(s)	<p>Handle Handle to a <code>syncAXIS</code> control instance.</p>
	<p>StageAxisX X axis of the positioning stage to be moved. The to-be-specified parameter value is going to be communicated to the customer in collaboration with ACS.</p>
	<p>StageAxisY Y axis of the positioning stage to be moved. The to-be-specified parameter value is going to be communicated to the customer in collaboration with ACS.</p>
	<p>SlecEtherCATNodeID Position of the SLEC in the EtherCAT network. The to-be-specified parameter value is going to be communicated to the customer in collaboration with ACS.</p>
	<p>DriveEtherCATNodeID Position of the drive in the EtherCAT network. The to-be-specified parameter value is going to be communicated to the customer in collaboration with ACS.</p>
	<p>CorrectionFileIndex Index of the correction file to be used (correction files are specified in the <code>syncAXISConfig.xml</code>; see also Section "Correction File-related Functions", page 99). Allowed values: 0...3.</p>
Return value	See Chapter 4 "Standard Return Values of the <code>syncAXIS-DLL</code> Functions", page 279 .
Comment(s)	<ul style="list-style-type: none"> To be able to use <code>slsc_cfg_select_stage_axis</code> (deprecated) a Dongle must be used which allows the use of several positioning stages (standard Dongle not sufficient)! Otherwise, the return value indicates that Bit #31 is set (<code>InvalidOrMissingDongle</code>). After the call of <code>slsc_cfg_select_stage_axis</code> (deprecated), <code>slsc_cfg_acquire_stage</code> (deprecated) must follow in order to apply the parameter values. See also Chapter 2.12.2 "Example – Temporarily Releasing the Positioning Stage and Changing the Target Positioning Stage", page 74. <code>slsc_cfg_select_stage_axis</code> (deprecated) changes the current configuration of the <code>syncAXIS</code> control instance. In the process, the <code>syncAXIS</code> control instance is not reinitialized. In the <code>syncAXISConfig.xml</code> there are default values for the arguments of <code>slsc_cfg_select_stage_axis</code> (deprecated) to initialize the <code>syncAXIS</code> control instance. On the permissibility of <code>syncAXIS</code> control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of <code>syncAXIS-DLL</code> V0.9.0...1.1.0.
References	<code>slsc_cfg_select_stage</code>

Name of the function	slsc_cfg_set_bandwidth
Purpose	Changes the FilterBandwidth value of the specified syncAXIS control instance .
Function signature	<code>uint32_t slsc_cfg_set_bandwidth(size_t Handle, double FilterBandwidth);</code>
Argument(s)	<p>Handle Handle to a syncAXIS control instance.</p> <p>FilterBandwidth Desired FilterBandwidth value. In Hz. Values <0.23 are not allowed. Otherwise, the return value indicates that Bit #06 is set (UnplausibleOrUnknownParameter). Typical values are between 1...3 Hz, depending on the scan head working field and the dynamic range of the positioning stage.</p>
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_set_bandwidth requires the Operation mode "ScannerAndStage" to be active. Otherwise, the return value indicates that Bit #11 is set (NotAllowedInCurrentMode). FilterBandwidth and the syncAXISConfig.xml tag FilterBandwidth correspond to each other. When initializing the syncAXIS control instance (by slsc_cfg_initialize_from_file), the FilterBandwidth value is read from the syncAXISConfig.xml. slsc_cfg_set_bandwidth changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. There is no corresponding Job function (slsc_list_*) for slsc_cfg_set_bandwidth. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.3.0.
References	–

Name of the function	<code>slsc_cfg_set_calculation_dynamics_jump_scan_device</code>
Purpose	<p>Changes the setting of the specified syncAXIS control instance for:</p> <ul style="list-style-type: none"> The maximum acceleration and jerk value of the intended scan device type. The values are used only in Trajectory planning calculations of the scan device motion – however, only for jumps but not markings
Function signature	<pre>uint32_t slsc_cfg_set_calculation_dynamics_jump_scan_device(size_t Handle, double JumpAngularAcc, double JumpAngularJerk);</pre>
Argument(s)	<p>Handle Handle to a syncAXIS control instance.</p>
	<p>JumpAngularAcc Corresponds to Acceleration under <code><cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:CalculationDynamics> → <cfg:JumpDynamics></code>.</p>
	<p>JumpAngularJerk Corresponds to Jerk under <code><cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:CalculationDynamics> → <cfg:JumpDynamics></code>.</p>
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> ⚠ Caution! syncAXIS control uses the Acceleration value = JumpAngularAcc = JumpAngularAcc to plan trajectories for the Operation modes “ScannerOnly” and “ScannerAndStage”. Make sure that the entered values are correct. ⚠ Caution! syncAXIS control uses the Jerk value = JumpAngularJerk = JumpAngularJerk to plan trajectories for the Operation modes “ScannerOnly” and “ScannerAndStage”. Make sure that the entered values are correct. slsc_cfg_set_calculation_dynamics_jump_scan_device changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. slsc_cfg_set_calculation_dynamics_jump_scan_device is allowed in Operation mode “ScannerOnly” and “ScannerAndStage”. Otherwise, the return value indicates that Bit #11 is set (NotAllowedInCurrentMode). In the <code>syncAXISConfig.xml</code> there are default values for the arguments of slsc_cfg_set_calculation_dynamics_jump_scan_device to initialize the syncAXIS control instance: <ul style="list-style-type: none"> <code><cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:CalculationDynamics> → <cfg:JumpDynamics> → <cfg:Acceleration></code> <code><cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:CalculationDynamics> → <cfg:JumpDynamics> → <cfg:Jerk></code> For slsc_cfg_set_calculation_dynamics_jump_scan_device, there is: <ul style="list-style-type: none"> A corresponding Job function (slsc_list_*) slsc_list_set_calculation_dynamics_jump_scan_device No corresponding Control function (slsc_ctrl_*) Available to query the setting is: <ul style="list-style-type: none"> slsc_cfg_get_calculation_dynamics_jump_scan_device

Name of the function	<code>slsc_cfg_set_calculation_dynamics_jump_scan_device</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.6.0.
References	slsc_cfg_get_calculation_dynamics_jump_scan_device , slsc_list_set_calculation_dynamics_jump_scan_device

Name of the function	<code>slsc_cfg_set_calculation_dynamics_mark_scan_device</code>
Purpose	<p>Changes the setting of the specified syncAXIS control instance for:</p> <ul style="list-style-type: none"> The maximum acceleration and jerk value of the intended scan device type. The values are used only in Trajectory planning calculations of the scan device motion – however, only for markings but not jumps
Function signature	<pre>uint32_t slsc_cfg_set_calculation_dynamics_mark_scan_device(size_t Handle, double MarkAngularAcc, double MarkAngularJerk);</pre>
Argument(s)	<p>Handle Handle to a syncAXIS control instance.</p>
	<p>MarkAngularAcc Corresponds to Acceleration under <code><cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:CalculationDynamics> → <cfg:MarkDynamics></code>.</p>
	<p>MarkAngularJerk Corresponds to Jerk under <code><cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:CalculationDynamics> → <cfg:MarkDynamics></code>.</p>
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> ⚠ Caution! syncAXIS control uses the Acceleration value = MarkAngularAcc = MarkAngularAcc to plan trajectories for the Operation modes “ScannerOnly” and “ScannerAndStage”. Make sure that the entered values are correct. ⚠ Caution! syncAXIS control uses the Jerk value = MarkAngularJerk = MarkAngularJerk to plan trajectories for the Operation modes “ScannerOnly” and “ScannerAndStage”. Make sure that the entered values are correct. slsc_cfg_set_calculation_dynamics_mark_scan_device changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. slsc_cfg_set_calculation_dynamics_mark_scan_device is allowed in Operation mode “ScannerOnly” and “ScannerAndStage”. Otherwise, the return value indicates that Bit #11 is set (NotAllowedInCurrentMode). In the <code>syncAXISConfig.xml</code> there are default values for the arguments of slsc_cfg_set_calculation_dynamics_mark_scan_device to initialize the syncAXIS control instance: <ul style="list-style-type: none"> <code><cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:CalculationDynamics> → <cfg:MarkDynamics> → <cfg:Acceleration></code> <code><cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:CalculationDynamics> → <cfg:MarkDynamics> → <cfg:Jerk></code> For slsc_cfg_set_calculation_dynamics_mark_scan_device, there is: <ul style="list-style-type: none"> A corresponding Job function (slsc_list_*) slsc_list_set_calculation_dynamics_mark_scan_device No corresponding Control function (slsc_ctrl_*) Available to query the setting is: <ul style="list-style-type: none"> slsc_cfg_get_calculation_dynamics_mark_scan_device

Name of the function	slsc_cfg_set_calculation_dynamics_mark_scan_device
Comment(s) (cont'd)	<ul style="list-style-type: none"> On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.6.0.
References	slsc_cfg_get_calculation_dynamics_mark_scan_device , slsc_list_set_calculation_dynamics_mark_scan_device

Name of the function	<code>slsc_cfg_set_calculation_dynamics_stage</code>
Purpose	<p>Changes the setting of the specified syncAXIS control instance for:</p> <ul style="list-style-type: none"> The maximum dynamic capabilities ("dynamic limits") of the intended positioning stage type. The values are used only in Trajectory planning calculations of the positioning stage motion
Function signature	<code>uint32_t slsc_cfg_set_calculation_dynamics_stage(size_t Handle, slsc_Stage Stage, double StageVel, double StageAcc, double StageJerk);</code>
Argument(s)	<p>Handle Handle to a syncAXIS control instance.</p>
	<p>Stage See enum slsc_Stage.</p>
	<p>StageVel Corresponds to Velocity under <code><cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage> → <cfg:CalculationDynamics></code>.</p>
	<p>StageAcc Corresponds to Acceleration under <code><cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage> → <cfg:CalculationDynamics></code>.</p>
	<p>StageJerk Corresponds to Jerk under <code><cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage> → <cfg:CalculationDynamics></code>.</p>
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> ⚠ Caution! syncAXIS control uses the values at Velocity, Acceleration and Jerk to plan trajectories for the Operation mode "StageOnly" as well as for the end motion at Job ends. Make sure that the entered values are correct. <code>slsc_cfg_set_calculation_dynamics_stage</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. <code>slsc_cfg_set_calculation_dynamics_stage</code> is allowed in Operation mode "StageOnly" and "ScannerAndStage". Otherwise, the return value indicates that Bit #11 is set (<code>NotAllowedInCurrentMode</code>). In the <code>syncAXISConfig.xml</code> there are default values for the arguments of <code>slsc_cfg_set_calculation_dynamics_stage</code> to initialize the syncAXIS control instance: <ul style="list-style-type: none"> <code><cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage> → <cfg:CalculationDynamics></code> For <code>slsc_cfg_set_calculation_dynamics_stage</code>, there is: <ul style="list-style-type: none"> No corresponding Job function (<code>slsc_list_*</code>) No corresponding Control function (<code>slsc_ctrl_*</code>) Available to query the setting is: <ul style="list-style-type: none"> <code>slsc_cfg_get_calculation_dynamics_stage</code>

Name of the function	slsc_cfg_set_calculation_dynamics_stage
Comment(s) (cont'd)	<ul style="list-style-type: none"> On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_cfg_get_calculation_dynamics_stage

Name of the function	<code>slsc_cfg_set_contour_dependent_speed_control_2d</code>
Purpose	Switch on/off the “Contour-dependent speed calculation”. Furthermore, it is configured how the syncAXIS control instance internally determines speeds along curves (“left” or “right” of the curve mid-line; distance to it). Once the “Automatic Laser Control” is activated, these results are used to correspondingly set, for example, the laser spot distances equidistant.
Function signature	<code>uint32_t slsc_cfg_set_contour_dependent_speed_control_2d(size_t Handle, int32_t Direction, double SpotRadius);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
	Direction 0: “Contour-dependent speed calculation” = off. Speeds are determined on the curve mid-line. Is also the default status after syncAXIS control instance initialization by slsc_cfg_initialize_from_file . +1: “Contour-dependent speed calculation” = on. Speeds are determined right of the curve mid-line. -1: “Contour-dependent speed calculation” = on. Speeds are determined left of the curve mid-line.
	SpotRadius Radius of the laser spot in the working plane. In mm. The value specifies how far to the right or left of the curve mid-line the speeds are determined.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_set_contour_dependent_speed_control_2d changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. In the syncAXISConfig.xml there are <i>no</i> default values for the arguments of slsc_cfg_set_contour_dependent_speed_control_2d to initialize the syncAXIS control instance. slsc_cfg_set_contour_dependent_speed_control_2d has no effect (no error is returned), if the “Automatic Laser Control” is not switched on (for example, no ActiveChannel is entered in syncAXISConfig.xml). See also Chapter 2.9.5 “About the “Contour-dependent speed calculation””, page 60.

Name of the function	slsc_cfg_set_contour_dependent_speed_control_2d
Comment(s) (cont'd)	<ul style="list-style-type: none"> The corresponding Job function (slsc_list_*) of slsc_cfg_set_contour_dependent_speed_control_2d is slsc_list_set_contour_dependent_speed_control_2d. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0.
References	slsc_list_set_contour_dependent_speed_control_2d

Name of the function	<code>slsc_cfg_set_dynamic_limits_scan_device</code>
Purpose	Changes the setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The maximum dynamic capabilities ("dynamic limits") of the intended scan device type
Function signature	<code>uint32_t slsc_cfg_set_dynamic_limits_scan_device(size_t Handle, double AngularVel, double AngularAcc, double AngularJerk);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	AngularVel Corresponds to Velocity under <cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:DynamicLimits>.
	AngularAcc Corresponds to Acceleration under <cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:DynamicLimits>.
	AngularJerk Corresponds to Jerk under <cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:DynamicLimits>.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions", page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_cfg_set_dynamic_limits_scan_device</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. <code>slsc_cfg_set_dynamic_limits_scan_device</code> is allowed in Operation mode "ScannerOnly" and "ScannerAndStage". Otherwise, the return value indicates that Bit #11 is set (NotAllowedInCurrentMode). In the <code>syncAXISConfig.xml</code> there are default values for the arguments of <code>slsc_cfg_set_dynamic_limits_scan_device</code> to initialize the syncAXIS control instance: <ul style="list-style-type: none"> <code><cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:DynamicLimits></code> For <code>slsc_cfg_set_dynamic_limits_scan_device</code>, there is: <ul style="list-style-type: none"> No corresponding Job function (<code>slsc_list_*</code>) No corresponding Control function (<code>slsc_ctrl_*</code>) Available to query the setting is: <ul style="list-style-type: none"> <code>slsc_cfg_get_dynamic_limits_scan_device</code> Exceedances automatically trigger the reaction defined in <code>DynamicViolationReaction</code> or <code>slsc_cfg_set_dynamic_violation_reaction</code>. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	<code>slsc_cfg_get_dynamic_limits_scan_device</code> , <code>slsc_cfg_set_dynamic_violation_reaction</code> , <code>slsc_cfg_set_scan_device_dynamic_monitoring_level</code>

Name of the function	<code>slsc_cfg_set_dynamic_limits_stage</code>
Purpose	Changes the setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The dynamic limits of the intended positioning stage type
Function signature	<code>uint32_t slsc_cfg_set_dynamic_limits_stage(size_t Handle, slsc_Stage Stage, double StageVel, double StageAcc, double StageJerk);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Stage See enum slsc_Stage .
	StageVel Corresponds to Velocity under <code><cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage> → <cfg:DynamicLimits></code> .
	StageAcc Corresponds to Acceleration under <code><cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage> → <cfg:DynamicLimits></code> .
	StageJerk Corresponds to Jerk under <code><cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage> → <cfg:DynamicLimits></code> .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_set_dynamic_limits_stage changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. slsc_cfg_set_dynamic_limits_stage is allowed in Operation mode "StageOnly" and "ScannerAndStage". Otherwise, the return value indicates that Bit #11 is set (NotAllowedInCurrentMode). In the <code>syncAXISConfig.xml</code> there are default values for the arguments of slsc_cfg_set_dynamic_limits_stage to initialize the syncAXIS control instance: <ul style="list-style-type: none"> <code><cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage> → <cfg:DynamicLimits></code> For slsc_cfg_set_dynamic_limits_stage, there is: <ul style="list-style-type: none"> No corresponding Job function (slsc_list_*) No corresponding Control function (slsc_ctrl_*) Available to query the setting is: <ul style="list-style-type: none"> slsc_cfg_get_dynamic_limits_stage Exceedances automatically trigger the reaction defined in DynamicViolationReaction or slsc_cfg_set_dynamic_violation_reaction. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_cfg_get_dynamic_limits_stage , slsc_cfg_set_dynamic_violation_reaction , slsc_cfg_set_stage_dynamic_monitoring_level

Name of the function	slsc_cfg_set_dynamic_violation_reaction
Purpose	Changes the setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The reaction when a limit value exceedance occurs
Function signature	<code>uint32_t slsc_cfg_set_dynamic_violation_reaction(size_t Handle, slsc_DynamicViolationReaction DynamicViolationReaction);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	DynamicViolationReaction See enum slsc_DynamicViolationReaction .
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_set_dynamic_violation_reaction changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. slsc_cfg_set_dynamic_violation_reaction is allowed in Operation mode “ScannerOnly”, “StageOnly”, “ScannerAndStage”. In the syncAXISConfig.xml there are default values for the arguments of slsc_cfg_set_dynamic_violation_reaction to initialize the syncAXIS control instance: <ul style="list-style-type: none"> <code><cfg:Configuration> → <cfg:GeneralConfig> → <cfg:DynamicViolationReaction></code> For slsc_cfg_set_dynamic_violation_reaction, there is: <ul style="list-style-type: none"> No corresponding Job function (slsc_list_*) No corresponding Control function (slsc_ctrl_*) On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. // Handle: see Code example at slsc_cfg_initialize_from_file // Only Warnings slsc_cfg_set_dynamic_violation_reaction(Handle, slsc_DynamicViolationReaction_WarningOnly);</pre>
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_cfg_get_dynamic_violation_reaction

Name of the function	<code>slsc_cfg_set_field_limits_scan_device</code>
Purpose	Changes the setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The working field limits of the intended scan device type
Function signature	<code>uint32_t slsc_cfg_set_field_limits_scan_device(size_t Handle, const double* FieldLimitsMin, const double* FieldLimitsMax);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	FieldLimitsMin Pointer to an array of dimension 2. Corresponds to the Min attributes of <code><cfg:XDirection Min="..." ... /></code> and <code><cfg:YDirection Min="..." ... /></code> under <code><cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:FieldLimits></code>.
	FieldLimitsMax Pointer to an array of dimension 2. Corresponds to the Max attributes of <code><cfg:XDirection Max="..." ... /></code> and <code><cfg:YDirection Max="..." ... /></code> under <code><cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:FieldLimits></code>.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_cfg_set_field_limits_scan_device</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next <code>slsc_list_begin*</code>. It is kept until the next <code>slsc_cfg_delete</code> or <code>slsc_cfg_reinitialize_from_file</code>. <code>slsc_cfg_set_field_limits_scan_device</code> is allowed in Operation mode "ScannerOnly" and "ScannerAndStage". Otherwise, the return value indicates that Bit #11 is set (<code>NotAllowedInCurrentMode</code>). In the <code>syncAXISConfig.xml</code> there are default values for the arguments of <code>slsc_cfg_set_field_limits_scan_device</code> to initialize the syncAXIS control instance: <ul style="list-style-type: none"> <code><cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:FieldLimits></code> For <code>slsc_cfg_set_field_limits_scan_device</code>, there is: <ul style="list-style-type: none"> No corresponding Job function (<code>slsc_list_*</code>) No corresponding Control function (<code>slsc_ctrl_*</code>) Available to query the setting is: <ul style="list-style-type: none"> <code>slsc_cfg_get_field_limits_scan_device</code> Exceedances automatically trigger the reaction defined in <code>DynamicViolationReaction</code> or <code>slsc_cfg_set_dynamic_violation_reaction</code>. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. double FieldLimitsMin[2] = { -10.0, -10.0 }; double FieldLimitsMax[2] = { 10.0, 10.0 }; // Handle: see Code example bei slsc_cfg_initialize_from_file slsc_cfg_set_field_limits_scan_device(Handle, FieldLimitsMin, FieldLimitsMax);</pre>
Version info	Available as of syncAXIS-DLL V1.5.0.
References	<code>slsc_cfg_get_field_limits_scan_device</code>

Name of the function	<code>slsc_cfg_set_field_limits_stage</code>
Purpose	Changes the setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The working field limits of the intended positioning stage type
Function signature	<code>uint32_t slsc_cfg_set_field_limits_stage(size_t Handle, slsc_Stage Stage, const double* FieldLimitsMin, const double* FieldLimitsMax);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Stage See enum <code>slsc_Stage</code>.
	FieldLimitsMin Pointer to an array of dimension 2. Corresponds to the Min attributes of <code><cfg:XDirection Min="" ... /></code> and <code><cfg:YDirection Min="" ... /></code> under <code><cfg:Configuration></code> → <code><cfg:StageConfig></code> → <code><cfg:StageList></code> → <code><cfg:Stage></code> → <code><cfg:FieldLimits></code>.
	FieldLimitsMax Pointer to an array of dimension 2. Corresponds to the Max attributes of <code><cfg:XDirection Max="" ... /></code> and <code><cfg:YDirection Max="" ... /></code> under <code><cfg:Configuration></code> → <code><cfg:StageConfig></code> → <code><cfg:StageList></code> → <code><cfg:Stage></code> → <code><cfg:FieldLimits></code>.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_cfg_set_field_limits_stage</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next <code>slsc_list_begin*</code>. It is kept until the next <code>slsc_cfg_delete</code> or <code>slsc_cfg_reinitialize_from_file</code>. <code>slsc_cfg_set_field_limits_stage</code> is allowed in Operation mode “StageOnly” and “ScannerAndStage”. Otherwise, the return value indicates that Bit #11 is set (<code>NotAllowedInCurrentMode</code>). In the <code>syncAXISConfig.xml</code> there are default values for the arguments of <code>slsc_cfg_set_field_limits_stage</code> to initialize the syncAXIS control instance: <ul style="list-style-type: none"> <code><cfg:Configuration></code> → <code><cfg:StageConfig></code> → <code><cfg:StageList></code> → <code><cfg:Stage></code> → <code><cfg:FieldLimits></code> For <code>slsc_cfg_set_field_limits_stage</code>, there is: <ul style="list-style-type: none"> No corresponding Job function (<code>slsc_list_*</code>) No corresponding Control function (<code>slsc_ctrl_*</code>) Available to query the setting is: <ul style="list-style-type: none"> <code>slsc_cfg_get_field_limits_stage</code> Exceedances automatically trigger the reaction defined in <code>DynamicViolationReaction</code> or <code>slsc_cfg_set_dynamic_violation_reaction</code>. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.



Name of the function	slsc_cfg_set_field_limits_stage
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. double FieldLimitsMin[2] = { -150.0, -150.0 }; double FieldLimitsMax[2] = { 150.0, 150.0 }; // Handle: see Code example at slsc_cfg_initialize_from_file slsc_cfg_set_field_limits_stage(Handle, slsc_Stage1, FieldLimitsMin, FieldLimitsMax);</pre>
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_cfg_get_field_limits_stage

Name of the function	slsc_cfg_set_jump_speed
Purpose	Changes the setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The jump speed
Function signature	<code>uint32_t slsc_cfg_set_jump_speed(size_t Handle, double JumpSpeed);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
	JumpSpeed Speed. In mm/s.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_set_jump_speed changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. slsc_cfg_set_jump_speed is allowed in Operation mode “ScannerOnly”, “StageOnly”, “ScannerAndStage”. In the syncAXISConfig.xml there are default values for the arguments of slsc_cfg_set_jump_speed to initialize the syncAXIS control instance: – <code><cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:MarkConfig> → <cfg:JumpSpeed ...></code> Prior to the first call of slsc_cfg_set_jump_speed, the configuration parameter values in the syncAXIS control instance are possibly no longer set in the same way as defined in syncAXISConfig.xml (see slsc_cfg_initialize_from_file). The jump speed could have been changed by slsc_cfg_set_trajectory_config in the meantime. The corresponding Job function (slsc_list_*) of slsc_cfg_set_jump_speed is slsc_list_set_jump_speed. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	slsc_cfg_initialize_from_file , slsc_cfg_set_trajectory_config , slsc_list_set_jump_speed

Name of the function	<code>slsc_cfg_set_list_handling_mode</code>
Purpose	Sets the handling and the return behavior of the Job functions (<code>slsc_list_*</code>).
Function signature	<code>uint32_t slsc_cfg_set_list_handling_mode(size_t Handle, slsc_ListHandlingMode Mode, bool (*Predicate)(uint32_t));</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Mode See enum <code>slsc_ListHandlingMode</code> .
	Predicate Function signature for the user-supplied function.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> • <code>slsc_cfg_set_list_handling_mode</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime at the time of the call. It is kept until the next <code>slsc_cfg_delete</code> or <code>slsc_cfg_reinitialize_from_file</code>. • In the <code>syncAXISConfig.xml</code> there are default values for the arguments of <code>slsc_cfg_set_list_handling_mode</code> to initialize the syncAXIS control instance. • <code>slsc_cfg_set_list_handling_mode</code> changes the behavior of all Job functions (<code>slsc_list_*</code>). • The <code>Predicate</code> argument is only evaluated for <code>Mode = slsc_ListHandlingMode_RepeatWhilePredicate</code> (a Predicate function can be specified for this mode only). With <code>slsc_ListHandlingMode_ReturnAtOnce</code> and <code>slsc_ListHandlingMode_RepeatWhileBufferFull</code>, no Predicate function is necessary. • Prior to the first call of <code>slsc_cfg_set_list_handling_mode</code>, the configuration parameter values in the syncAXIS control instance are set as defined in <code>syncAXISConfig.xml</code> (see <code>slsc_cfg_initialize_from_file</code>). • On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.

Name of the function	slsc_cfg_set_list_handling_mode
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. // Example for "ReturnAtOnce" slsc_cfg_set_list_handling_mode(Handle, slsc_ListHandlingMode::slsc_ListHandlingMode_ReturnAtOnce, nullptr); // Example for "RepeatWhileBufferFull" slsc_cfg_set_list_handling_mode(Handle, slsc_ListHandlingMode::slsc_ListHandlingMode_RepeatWhileBufferFull, nullptr); // Example for "RepeatWhilePredicate" slsc_cfg_set_list_handling_mode(Handle, slsc_ListHandlingMode::slsc_ListHandlingMode_RepeatWhilePredicate, [](uint32_t RetVal) { bool Flag = (0x0010 == RetVal); if (Flag) { std::this_thread::sleep_for(std::chrono::milliseconds(1)); } return Flag; });</pre>
Version info	Available as of syncAXIS-DLL V0.11.0.
References	slsc_cfg_initialize_from_file

Name of the function	<code>slsc_cfg_set_list_handling_mode_with_context</code>
Purpose	Like <code>slsc_cfg_set_list_handling_mode</code> . But in addition, a context can be specified. Sets the handling and the return behavior of the Job functions (<code>slsc_list_*</code>).
Function signature	<code>uint32_t slsc_cfg_set_list_handling_mode_with_context(size_t Handle, slsc_ListHandlingMode Mode, bool (*Predicate)(uint32_t, void*), void* Context);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Mode See enum <code>slsc_ListHandlingMode</code> .
	Predicate Function signature for the user-supplied function.
	Context Pointer to a user-supplied object. The object can be accessed in the function <code>Predicate</code> .
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_cfg_set_list_handling_mode_with_context</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime at the time of the call. It is kept until the next <code>slsc_cfg_delete</code> or <code>slsc_cfg_reinitialize_from_file</code>. In the <code>syncAXISConfig.xml</code> there are default values for the arguments of <code>slsc_cfg_set_list_handling_mode_with_context</code> to initialize the syncAXIS control instance. <code>slsc_cfg_set_list_handling_mode_with_context</code> (just like <code>slsc_cfg_set_list_handling_mode</code>) changes the behavior of all Job functions (<code>slsc_list_*</code>). The <code>Predicate</code> argument is only evaluated for <code>Mode = slsc_ListHandlingMode_RepeatWhilePredicate</code> (a Predicate function can be specified for this mode only). With <code>slsc_ListHandlingMode_ReturnAtOnce</code> and <code>slsc_ListHandlingMode_RepeatWhileBufferFull</code> no Predicate function is necessary. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.1.0.
References	<code>slsc_cfg_set_list_handling_mode</code> , <code>slsc_cfg_initialize_from_file</code>

Name of the function	<code>slsc_cfg_set_mark_speed</code>
Purpose	Changes the setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The marking speed
Function signature	<code>uint32_t slsc_cfg_set_mark_speed(size_t Handle, double MarkSpeed);</code>
Argument(s)	<code>Handle</code> Handle to a syncAXIS control instance .
	<code>MarkSpeed</code> Speed. In mm/s.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_set_mark_speed changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. slsc_cfg_set_mark_speed is allowed in Operation mode “ScannerOnly”, “StageOnly”, “ScannerAndStage”. In the syncAXISConfig.xml there are default values for the arguments of slsc_cfg_set_mark_speed to initialize the syncAXIS control instance: <ul style="list-style-type: none"> – <code><cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:MarkConfig> → <cfg:MarkSpeed ...></code> Prior to the first call of slsc_cfg_set_mark_speed, the configuration parameter values in the syncAXIS control instance are possibly no longer set in the same way as defined in syncAXISConfig.xml (see slsc_cfg_initialize_from_file). The marking speed could have been changed by slsc_cfg_set_trajectory_config in the meantime. The corresponding Job function (slsc_list_*) of slsc_cfg_set_mark_speed is slsc_list_set_mark_speed. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	slsc_cfg_initialize_from_file , slsc_cfg_set_trajectory_config , slsc_list_set_mark_speed

Name of the function	<code>slsc_cfg_set_matrix_and_offset</code>
Purpose	Changes the setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> Target point coordinates according to a transformation matrix and an offset value
Function signature	<code>uint32_t slsc_cfg_set_matrix_and_offset(size_t Handle, const double* Matrix, const double* Offset);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
	Matrix Pointer to an array of dimension 4. Coefficients m11...m22 of a (2 × 2) transformation matrix.
	Offset Pointer to an array of dimension 2. x value and y value by which target points are moved in the working field. In mm.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_cfg_set_matrix_and_offset</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. In the <code>syncAXISConfig.xml</code> there are <i>no</i> default values for the arguments of <code>slsc_cfg_set_matrix_and_offset</code> to initialize the syncAXIS control instance. Target point coordinates of Job functions (slsc_list_*): see list bullet on page 268. <code>slsc_cfg_set_matrix_and_offset</code> calculates the new target points as slsc_list_set_matrix_and_offset, see list bullet on page 266. With suitable transformation matrix coefficients (argument <code>Matrix</code>), for example, scaling, rotating or flipping of marking patterns can be achieved. See also Section “Functions for Changing Target Point Coordinates”, page 92. The corresponding Job function (slsc_list_*) of <code>slsc_cfg_set_matrix_and_offset</code> is slsc_list_set_matrix_and_offset. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.1.0.
References	slsc_list_set_matrix_and_offset

Name of the function	<code>slsc_cfg_set_mode</code>
Purpose	Changes the setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The Operation mode (<code>ScannerOnly</code>, <code>StageOnly</code>, <code>ScannerAndStage</code>)
Function signature	<code>uint32_t slsc_cfg_set_mode(size_t Handle, slsc_OperationMode Mode);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
	Mode See enum slsc_OperationMode .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_set_mode changes the current configuration of the syncAXIS control instance. <i>In the process, the syncAXIS control instance is reinitialized!</i> The change will take effect at runtime at the time of the call. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. In the <code>syncAXISConfig.xml</code> there are default values for the arguments of slsc_cfg_set_mode to initialize the syncAXIS control instance. slsc_cfg_set_mode is not accepted, when a Job is currently being executed. Then, the return value indicates that Bit #03 is set (<code>NotAllowedInExecuting</code>). After slsc_cfg_set_mode, all Jobs from the Job queue are deleted. For the dynamic limits which are used in the various Operation modes, see enum slsc_OperationMode. In Operation mode <code>ScannerOnly</code>, the to-be-executed motions of the Job are only sent to the scan head. In Operation mode <code>StageOnly</code>, the to-be-executed motions of the Job are only sent to the positioning stage. In Operation mode <code>ScannerAndStage</code>, the to-be-executed motions of the Job are correspondingly assigned to both positioning stage and scan head ("Motion decomposition"). The Operation mode of the syncAXIS control instance is queried by slsc_cfg_get_mode. If slsc_cfg_set_mode is not called, then the Operation mode of the syncAXIS control instance remains set as specified in the <code>syncAXISConfig.xml</code> (see slsc_cfg_initialize_from_file) used for initialization. If a function (this is generally true in syncAXIS control) is not allowed due to the current Operation mode, then the return value indicates that Bit #11 is set (<code>NotAllowedInCurrentMode</code>). On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. // Handle: see Code example at slsc_cfg_initialize_from_file slsc_cfg_set_mode(Handle, ScannerOnly);</pre>
Version info	Available as of syncAXIS-DLL V0.11.0.
References	slsc_cfg_get_mode

Name of the function	slsc_cfg_set_part_displacement
Purpose	Applies a Matrix and an Offset to the set trajectory for the specified scan device (scan head). See Chapter 8.3 "About Transformations in syncAXIS control V1.2.4 and Higher" , page 332.
Function signature	<code>uint32_t slsc_cfg_set_part_displacement(size_t Handle, slsc_ScanDevice ScanDevice, const double* Matrix, const double* Offset);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
	ScanDevice See enum slsc_ScanDevice .
	Matrix Pointer to an array of dimension 4. Coefficients $m_{11}...m_{22}$ of a (2×2) transformation matrix.
	Offset Pointer to an array of dimension 2. x value and y value by which target points are moved in the working field. In mm.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_set_part_displacement changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. Optionally, in the <code>syncAXISConfig.xml</code> a "base" transformation can be setup for the specified scan device (tag <code><cfg:BasePartDisplacement></code>). Matrix and Offset of slsc_cfg_set_part_displacement are additional to this "base" transformation, see Chapter 8.3 "About Transformations in syncAXIS control V1.2.4 and Higher", page 332. slsc_cfg_set_part_displacement is designed to compensate for minor offsets and rotations. How much exactly can be compensated for is restricted by the scan device dynamics and depends heavily on the marking speed, size and shape of the marking pattern as well as the scan head working field size. The effects of Matrix and Offset are visible in the simulation result. There is no corresponding Job function (slsc_list_*) for slsc_cfg_set_matrix_and_offset. With slsc_cfg_set_part_displacement, transformations received from vision systems can be applied. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.2.0.
References	–

Name of the function	<code>slsc_cfg_set_rot_and_offset_2d</code>
Purpose	Changes the setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> Target point coordinates by an angle and an offset value
Function signature	<code>uint32_t slsc_cfg_set_rot_and_offset_2d(size_t Handle, double Angle, const double* Offset);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Angle Angle (about the origin 0,0) by which target points are rotated in the working field. In rad. Positive values: rotation is counterclockwise. Negative values: rotation is clockwise.
	Offset Pointer to an array of dimension 2. x value and y value by which target points are moved in the working field. In mm.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_cfg_set_rot_and_offset_2d</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next <code>slsc_list_begin*</code>. It is kept until the next <code>slsc_cfg_delete</code> or <code>slsc_cfg_reinitialize_from_file</code>. In the <code>syncAXISConfig.xml</code> there are <i>no</i> default values for the arguments of <code>slsc_cfg_set_rot_and_offset_2d</code> to initialize the syncAXIS control instance. Target point coordinates of Job functions (<code>slsc_list_*</code>): see list bullet on page 268. <code>slsc_cfg_set_rot_and_offset_2d</code> calculates the new target points as <code>slsc_list_set_rot_and_offset_2d</code>, see there. The corresponding Job function (<code>slsc_list_*</code>) of <code>slsc_cfg_set_rot_and_offset_2d</code> is <code>slsc_list_set_rot_and_offset_2d</code>. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0.
References	<code>slsc_list_arc_abs</code> , <code>slsc_list_circle_2d_abs</code> , <code>slsc_list_jump_abs</code> , <code>slsc_list_mark_abs</code> , <code>slsc_list_set_rot_and_offset_2d</code>

Name of the function	<code>slsc_cfg_set_scan_device_dynamic_monitoring_level</code>
Purpose	<p>Changes the setting of the specified syncAXIS control instance for:</p> <ul style="list-style-type: none"> The criterion for which the scan devices are to be monitored (for example, <code>slsc_DynamicsMonitoringLevel_Velocity</code>)
Function signature	<code>uint32_t slsc_cfg_set_scan_device_dynamic_monitoring_level(size_t Handle, slsc_DynamicsMonitoringLevel DynamicMonitoringLevel);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	DynamicMonitoringLevel See enum slsc_DynamicsMonitoringLevel .
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_cfg_set_scan_device_dynamic_monitoring_level</code> changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. <code>slsc_cfg_set_scan_device_dynamic_monitoring_level</code> is allowed in Operation mode “ScannerOnly” and “ScannerAndStage”. Otherwise, the return value indicates that Bit #11 is set (NotAllowedInCurrentMode). In the <code>syncAXISConfig.xml</code> there are default values for the arguments of <code>slsc_cfg_set_scan_device_dynamic_monitoring_level</code> to initialize the syncAXIS control instance: <ul style="list-style-type: none"> <code><cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:MonitoringLevel></code> For <code>slsc_cfg_set_scan_device_dynamic_monitoring_level</code>, there is: <ul style="list-style-type: none"> No corresponding Job function (<code>slsc_list_*</code>) No corresponding Control function (<code>slsc_ctrl_*</code>) Available to change the setting is: <ul style="list-style-type: none"> <code>slsc_cfg_get_scan_device_dynamic_monitoring_level</code> Exceedances automatically trigger the reaction defined in <code>DynamicViolationReaction</code> or <code>slsc_cfg_set_dynamic_violation_reaction</code>. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. // Handle: see Code example at slsc_cfg_initialize_from_file // Position AND velocity slsc_cfg_set_scan_device_dynamic_monitoring_level(Handle, slsc_DynamicsMonitoringLevel_Velocity);</pre>
Version info	Available as of syncAXIS-DLL V1.5.0.
References	<code>slsc_cfg_get_scan_device_dynamic_monitoring_level</code> , <code>slsc_cfg_set_dynamic_violation_reaction</code> , <code>slsc_cfg_set_dynamic_limits_scan_device</code>

Name of the function	slsc_cfg_set_simulation_setting
Purpose	Changes the setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The Simulation Setting
Function signature	<code>uint32_t slsc_cfg_set_simulation_setting(size_t Handle, slsc_SimulationSetting SimulationSetting);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	SimulationSetting See enum slsc_SimulationSetting .
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_set_simulation_setting changes the current configuration of the syncAXIS control instance. <i>In the process, the syncAXIS control instance is reinitialized!</i> The change will take effect at runtime at the time of the call. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. slsc_cfg_set_simulation_setting has no effect, if the addressed syncAXIS control instance has already the Simulation Setting. In the <code>syncAXISConfig.xml</code> there are default values for the arguments of slsc_cfg_set_simulation_setting to initialize the syncAXIS control instance: <ul style="list-style-type: none"> <code><cfg:Configuration> → <cfg:GeneralConfig> → <cfg:SimulationConfig> → <cfg:SimulationMode></code> slsc_cfg_set_simulation_setting is not accepted, when a Job is currently being executed. Then, the return value indicates that Bit #03 is set (NotAllowedInExecuting). After slsc_cfg_set_simulation_setting, all Jobs from the Job queue are deleted. The Simulation Setting of the syncAXIS control instance is queried by slsc_cfg_get_simulation_setting. See also Chapter 2.5 “About the syncAXIS control Simulation Mode”, page 31. For slsc_cfg_set_simulation_setting, there is: <ul style="list-style-type: none"> No corresponding Job function (slsc_list_*) No corresponding Control function (slsc_ctrl_*) On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. // Handle: see Code example at slsc_cfg_initialize_from_file // hardware mode // slsc_cfg_set_simulation_setting(Handle, slsc_SimulationSetting_HardwareMode); // simulation mode slsc_cfg_set_simulation_setting(Handle, slsc_SimulationSetting_SimulationMode);</pre>
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_cfg_get_simulation_setting

Name of the function	<code>slsc_cfg_set_stage_dynamic_monitoring_level</code>
Purpose	Changes the setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The criterion for which the positioning stages are to be monitored (for example, <code>slsc_DynamicsMonitoringLevel_Velocity</code>)
Function signature	<code>uint32_t slsc_cfg_set_stage_dynamic_monitoring_level(size_t Handle, slsc_DynamicsMonitoringLevel DynamicMonitoringLevel);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	DynamicMonitoringLevel See enum slsc_DynamicsMonitoringLevel .
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_set_stage_dynamic_monitoring_level changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. slsc_cfg_set_stage_dynamic_monitoring_level is allowed in Operation mode “StageOnly” and “ScannerAndStage”. Otherwise, the return value indicates that Bit #11 is set (NotAllowedInCurrentMode). In the syncAXISConfig.xml there are default values for the arguments of slsc_cfg_set_stage_dynamic_monitoring_level to initialize the syncAXIS control instance: <ul style="list-style-type: none"> <code><cfg:Configuration> → <cfg:StageConfig> → <cfg:MonitoringLevel></code> For slsc_cfg_set_stage_dynamic_monitoring_level, there is: <ul style="list-style-type: none"> No corresponding Job function (slsc_list_*) No corresponding Control function (slsc_ctrl_*) Available to change the setting is: <ul style="list-style-type: none"> slsc_cfg_get_stage_dynamic_monitoring_level Exceedances automatically trigger the reaction defined in DynamicViolationReaction or slsc_cfg_set_dynamic_violation_reaction. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. // Handle: see Code example at slsc_cfg_initialize_from_file // Position AND velocity slsc_cfg_set_stage_dynamic_monitoring_level(Handle, slsc_DynamicsMonitoringLevel_Velocity);</pre>
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_cfg_get_stage_dynamic_monitoring_level , slsc_cfg_set_dynamic_violation_reaction , slsc_cfg_set_dynamic_limits_stage

Name of the function	slsc_cfg_set_trajectory_config
Purpose	Changes the setting of the specified syncAXIS control instance for: <ul style="list-style-type: none"> The Trajectory planning configuration
Function signature	<code>uint32_t slsc_cfg_set_trajectory_config(size_t Handle, const slsc_TrajectoryConfig* TrajConfig);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
	TrajConfig See structure slsc_TrajectoryConfig .
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_set_trajectory_config changes the current configuration of the syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change will take effect at runtime as of the next slsc_list_begin*. It is kept until the next slsc_cfg_delete or slsc_cfg_reinitialize_from_file. In the syncAXISConfig.xml there are default values for the arguments of slsc_cfg_set_trajectory_config to initialize the syncAXIS control instance. To delete the trajectory configuration object (in order to avoid memory leaks) after slsc_cfg_set_trajectory_config again, slsc_cfg_delete_trajectory_config is available. Prior to the first call of slsc_cfg_set_trajectory_config, the configuration parameter values in the syncAXIS control instance are set as defined in syncAXISConfig.xml (see slsc_cfg_initialize_from_file). On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0.
References	slsc_cfg_delete_trajectory_config , slsc_cfg_get_trajectory_config , slsc_cfg_initialize_from_file

Name of the function	<code>slsc_ctrl_disable_laser</code>
Purpose	Inhibits that the laser control signals LASERON, LASER1 and LASER2 (see RTC6 Manual) are outputted at the RTC6 board.
Function signature	<code>uint32_t slsc_ctrl_disable_laser(size_t Handle);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • Prior to <code>slsc_ctrl_disable_laser</code>, slsc_cfg_initialize_from_file must have been executed. Among other things, slsc_cfg_initialize_from_file activates (arms) the laser control on the RTC6 board (internally the RTC control command <code>set_laser_control</code> is used for this purpose). • By <code>slsc_ctrl_disable_laser</code>, the laser control signals are inhibited in the specified syncAXIS control instance which have been released by slsc_cfg_initialize_from_file or slsc_ctrl_enable_laser before. • The behavior of <code>slsc_ctrl_disable_laser</code> and the RTC6 control command <code>disable_laser</code> correspond (see RTC6 Manual on suppressed ports, port values etc.). However, <code>slsc_ctrl_disable_laser</code> has a return value. • <code>slsc_ctrl_disable_laser</code> is always accepted (as with all Control functions (<code>slsc_ctrl_*</code>)), if the operation status is "green" (see slsc_cfg_get_operation_status). • For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	slsc_ctrl_enable_laser , slsc_cfg_get_operation_status , slsc_cfg_initialize_from_file

Name of the function	<code>slsc_ctrl_enable_laser</code>
Purpose	Releases the laser control signals LASERON, LASER1 and LASER2 (see RTC6 Manual) at the RTC6 board.
Function signature	<code>uint32_t slsc_ctrl_enable_laser(size_t Handle);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • Prior to <code>slsc_ctrl_enable_laser</code>, slsc_cfg_initialize_from_file must have been executed. • <code>slsc_ctrl_enable_laser</code> is not allowed, when a list is currently processed. Otherwise, the return value indicates that Bit #03 is set (NotAllowedInExecuting). Other than that (and that <code>slsc_ctrl_enable_laser</code> has a return value), the behavior of <code>slsc_ctrl_enable_laser</code> and the RTC6 control command <code>enable_laser</code> correspond (see RTC6 Manual on enabled ports, port values etc.). • ⚠ Caution! Make sure that laser safety is ensured in the entire system. • For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0 .
References	slsc_cfg_initialize_from_file , slsc_ctrl_disable_laser

Name of the function	<code>slsc_ctrl_follow</code>
Purpose	To re-acquire the positioning stage after a <code>slsc_ctrl_unfollow</code> .
Function signature	<code>uint32_t slsc_ctrl_follow(size_t Handle);</code>
Argument(s)	Handle Handle to a <code>syncAXIS</code> control instance.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • <code>slsc_ctrl_follow</code> also automatically terminates the Mode "Manual Positioning" of the specified <code>syncAXIS</code> control instance, see also Chapter 2.12 "About the Mode "Manual Positioning"", page 70. • <code>slsc_ctrl_follow</code> is similar to <code>slsc_cfg_acquire_stage (deprecated)</code> but faster. Furthermore, <code>syncAXIS</code>-DLL-internal <code>Job</code> planning is not interrupted and already calculated <code>Jobs</code> are not lost. • The complementary function of <code>slsc_ctrl_follow</code> is <code>slsc_ctrl_unfollow</code>. • <code>slsc_ctrl_follow</code> is only allowed after a <code>slsc_ctrl_unfollow</code>. • <code>slsc_ctrl_follow</code> allows changing positioning stages in combination with <code>slsc_cfg_select_stage</code>. Note that with <code>syncAXIS</code>-DLL \geq V1.0.7 the currently used correction file is <i>not</i> exchanged in the process. • If there is at least one <code>Job</code> in the <code>Job queue</code>, then it is checked whether the position of the positioning stage planned for the <code>Job</code> start and the current position of the positioning stage match. With a mismatch (cause: because of <code>slsc_list_begin_absolute</code> or the positioning stage has been manually positioned), the return value indicates that Bit #12 is set (<code>InvalidPosition</code>; the positioning stage must then be moved to the correct position, see also Chapter 2.12 "About the Mode "Manual Positioning"", page 70). • If there is no <code>Job</code> in the <code>Job queue</code>, then the current positioning stage position is used for Trajectory planning of the next <code>Job</code>. • For the Operation modes (see enum <code>slsc_OperationMode</code>) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. • On the permissibility of <code>syncAXIS</code> control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of <code>syncAXIS</code> -DLL V1.0.7.
References	<code>slsc_ctrl_unfollow</code>

Name of the function	<code>slsc_ctrl_get_error</code>
Purpose	Returns information on an error that occurred (error number <code>ErrorNr</code>).
Function signature	<code>uint32_t slsc_ctrl_get_error(size_t Handle, size_t ErrorNr, uint64_t* ErrorCode, char* ErrorMessage, size_t MsgBufSize);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	ErrorNr Error number for which the information is to be queried. Allowed value range: 0...[(ErrorCount of slsc_ctrl_get_error_count) – 1].
	ErrorCode Error code. Returned parameter value: pointer. See Chapter 5 “Error Codes with slsc_ctrl_get_error, Log File and Console” , page 282.
	ErrorMsg Returned parameter value: pointer to a char array. The char array must have been allocated by users previously. Forwards additional text on the error, if available.
	MsgBufSize Character count of the char array allocated to <code>ErrorMsg</code> .
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> In order to specify a meaningful value for <code>ErrorNr</code>, the number of present errors must be determined first. Therefore, slsc_ctrl_get_error_count must be called before slsc_ctrl_get_error. The first occurred error is the oldest detected error. It has the number 0. slsc_ctrl_get_error throws an exception with: <ul style="list-style-type: none"> (ErrorCount from slsc_ctrl_get_error_count) = 0 (that is, there is no error at all) <code>ErrorNr</code> > [(ErrorCount from slsc_ctrl_get_error_count) – 1] The text message is clipped to the size of <code>MsgBufSize</code> (that is, if the text message is bigger than <code>MsgBufSize</code>, then <code>ErrorMsg</code> contains only the number of characters as defined by <code>MsgBufSize</code>). For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (slsc_ctrl_*) are allowed, see Figure 37, page 98. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0. Latest change with syncAXIS-DLL V1.1.0: data type of <code>ErrorNr</code> .
References	slsc_ctrl_get_error_count

Name of the function	slsc_ctrl_get_error_count
Purpose	Returns the number of present errors.
Function signature	<code>uint32_t slsc_ctrl_get_error_count(size_t Handle, size_t* ErrorCount);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	ErrorCount Returned parameter value: pointer. Number of present errors.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • In order to specify a meaningful value for ErrorNr (at slsc_ctrl_get_error), the number of present errors must be determined first. Therefore, slsc_ctrl_get_error_count must be called before slsc_ctrl_get_error. • The first occurred error is the oldest detected error. It has the number 0. • For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (slsc_ctrl_*) are allowed, see Figure 37, page 98. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	slsc_ctrl_get_error

Name of the function	<code>slsc_ctrl_get_exec_state</code>
Purpose	Returns the state of the Execution Layer .
Function signature	<code>uint32_t slsc_ctrl_get_exec_state(size_t Handle, slsc_ExecState* State);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	State Returned parameter value: pointer. See enum slsc_ExecState .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> The RTC6 board execution state (see enum slsc_ExecState: <code>slsc_ExecState_Idle</code>, <code>slsc_ExecState_ReadyForExecution</code>, <code>slsc_ExecState_Executing</code>, <code>slsc_ExecState_NotInitOrError</code>) does <i>not</i> give any information whether additional Job functions (<code>slsc_list_*</code>) can be submitted at the moment (for this, see slsc_ctrl_is_list_input_buffer_full). These both properties are <i>not</i> interdependent. For example, by <code>slsc_ctrl_get_exec_state</code> it can be detected whether <ul style="list-style-type: none"> an execution can be started (for this purpose, the execution state must be <code>slsc_ExecState_ReadyForExecution</code>) the RTC6 board can be released (for example, at the end of the user program) again (for this purpose, the execution state must not be <code>slsc_ExecState_Executing</code>) the operating mode of the syncAXIS control instance can be changed (for this purpose, the execution state must not be <code>slsc_ExecState_Executing</code>) For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. slsc_ExecState State; // Handle: see Code example at slsc_cfg_initialize_from_file slsc_ctrl_get_exec_state(Handle, &State)</pre>
Version info	Available as of syncAXIS-DLL V0.11.0.
References	slsc_ctrl_is_list_input_buffer_full

Name of the function	<code>slsc_ctrl_get_free_variable</code>
Purpose	Returns the current value of a free variable of the RTC6.
Function signature	<code>uint32_t slsc_ctrl_get_free_variable(size_t Handle, uint32_t Number, uint32_t* Value);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Number Number of the free variable on the RTC6 to be queried. Allowed value range: [0...7]. Only the three least significant bits are evaluated.
	Value Value of the free variable currently set on the RTC6. Returned parameter value: pointer.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> • In simulation mode, slsc_ctrl_get_free_variable, slsc_ctrl_set_free_variable, as well as slsc_list_set_free_variable have no effect. • For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (slsc_ctrl_*) are allowed, see Figure 37, page 98. • slsc_ctrl_get_free_variable is a direct implementation of the RTC6 command get_free_variable in syncAXIS control. However, get_free_variable does not provide the argument Value. • The functions for free variables (slsc_ctrl_set_free_variable, slsc_list_set_free_variable and slsc_ctrl_get_free_variable) can be used, for example, to determine and count increments (within Jobs). • For further information on free variables, refer to the RTC6 Manual, Chapter 6.9.1 “Free Variables”, page 134. • On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.1.2 .
References	slsc_ctrl_set_free_variable , slsc_list_set_free_variable

Name of the function	<code>slsc_ctrl_get_job_characteristic</code>
Purpose	Returns – for a specified Job-ID – the value of a Job characteristic ("key", see enum <code>slsc_JobCharacteristic</code>) which has been calculated by the Trajectory planning .
Function signature	<code>uint32_t slsc_ctrl_get_job_characteristic(size_t Handle, size_t JobID, slsc_JobCharacteristic Key, double* Value);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	JobID Job-ID . Is returned by <code>slsc_list_begin*</code> .
	Key See enum <code>slsc_JobCharacteristic</code> .
	Value Returned parameter value: pointer.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_ctrl_get_job_characteristic</code> requires that the status of the specified Job-ID is at least "Calculation: Finished" (see Figure 12, page 43). Otherwise, the return value indicates that Bit #06 is set (<code>UnplausibleOrUnknownParameter</code>). <code>slsc_ctrl_get_job_characteristic</code> functions even without activated file output (<code>DisableFileOutput</code>). With <code>slsc_ctrl_get_job_characteristic</code> the last 10 calculated Job-IDs can be queried. <code>slsc_ctrl_get_job_characteristic</code> is intended to evaluate the effect of parameter value permutations (in simulation mode) by algorithms (that is, to automate parameter value optimization) Only with short Job-IDs (not quantifiable more precisely; due to the prerequisite mentioned in the first list bullet), <code>slsc_ctrl_get_job_characteristic</code> can also be used to implement confirmation messages in a GUI, that is, if the Trajectory planning calculation results (for example, max control values for the positioning stage) exceed defined limits (for example, "...do you really want to execute Job..."). For the Operation modes (see enum <code>slsc_OperationMode</code>) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.

Name of the function	slsc_ctrl_get_job_characteristic
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. // What is StagePosX, StagePosY in the Job size_t Handle; // from above somewhere double Value0 = 0; // init variable for 1st key double Value1 = 0; // init variable for 2nd key size_t JobID = ; // received by list_begin() slsc_ctrl_get_job_characteristic(Handle, JobID, slsc_JobCharacteristic::slsc_JobCharacteristic_StagePosX, &Value0); slsc_ctrl_get_job_characteristic(Handle, JobID, slsc_JobCharacteristic::slsc_JobCharacteristic_StagePosY, &Value1); // Value0 contains now the max. X position calculated for the Stage // Value1 contains now the max. Y position calculated for the Stage</pre>
Version info	Available as of syncAXIS-DLL V0.11.0.
References	–

Name of the function	slsc_ctrl_get_scan_device_position
Purpose	Returns the set position or actual position of the specified scan device (scan head).
Function signature	<code>uint32_t slsc_ctrl_get_scan_device_position(size_t Handle, slsc_ScanDevice ScanDevice, slsc_PositionType Type, double* Position);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	ScanDevice See enum slsc_ScanDevice .
	Type See enum slsc_PositionType .
	Position Returned parameter value: Pointer to an array of dimension 2.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279 .
Comment(s)	<ul style="list-style-type: none"> For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (slsc_ctrl_*) are allowed, see Figure 37, page 98. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.2.0.
References	slsc_ctrl_get_stage_position

Name of the function	slsc_ctrl_get_simulation_filename
Purpose	Deprecated. For querying simulation file names with syncAXIS-DLL V1.2. As of syncAXIS-DLL \geq V1.3.0, slsc_ctrl_get_syncaxis_simulation_filename must be used!
Function signature	<code>uint32_t slsc_ctrl_get_simulation_filename(size_t Handle, size_t JobID, slsc_ScanDevice ScanDevice, char* SimulationFileName, size_t FileNameBufSize);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	JobID Job-ID . Is returned by slsc_list_begin *
	ScanDevice See enum slsc_ScanDevice . The specified scan device must be entered in syncAXISConfig.xml .
	SimulationFileName Returned parameter value: pointer to a char array. The char array must have been allocated by users previously. Forwards the simulation file name.
	FileNameBufSize Character count of the char array allocated to SimulationFileName . Must be at least 49 for single digit Job-ID s.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_ctrl_get_simulation_filename is only useful with syncAXIS-DLL V1.2 because a separate simulation file is created yet for each individual scan device. File name convention in V1.2: "Simulation_ID_<Job-ID>_<scan device>_TS_<13 digits>.txt" [TS = Time Stamp; * _<scan device>* does not exist in \leq V1.1]. Example: Simulation_ID_1_ScanDevice2_TS_1546938743472.txt. With syncAXIS-DLL \geq V1.3.0, only one simulation file is generated which contains the data of the individual scan devices.
Code example	–
Version info	Available as of syncAXIS-DLL V1.2.2...V1.2.6.
References	slsc_ctrl_get_syncaxis_simulation_filename

Name of the function	<code>slsc_ctrl_get_stage_position</code>
Purpose	Returns the set position or actual position of the positioning stage.
Function signature	<code>uint32_t slsc_ctrl_get_stage_position(size_t Handle, slsc_PositionType Type, double* Position);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Type See enum slsc_PositionType .
	Position Returned parameter value: Pointer to an array of dimension 2.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.2.0.
References	<code>slsc_ctrl_get_scan_device_position</code>

Name of the function	<code>slsc_ctrl_get_syncaxis_simulation_filename</code>
Purpose	Only in simulation mode! Returns the corresponding simulation file name for a specified Job-ID .
Function signature	<code>uint32_t slsc_ctrl_get_syncaxis_simulation_filename(size_t Handle, size_t JobID, char* SimulationFileName, size_t FileNameBufSize);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	JobID Job-ID . Is returned by slsc_list_begin* .
	SimulationFileName Returned parameter value: pointer to a char array. The char array must have been allocated by users previously. Forwards the simulation file name.
	FileNameBufSize Character count of the char array allocated to SimulationFileName . Must be at least 37 for single digit Job-ID s.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_ctrl_get_syncaxis_simulation_filename replaces slsc_ctrl_get_simulation_filename. slsc_ctrl_get_syncaxis_simulation_filename requires the simulation mode to be active. Otherwise, the return value indicates that Bit #11 is set (NotAllowedInCurrentMode). slsc_ctrl_get_syncaxis_simulation_filename requires that the status of the specified Job-ID is at least “Calculation: Finished” (see Figure 12, page 43). Otherwise, the return value indicates that Bit #06 is set (UnplausibleOrUnknownParameter). slsc_ctrl_get_syncaxis_simulation_filename requires that the file output is switched on: <code><cfg:DisableFileOutput>false</cfg:DisableFileOutput></code>. Otherwise, the return value indicates that Bit #06 is set (UnplausibleOrUnknownParameter). In syncAXIS control \geq V1.3, the following convention applies to simulation file names: “Simulation_ID_<JobID>_TS_<13 Ziffern>.txt” [TS = Time Stamp]. Example: Simulation_ID_1_TS_1546938743472.txt. With slsc_ctrl_get_syncaxis_simulation_filename the last 10 calculated Job-IDs can be queried. For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (slsc_ctrl_*) are allowed, see Figure 37, page 98. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.3.0 .
References	slsc_ctrl_get_simulation_filename

Name of the function	slsc_ctrl_get_value
Purpose	Returns the present value of the specified signals at the specified axis.
Function signature	<code>uint32_t slsc_ctrl_get_value(size_t Handle, size_t AxisIndex, slsc_MeasurementSignal Signal, double* Value);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	AxisIndex 0: Axis X of scan head 1: Axis Y of scan head 2: Axis X of positioning stage 3: Axis Y of positioning stage
	Signal See enum slsc_MeasurementSignal .
	Value Returned parameter value: pointer.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (slsc_ctrl_*) are allowed, see Figure 37, page 98. Do not use slsc_ctrl_get_value to query set positions and actual positions. For this purpose, slsc_ctrl_get_scan_device_position and slsc_ctrl_get_stage_position are available (as of syncAXIS-DLL V1.2.0). On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0. Latest change with syncAXIS-DLL V1.1.0: data type of AxisIndex, Signal.
References	slsc_ctrl_get_scan_device_position , slsc_ctrl_get_stage_position

Name of the function	slsc_ctrl_is_list_input_buffer_full
Purpose	Checks whether the syncAXIS-DLL Input buffer is full (and therefore, cannot accept an additional Job function (slsc_list_*) at the moment).
Function signature	<code>uint32_t slsc_ctrl_is_list_input_buffer_full(size_t Handle, bool* Flag);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Flag Returned parameter value: pointer. true: The Input buffer is full. false: The Input buffer is not full.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> The Input buffer has a limited intake capacity (not directly quantifiable) for Job functions (slsc_list_*). Prior to send off a Job function (slsc_list_*), slsc_ctrl_is_list_input_buffer_full can be used to check whether the Input buffer is receptive (Flag false). If a Job function (slsc_list_*) is submitted and the Input buffer is full, its return value indicates that Bit #04 is set (BUFFER_FULL). If the Input buffer is full, then a sent off Job function (slsc_list_*) is not executed. In the process, <i>no</i> exception is thrown! For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (slsc_ctrl_*) are allowed, see Figure 37, page 98. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	–

Name of the function	slsc_ctrl_laser_signal_off
Purpose	Only in Mode "Manual Positioning": Switches the laser off immediately.
Function signature	<code>uint32_t slsc_ctrl_laser_signal_off(size_t Handle);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • slsc_ctrl_laser_signal_off is intended for direct laser control in combination with slsc_ctrl_laser_signal_on. • slsc_ctrl_laser_signal_off is only accepted in Mode "Manual Positioning". Otherwise, the return value indicates that Bit #11 is set (NotAllowedInCurrentMode). • See also Chapter 2.12 "About the Mode "Manual Positioning"", page 70. • For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (slsc_ctrl_*) are allowed, see Figure 37, page 98. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.1.0.
References	slsc_ctrl_laser_signal_on

Name of the function	<code>slsc_ctrl_laser_signal_on</code>
Purpose	Only in Mode "Manual Positioning": Switches the laser on immediately.
Function signature	<code>uint32_t slsc_ctrl_laser_signal_on(size_t Handle);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • slsc_ctrl_laser_signal_on is intended for direct laser control in combination with slsc_ctrl_laser_signal_off, for example, with adjustments and alignments. • The laser is also switched off, if <ul style="list-style-type: none"> – the syncAXIS control instance is destroyed – the Operation mode is changed • slsc_ctrl_laser_signal_on is only accepted in Mode "Manual Positioning". Otherwise, the return value indicates that Bit #11 is set (NotAllowedInCurrentMode). • ⚠ Caution! Make sure that laser safety is ensured in the entire system. • See also Chapter 2.12 "About the Mode "Manual Positioning"", page 70. • For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (slsc_ctrl_*) are allowed, see Figure 37, page 98. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.1.0.
References	slsc_ctrl_laser_signal_off

Name of the function	<code>slsc_ctrl_move_scanner_abs</code>
Purpose	Only in Mode "Manual Positioning": Moves all scan devices to the specified position with jump speed (starting from the current position).
Function signature	<code>uint32_t slsc_ctrl_move_scanner_abs(size_t Handle, const double* Position);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Position Pointer to an array of dimension 2. Target position in absolute (not relative) coordinates. In mm.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_ctrl_move_scanner_abs</code> is only accepted in Mode "Manual Positioning". Otherwise, the return value indicates that Bit #11 is set (NotAllowedInCurrentMode). <code>slsc_ctrl_move_stage_abs</code> and <code>slsc_ctrl_move_scanner_abs</code> switch off the laser ("actively") before the movement starts. By <code>slsc_ctrl_move_scanner_abs</code> and <code>slsc_ctrl_move_stage_abs</code> (which are similar to the RTC6 command goto_xy), it is possible to move the galvanometer scanners in the scan head and the positioning stage independently from each other. <code>slsc_ctrl_move_scanner_abs</code> moves all scan devices to the specified position even in Multi-Head systems. See also Chapter 2.12 "About the Mode "Manual Positioning"", page 70. For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.1.0.
References	<code>slsc_ctrl_move_stage_abs</code>

Name of the function	<code>slsc_ctrl_move_stage_abs</code>
Purpose	Only in Mode “Manual Positioning”: Moves the positioning stage to the specified position with the dynamics (acceleration and jerk) set by the ACS API (starting from the current position).
Function signature	<code>uint32_t slsc_ctrl_move_stage_abs(size_t Handle, const double* Position, double Speed, double Timeout);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Position Pointer to an array of dimension 2. Target position in absolute (not relative) coordinates. In mm.
	Speed Max. velocity of the positioning stage. In mm/s.
	Timeout Time after which this function should have returned at the latest. In s.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_ctrl_move_stage_abs</code> is only accepted in Mode “Manual Positioning”. Otherwise, the return value indicates that Bit #11 is set (NotAllowedInCurrentMode). <code>slsc_ctrl_move_stage_abs</code> and <code>slsc_ctrl_move_scanner_abs</code> switch off the laser (“actively”) before the movement starts. This safety feature ensures that the laser cannot be on uncontrolled. ⚠ Caution! A moving positioning stage poses mechanical hazards. There are risks of injuries to fingers and hands from crushing. Make sure that all bystanders keep sufficient distance to the appliance during execution. By <code>slsc_ctrl_move_scanner_abs</code> and <code>slsc_ctrl_move_stage_abs</code> (which are similar to the RTC6 command <code>goto_xy</code>), it is possible to move the galvanometer scanners in the scan head and the positioning stage independently from each other. Sends an ACS point-to-point motion command for the selected positioning stage (see <code>slsc_cfg_select_stage</code>) to ACS Motion Controller via TCP/IP. See also Chapter 2.12 “About the Mode “Manual Positioning””, page 70. ⚠ Caution! Make sure that the specified <code>Speed</code> value is within the permissible range, see also Chapter 2.2 “About the SAFE Use of syncAXIS control – General Approach”, page 18. The <code>Timeout</code> value specifies after what time <code>slsc_ctrl_move_stage_abs</code> should have returned at the latest (typical <code>Timeout</code> value: several seconds). After this time has expired the return value indicates that Bit #13 is set (Timeout). For the Operation modes (see enum <code>slsc_OperationMode</code>) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.1.0.
References	<code>slsc_ctrl_move_scanner_abs</code>

Name of the function	<code>slsc_ctrl_refresh_correction_file</code>
Purpose	Immediately transfers a correction file to the RTC6 board.
Function signature	<code>uint32_t slsc_ctrl_refresh_correction_file(size_t Handle, uint32_t CorrectionFileIndex);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
	CorrectionFileIndex Index of the correction file to be transferred to the RTC6 board (correction files are specified in the syncAXISConfig.xml ; see also Section "Correction File-related Functions", page 99). Allowed values: 0...3.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions", page 279 .
Comment(s)	<ul style="list-style-type: none"> Prerequisite for <code>slsc_ctrl_refresh_correction_file</code> (just like <code>slsc_ctrl_select_correction_file</code>) is an execution state <code>slsc_ExecState_Idle = 1</code> or <code>slsc_ExecState_ReadyForExecution = 1</code> (to be queried by <code>slsc_ctrl_get_exec_state</code>). <code>slsc_ctrl_select_correction_file</code> selects a correction file which is already present on the RTC6 board. In contrast, <code>slsc_ctrl_refresh_correction_file</code> immediately transfers a correction file to the RTC6 board (without the need to destroy/recreate the syncAXIS control instance). See also Section "Correction File-related Functions", page 99. For the Operation modes (see enum <code>slsc_OperationMode</code>) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.1.0.
References	<code>slsc_ctrl_select_correction_file</code>

Name of the function	<code>slsc_ctrl_select_correction_file</code>
Purpose	To specify a correction file, which is to be used immediately.
Function signature	<code>uint32_t slsc_ctrl_select_correction_file(size_t Handle, uint32_t CorrectionFileIndex);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	CorrectionFileIndex Index of the correction file to be used (correction files are specified in the <code>syncAXISConfig.xml</code> ; see also Section “Correction File-related Functions”, page 99). Allowed values: 0...3.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279 .
Comment(s)	<ul style="list-style-type: none"> • Prerequisite for <code>slsc_ctrl_select_correction_file</code> is an execution state <code>slsc_ExecState_Idle = 1</code> or <code>slsc_ExecState_ReadyForExecution = 1</code> (to be queried by <code>slsc_ctrl_get_exec_state</code>). • See also Section “Correction File-related Functions”, page 99. • See also XML tag <code>CorrectionFilePath</code>. • For the Operation modes (see enum <code>slsc_OperationMode</code>) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. • On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.1.0.
References	–

Name of the function	<code>slsc_ctrl_set_free_variable</code>
Purpose	Sets the value of a free variable on the RTC6.
Function signature	<code>uint32_t slsc_ctrl_set_free_variable(size_t Handle, uint32_t Number, uint32_t Value);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Number Number of the free variable on the RTC6 to be set. Allowed value range: [0...7]. Only the three least significant bits are evaluated.
	Value Value of the free variable to be set on the RTC6.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • In simulation mode, <code>slsc_ctrl_get_free_variable</code>, <code>slsc_ctrl_set_free_variable</code>, as well as <code>slsc_list_set_free_variable</code> have no effect. • For the Operation modes (see enum <code>slsc_OperationMode</code>) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. • <code>slsc_ctrl_set_free_variable</code> is a direct implementation of the RTC6 command <code>set_free_variable</code> in syncAXIS control. • The functions for free variables (<code>slsc_ctrl_set_free_variable</code>, <code>slsc_list_set_free_variable</code> and <code>slsc_ctrl_get_free_variable</code>) can be used, for example, to determine and count increments (within Jobs). • For further information on free variables, refer to the RTC6 Manual, Chapter 6.9.1 "Free Variables", page 134. • The corresponding Job function (<code>slsc_list_*</code>) of <code>slsc_ctrl_set_free_variable</code> is <code>slsc_list_set_free_variable</code>. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.1.2 .
References	<code>slsc_ctrl_get_free_variable</code> , <code>slsc_list_set_free_variable</code>

Name of the function	<code>slsc_ctrl_set_laser_pulses</code>
Purpose	Defines the output period and the pulse lengths for the laser signals LASER1 and LASER2 for “laser active” operation of the RTC6 board.
Function signature	<code>uint32_t slsc_ctrl_set_laser_pulses(size_t Handle, double HalfPeriod, double PulseLength);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	HalfPeriod <i>Half</i> of the output period. In s. Allowed value range: [0...67].
	PulseLength Pulse length of the laser signals LASER1 and LASER2. In s. Allowed value range: [0...67].
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> In simulation mode, <code>slsc_ctrl_set_laser_pulses</code> and <code>slsc_list_set_laser_pulses</code> have no effect. If <code>HalfPeriod</code> and/or <code>PulseLength</code> have the value 0, no laser signals are outputted. Negative values are rejected. Then, the return value indicates that Bit #06 is set (<code>UnplausibleOrUnknownParameter</code>). With $PulseLength \geq 2 \times HalfPeriod$, the laser remains on all the time. For the Operation modes (see enum <code>slsc_OperationMode</code>) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. <code>slsc_ctrl_set_laser_pulses</code> is rather similar to the RTC6 command <code>set_laser_pulses_ctrl</code>. However, the syncAXIS control function <code>slsc_ctrl_set_laser_pulses</code> is not always accepted (for example, when a Job is being executed). <code>slsc_ctrl_set_laser_pulses</code> and <code>slsc_list_set_laser_pulses</code> are provided for those Sky Writings who (due to the laser they use) cannot use the “Automatic Laser Control” to achieve equidistant spot distances and instead want to influence the pulse output via <code>HalfPeriod</code> and <code>PulseLength</code>. <code>HalfPeriod</code> and <code>PulseLength</code> change what has been set during initialization (by the attributes of the same name in <code>syncAXISConfig.xml</code>-tag <code><cfg:LaserOutput Unit="s" HalfPeriod="..." PulseLength="..." /></code>). If the “Automatic Laser Control” is active and <code>SpotDistance</code> is an “ActiveChannel”, see Chapter 2.9.2 “Definition of the Channels and ActiveChannel”, page 48, then: <ul style="list-style-type: none"> <code>HalfPeriod</code> is not effective <code>PulseLength</code> is effective (that is, pulse lengths of laser signal LASER1 and LASER2 are changed) The corresponding Job function (<code>slsc_list_*</code>) of <code>slsc_ctrl_set_laser_pulses</code> is <code>slsc_list_set_laser_pulses</code>. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.2.4.
References	slsc_list_set_laser_pulses

Name of the function	<code>slsc_ctrl_start_execution</code>
Purpose	Tries to start the execution of a Job by the Execution Layer .
Function signature	<code>uint32_t slsc_ctrl_start_execution(size_t Handle);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • Prerequisite for slsc_ctrl_start_execution: the execution status must be <code>slsc_ExecState_ReadyForExecution</code> (to be queried by slsc_ctrl_get_exec_state). • slsc_ctrl_start_execution only affects the oldest Job in the Job queue, which has not been executed yet and with status "Transfer: In progress (Enough loaded = yes)". See Figure 13, page 44. • It is immediately (!) tried to start the execution. However, the time to the actual execution start cannot be quantified. • ⚠ Caution! Make sure that laser safety is ensured in the entire system. • ⚠ Caution! A moving positioning stage poses mechanical hazards. There are risks of injuries to fingers and hands from crushing. Make sure that all bystanders keep sufficient distance to the appliance during execution. • MasterSlaveSynchronizer.exe users only, see "syncAXIS Master-Slave-Synchronizer" Manual: If several RTC6 boards have been configured for a synchronous list start by calling the <code>-ConnectExternalStartStop</code> option, then the list execution start triggered by slsc_ctrl_start_execution is also forwarded to the syncAXIS control instances on the other RTC6 boards. • For the Operation modes (see enum <code>slsc_OperationMode</code>) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	slsc_ctrl_stop , slsc_ctrl_stop_controlled

Name of the function	slsc_ctrl_stop
Purpose	Cancels the execution of the current Job uncontrolled and immediately by a direct access to the RTC6 board ("Emergency stop").
Function signature	<code>slsc_ctrl_stop(size_t Handle);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • slsc_ctrl_stop is designed to be an "emergency stop" only. slsc_ctrl_stop_controlled is supposed to be more hardware-friendly. • See Section "Comparison of slsc_ctrl_stop_controlled and slsc_ctrl_stop", page 97. • The syncAXIS control instance operation status immediately changes to "red". The syncAXIS control instance must be initialized again. • For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (slsc_ctrl_*) are allowed, see Figure 37, page 98. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	slsc_ctrl_start_execution, slsc_ctrl_stop_controlled

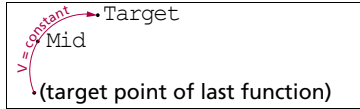
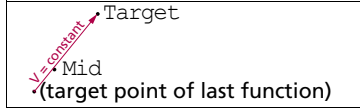
Name of the function	slsc_ctrl_stop_controlled
Purpose	Cancels the execution of the current Job controlled and inserts a compensation movement for deceleration.
Function signature	<code>slsc_ctrl_stop_controlled(size_t Handle);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • After the slsc_ctrl_stop_controlled call the RTC6 board is still processing already loaded RTC6 micro vector commands in its list memory. Therefore, it may take up to 30 s until the actual Job execution end. Afterwards the compensation movement for deceleration is executed. During this entire time, slsc_ctrl_get_exec_state returns "slsc_ExecState_Executing". • After the compensation movement, the syncAXIS control instance operation status changes to "red". The syncAXIS control instance must be initialized again. • See Section "Comparison of slsc_ctrl_stop_controlled and slsc_ctrl_stop", page 97. • For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (slsc_ctrl_*) are allowed, see Figure 37, page 98. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_ctrl_start_execution, slsc_ctrl_stop

Name of the function	slsc_ctrl_unfollow
Purpose	The specified syncAXIS control instance temporarily releases the positioning stage. Then, it can be controlled externally (for example, by a non-syncAXIS control-based user program).
Function signature	<code>uint32_t slsc_ctrl_unfollow(size_t Handle);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • slsc_ctrl_unfollow automatically puts the specified syncAXIS control instance into Mode "Manual Positioning", see also Chapter 2.12 "About the Mode "Manual Positioning"", page 70. • slsc_ctrl_unfollow is similar to slsc_cfg_release_stage (deprecated) but faster. Furthermore, syncAXIS-DLL-internal Job planning is not interrupted and already calculated Jobs are not lost. • The complementary function of slsc_ctrl_unfollow is slsc_ctrl_follow. • slsc_ctrl_unfollow must precede slsc_ctrl_follow. • For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (slsc_ctrl_*) are allowed, see Figure 37, page 98. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.0.7.
References	slsc_ctrl_follow

Name of the function	<code>slsc_ctrl_write_analog_x</code>
Purpose	Only in Mode “Manual Positioning”: Writes a output value to the 12-Bit-analog output port ANALOG OUT1 or ANALOG OUT2 of all RTC6 boards.
Function signature	<code>uint32_t slsc_ctrl_write_analog_x(size_t Handle, slsc_AnalogOutput Channel, double Value);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Channel Analog output port ANALOG OUT1 or ANALOG OUT2 (“channel”). = 1: ANALOG OUT1. = 2: ANALOG OUT2. Allowed value range: [1, 2]. See enum slsc_AnalogOutput .
	Value Output value at the ANALOG OUT1 or ANALOG OUT2 analog output port. Value = 0 corresponds to an output value of 0 V. Value = 1 corresponds to an output value of 10 V.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> The ANALOG OUT1 signal is outputted with <ul style="list-style-type: none"> RTC6 PCI Express Boards (as RTC5 boards): LASER connector, pin 08 The ANALOG OUT2 signal is outputted with <ul style="list-style-type: none"> RTC6 PCI Express Boards (as RTC5 boards): LASER connector, pin 15, as well as MARKING ON THE FLY socket connector, pin 14 slsc_ctrl_write_analog_x is not accepted, when a Job is currently being executed. Then, the return value indicates that Bit #03 is set (NotAllowedInExecuting). For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (slsc_ctrl_*) are allowed, see Figure 37, page 98. Related RTC6 command: write_da_x. The corresponding Job function (slsc_list_*) of slsc_ctrl_write_analog_x is slsc_list_write_analog_x. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.2.0.
References	slsc_list_write_analog_x

Name of the function	<code>slsc_ctrl_write_digital_out</code>
Purpose	Only in Mode "Manual Positioning": Writes a 16-bit output value to the 16-bit digital output port DIGITAL OUT 0...DIGITAL OUT 15 of all RTC6 boards.
Function signature	<code>uint32_t slsc_ctrl_write_digital_out(size_t Handle, uint16_t Value);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Value 16-bit output value (DIGITAL OUT0...DIGITAL OUT15) at the 16-bit digital output port.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> The DIGITAL OUT 0...DIGITAL OUT 15 signal is outputted with <ul style="list-style-type: none"> – RTC6 PCI Express Boards (as RTC5 boards): EXTENSION 1 socket connector, pin 01...pin 31 (odd-numbered pins only) <code>slsc_ctrl_write_digital_out</code> is not accepted, when a Job is currently being executed. Then, the return value indicates that Bit #03 is set (NotAllowedInExecuting). For the Operation modes (see enum slsc_OperationMode) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. Related RTC6 command: write_io_port. The corresponding Job function (<code>slsc_list_*</code>) of <code>slsc_ctrl_write_digital_out</code> is slsc_list_write_digital_out. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.2.0.
References	slsc_list_write_digital_out

Name of the function	<code>slsc_ctrl_write_digital_out_mask</code>
Purpose	Only in Mode “Manual Positioning”! Writes only those bits of the <code>Value</code> -values to the 16-bit digital output port of all RTC6 boards, which are specified in the user-defined bit mask (<code>Mask</code> parameter).
Function signature	<code>uint32_t slsc_ctrl_write_digital_out_mask(size_t Handle, uint16_t Value, uint16_t Mask);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Value 16-bit output value (DIGITAL OUT0 ... DIGITAL OUT15).
	Mask 16-bit mask (for DIGITAL OUT0 ... DIGITAL OUT15).
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> The DIGITAL OUT 0 ... DIGITAL OUT 15 signal is outputted with <ul style="list-style-type: none"> – RTC6 PCI Express Boards (as RTC5 boards): EXTENSION 1 socket connector, pin 01...pin 31 (odd-numbered pins only) The parameter <code>Mask</code> defines <i>which</i> bits of the 16-bit digital output port (see <code>slsc_list_write_digital_out</code>) are changed, whereas the argument <code>Value</code> defines <i>how</i> they are changed. All bits of the 16-bit digital output port which are not set in <code>Mask</code> remain unchanged. These are outputted again as previously. For <code>Mask = 0xFFFF</code> (“set all bits”), <code>slsc_ctrl_write_digital_out_mask</code> behaves like <code>slsc_ctrl_write_digital_out</code>. <code>slsc_ctrl_write_digital_out</code> is not accepted, when a Job is currently being executed. Then, the return value indicates that Bit #03 is set (<code>NotAllowedInExecuting</code>). For the Operation modes (see enum <code>slsc_OperationMode</code>) in which this and other Control functions (<code>slsc_ctrl_*</code>) are allowed, see Figure 37, page 98. Related RTC6 command: <code>write_io_port_mask</code>. The corresponding Control function (<code>slsc_list_*</code>) of <code>slsc_ctrl_write_digital_out_mask</code> is <code>slsc_list_write_digital_out_mask</code>. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.2.0.
References	<code>slsc_list_write_digital_out_mask</code>

Name of the function	slsc_list_arc_abs
Purpose	Defines a to-be-marked circular arc (not: elliptical arc) by absolute coordinate values.
Function signature	<code>uint32_t slsc_list_arc_abs(size_t Handle, const double* Mid, const double* Target);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Mid Pointer to an array of dimension 2. Coordinates of a point on the circular arc between target point of the last function and Target. In mm.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> A circular arc is defined by 3 points: <ul style="list-style-type: none"> First point = target of the last function. Mid = a point (on the circular arc) somewhere between first point and Target. Target = target point.  A straight line is marked, if the 3 points result in (nearly) lying on a single line (are collinear).  Once slsc_list_arc_abs is executed, the laser is switched on, and then the circular arc is scanned with a constant speed. <ul style="list-style-type: none"> If the marking is smaller than approx. the half scan head working field: the applied speed is the currently set marking speed. If the marking is greater than approx. the half scan head working field: the applied speed is reduced to a speed to a suiting positioning stage speed. This is specified in the characteristic (see DynamicReductionFunction). slsc_list_arc_abs belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. The corresponding slsc_list_para* function of slsc_list_arc_abs is slsc_list_para_arc_abs. slsc_cfg_set_rot_and_offset_2d and slsc_list_set_rot_and_offset_2d change the target point coordinates (argument Target and Mid, see also page 268) of slsc_list_arc_abs. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.

Name of the function	slsc_list_arc_abs
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. double Target [2]; // Array of size 2 Target[0] = 1.0; // x value Target[1] = 5.0; // y value double Mid [2]; // Array of size 2 Mid[0] = 2.0; // x value Mid[1] = 4.0; // y value // Handle: see Code example at slsc_cfg_initialize_from_file slsc_list_arc_abs(Handle, Mid, Target);</pre>
Version info	Available as of syncAXIS-DLL V0.9.0.
References	slsc_list_para_arc_abs

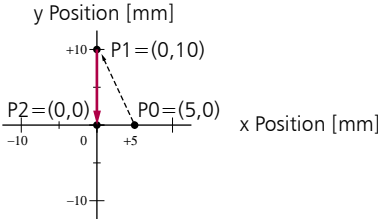
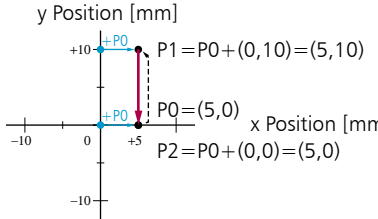
Name of the function	slsc_list_begin
Purpose	Defines the beginning of a Job . Is 1 of 2 mandatory structure elements of a Job .
Function signature	<code>uint32_t slsc_list_begin(size_t Handle, size_t* JobID);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	JobID Returned parameter value: pointer.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> On execution of slsc_list_begin the Job-ID is automatically generated by the syncAXIS control instance. It is consecutive and unique. See also Figure 13, page 44. The Job-ID is shared with event callbacks, see Section "Functions for Registering "Callback Events"", page 81. slsc_list_begin also applies any configuration changes specified with slsc_cfg_set_jump_speed and slsc_cfg_set_mark_speed. The starting point of a marking is the latest position of the positioning stage. The deflection of the scan head mirrors is (0,0) (because slsc_cfg_initialize_from_file as well as the very last Job function (slsc_list_*) positions the mirrors to (0,0). slsc_list_begin must not be followed by slsc_list_begin, slsc_list_begin_absolute or slsc_list_begin_relative. Otherwise, the return value indicates that Bit #07 is set (JobStructureNotValid). The first function of a Job must be slsc_list_begin, slsc_list_begin_absolute or slsc_list_begin_relative. Otherwise, the return value indicates that Bit #07 is set (JobStructureNotValid). On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0 .
References	slsc_cfg_initialize_from_file , slsc_cfg_set_jump_speed , slsc_cfg_set_mark_speed , slsc_list_end

Name of the function	slsc_list_begin_absolute						
Purpose	Defines (alternatively to slsc_list_begin , slsc_list_begin_relative) the beginning of a Job to compensate a position change (performed in Mode “Manual Positioning”) of the positioning stage caused by slsc_ctrl_move_stage_abs . Important: slsc_list_begin_absolute may cause the user program to crash when starting Jobs , if the end position of the preceding Job and the position specified with slsc_list_begin_absolute do not match!						
Function signature	<code>uint32_t slsc_list_begin_absolute(size_t Handle, size_t* JobID, const double* StartPosition);</code>						
Argument(s)	<table><tr><td>Handle</td><td>Handle to a syncAXIS control instance.</td></tr><tr><td>JobID</td><td>Returned parameter value: pointer.</td></tr><tr><td>StartPosition</td><td>Pointer to an array of dimension 2. Coordinates of the positioning stage: position at the time when this Job will be executed. In mm.</td></tr></table>	Handle	Handle to a syncAXIS control instance .	JobID	Returned parameter value: pointer.	StartPosition	Pointer to an array of dimension 2. Coordinates of the positioning stage: position at the time when this Job will be executed. In mm.
Handle	Handle to a syncAXIS control instance .						
JobID	Returned parameter value: pointer.						
StartPosition	Pointer to an array of dimension 2. Coordinates of the positioning stage: position at the time when this Job will be executed. In mm.						
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.						
Comment(s)	<ul style="list-style-type: none">On execution of or slsc_list_begin_absolute (just like with slsc_list_begin, slsc_list_begin_relative) the Job-ID is automatically generated by the syncAXIS control instance. It is consecutive and unique. See also Figure 13, page 44.The Job-ID is shared with event callbacks, see Section “Functions for Registering “Callback Events””, page 81.slsc_list_begin_absolute (just like slsc_list_begin, slsc_list_begin_relative) also applies any configuration changes specified with slsc_cfg_set_jump_speed and slsc_cfg_set_mark_speed.slsc_list_begin_absolute allows a Job start at a different position than the current positioning stage position or planned current positioning stage position. This allows a fast positioning stage release, moving, and reacquiring even the syncAXIS control instance is still calculating and preparing Jobs. Time saving possible, but usage is more complicated.There is a check whether the actual positioning stage position at the time of execution and the one specified at StartPosition do match. With a mismatch, the return value indicates that Bit #12 is set (InvalidPosition; the positioning stage must then be moved to the correct position, see also Chapter 2.12 “About the Mode “Manual Positioning””, page 70).slsc_list_begin_absolute must not be followed by slsc_list_begin, slsc_list_begin_absolute or slsc_list_begin_relative. Otherwise, the return value indicates that Bit #07 is set (JobStructureNotValid).The first function of a Job must be slsc_list_begin, slsc_list_begin_absolute or slsc_list_begin_relative. Otherwise, the return value indicates that Bit #07 is set (JobStructureNotValid).On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.						

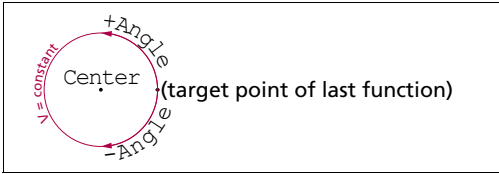


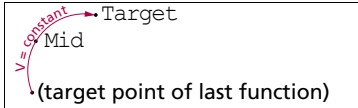
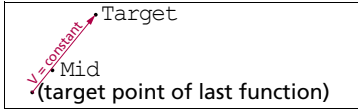
Name of the function	slsc_list_begin_absolute
Code example	–
Version info	Available as of syncAXIS-DLL V1.0.7.
References	slsc_ctrl_follow , slsc_ctrl_unfollow

Name of the function	slsc_list_begin_module
Purpose	<i>Only allowed in simulation mode.</i> To “precalculate a Job ”. Defines the beginning of a to-be-recorded Job (Module) which is closed as usual by slsc_list_end .
Function signature	<code>uint32_t slsc_list_begin_module(size_t Handle, size_t* JobID, const double* StartPosition, const char* ModuleFileName);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	JobID Returned parameter value: pointer.
	StartPosition Pointer to an array of dimension 2. Start position of the to-be-recorded Job . In mm.
	ModuleFileName Absolute file path of the generated Module file (*.slm). Pointer to a \0-terminated ANSI string, 1 byte per char.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_list_begin_module is only allowed in simulation mode. Otherwise, the return value indicates that Bit #09 is set (NotAllowedInCurrentConfiguration). Especially for this reason slsc_cfg_initialize_copy is made available. slsc_list_begin_module is allowed in Operation mode “ScannerOnly”, “StageOnly”, “ScannerAndStage”. For slsc_list_begin_module, there is no corresponding Configuration function (slsc_cfg_*). slsc_list_begin_module is similar to slsc_list_begin_absolute, therefore, see also Comment(s) there. The start position for the job needs to be specified by StartPosition. See also Section “Functions for “Modules””, page 95. For properties and content of Module files, see Section “Module file”, page 66. The recording of the Module file starts already with the call of slsc_list_begin_module. An additional function call, for example, slsc_ctrl_start_execution, is not necessary. The recording of the Module file is finished after Trajectory planning, see Figure 11, page 41. slsc_cfg_register_callback_job_end_planned can be used to determine the recording end time. A code example can be found in Figure 28, page 68. See Chapter 2.11 “About Working with “Modules””, page 65. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	See Figure 28 , page 68.
Version info	Available as of syncAXIS-DLL V1.3.0.
References	slsc_list_playback_module , slsc_cfg_initialize_copy

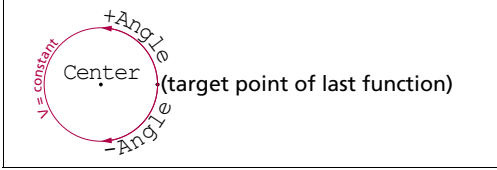
Name of the function	slsc_list_begin_relative
Purpose	<p>Defines (alternatively to slsc_list_begin) the beginning of a Job. Is 1 of 2 mandatory structure elements of a Job.</p> <p>Difference to slsc_list_begin: with ScannerAndStage and StageOnly, the position of the positioning stage (it would have at the start of the execution of this Job) is calculated. Then, it is added as offset to the coordinates in this Job (and this Job only!).</p>
Function signature	<code>uint32_t slsc_list_begin_relative(size_t Handle, size_t* JobID);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	JobID Returned parameter value: pointer.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> On execution of or slsc_list_begin_relative (just like with slsc_list_begin) the Job-ID is automatically generated by the syncAXIS control instance. It is consecutive and unique. See also Figure 13, page 44. The Job-ID is shared with event callbacks, see Section "Functions for Registering "Callback Events"", page 81. slsc_list_begin_relative (just like slsc_list_begin) also applies any configuration changes specified with slsc_cfg_set_jump_speed and slsc_cfg_set_mark_speed. Like with slsc_list_begin, the starting point of the marking is <i>also</i> the current position of the positioning stage. However, with ScannerAndStage and StageOnly, this position is added (until slsc_list_end) to the coordinates in this Job ("offset"). The deflection of the scan head mirrors is (0,0) (because slsc_cfg_initialize_from_file as well as the very last Job function (slsc_list_*) positions the mirrors to (0,0). <div style="display: flex; justify-content: space-around; align-items: flex-start;"> <div style="text-align: center;">  <pre> // Pseudo code slsc_list_begin // does NOT add initial stage pos P0 // as offset to coordinates slsc_list_jump_abs(0,10) slsc_list_mark_abs(0,0) slsc_list_end </pre> </div> <div style="text-align: center;">  <pre> // Pseudo code slsc_list_begin_relative // adds initial stage pos P0 // as offset to coordinates slsc_list_jump_abs(0,10) slsc_list_mark_abs(0,0) slsc_list_end </pre> </div> </div>

Name of the function	slsc_list_begin_relative
Comment(s) (cont'd)	<ul style="list-style-type: none"> • slsc_list_begin_relative must not be followed by slsc_list_begin, slsc_list_begin_absolute or slsc_list_begin_relative. Otherwise, the return value indicates that Bit #07 is set (JobStructureNotValid). • The first function of a Job must be slsc_list_begin, slsc_list_begin_absolute or slsc_list_begin_relative. Otherwise, the return value indicates that Bit #07 is set (JobStructureNotValid). • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.1.0.
References	slsc_list_begin

Name of the function	<code>slsc_list_circle_2d_abs</code>
Purpose	Defines a circle (not: ellipse) by the absolute coordinate value of the circle center. The parameter <code>Angle</code> determines the marking direction as well as the number of rotations (for example, $3,25 \times 2\pi$).
Function signature	<code>uint32_t slsc_list_circle_2d_abs(size_t Handle, const double* Center, double Angle);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Center Pointer to an array of dimension 2. Coordinates (x value and y value) of the circle center. In mm.
	Angle In rad. Positive values: marking is carried-out counterclockwise. Negative values: marking is carried-out clockwise.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> A circle is defined by 2 points: <ul style="list-style-type: none"> First point = target of the last Mark function or Jump command. Center = center of the circle.  Once <code>slsc_list_circle_2d_abs</code> is executed, the laser is switched on and then the circle is scanned with a constant speed. The execution direction of the marking as well as the number of rotations (for example, $3,25 \times 2\pi$) is determined by the argument <code>Angle</code>. Therefore, <code>slsc_list_circle_2d_abs</code> can be used to mark circle arcs (alternatively to <code>slsc_list_arc_abs</code>). <code>slsc_list_circle_2d_abs</code> belongs to the “MIRROR” functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 “About the Point in Time when Output Signals are actually set”, page 45. The corresponding <code>slsc_list_para*</code> function of <code>slsc_list_circle_2d_abs</code> is <code>slsc_list_para_circle_2d_abs</code>. <code>slsc_cfg_set_rot_and_offset_2d</code> and <code>slsc_list_set_rot_and_offset_2d</code> change the target point coordinates (argument <code>Center</code>, see also page 268) of <code>slsc_list_circle_2d_abs</code>. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	<code>slsc_list_arc_abs</code> , <code>slsc_list_para_circle_2d_abs</code>

Name of the function	slsc_list_dashed_arc_abs
Purpose	Like slsc_list_arc_abs , but the corresponding [*]dashed[*] Function. Therefore, offers the arguments NSwitches and LaserSwitches in addition, see Section "[*]dashed[*] Functions", page 91 .
Function signature	<code>uint32_t slsc_list_dashed_arc_abs(size_t Handle, const double* Mid, const double* Target, size_t NSwitches, const double* LaserSwitches);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Mid Pointer to an array of dimension 2. Coordinates of a point on the circular arc between target point of the last function and Target . In mm.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
	NSwitches NSwitches is the size of the LaserSwitches array. It specifies how often the laser is to be switched on/off along the marking pattern section. Minimum value: 1. See Section "[*]dashed[*] Functions", page 91.
	LaserSwitches LaserSwitches is an array of double values. The array specifies at which arc length values (in mm) a switching of "Laser Standby" Operation and "Laser Active" Operation has to occur. See Section "[*]dashed[*] Functions", page 91.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions", page 279 .
Comment(s)	<ul style="list-style-type: none"> A circular arc is defined by 3 points: <ul style="list-style-type: none"> First point = target of the last function. Mid = a point (on the circular arc) somewhere between first point and Target. Target = target point.  A straight line is marked, if the 3 points result in (nearly) lying on a single line (are collinear).  Once slsc_list_dashed_arc_abs is executed, the laser is switched on, and then the circular arc is scanned with a constant speed. <ul style="list-style-type: none"> If the marking is smaller than approx. the half scan head working field: the applied speed is the currently set marking speed. If the marking is greater than approx. the half scan head working field: the applied speed is reduced to a speed to a suiting positioning stage speed. This is specified in the characteristic (see DynamicReductionFunction). slsc_list_dashed_arc_abs belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45.

Name of the function	<code>slsc_list_dashed_arc_abs</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> The corresponding <code>slsc_list_para*</code> function of <code>slsc_list_dashed_arc_abs</code> is <code>slsc_list_para_dashed_arc_abs</code>. <code>slsc_cfg_set_rot_and_offset_2d</code> and <code>slsc_list_set_rot_and_offset_2d</code> change the target point coordinates (argument <code>Target</code> and <code>Mid</code>, see also page 268) of <code>slsc_list_dashed_arc_abs</code>. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_list_arc_abs , slsc_list_para_dashed_arc_abs

Name of the function	<code>slsc_list_dashed_circle_2d_abs</code>
Purpose	Like <code>slsc_list_circle_2d_abs</code> , but the corresponding <code>[*]dashed[*]</code> Function. Therefore, offers the arguments <code>NSwitches</code> and <code>LaserSwitches</code> in addition, see Section "[*]dashed[*] Functions", page 91 .
Function signature	<code>uint32_t slsc_list_dashed_circle_2d_abs(size_t Handle, const double* Center, double Angle, size_t NSwitches, const double* LaserSwitches);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
	Center Pointer to an array of dimension 2. Coordinates (x value and y value) of the circle center. In mm.
	Angle In rad. Positive values: marking is carried-out counterclockwise. Negative values: marking is carried-out clockwise.
	NSwitches NSwitches is the size of the LaserSwitches array. It specifies how often the laser is to be switched on/off along the marking pattern section. Minimum value: 1. See Section "[*]dashed[*] Functions", page 91 .
	LaserSwitches LaserSwitches is an array of double values. The array specifies at which arc length values (in mm) a switching of "Laser Standby" Operation and "Laser Active" Operation has to occur. See Section "[*]dashed[*] Functions", page 91 .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions", page 279 .
Comment(s)	<ul style="list-style-type: none"> A circle is defined by 2 points: <ul style="list-style-type: none"> First point = target of the last Mark function or Jump command. Center = center of the circle.  Once <code>slsc_list_dashed_circle_2d_abs</code> is executed, the laser is switched on and then the circle is scanned with a constant speed. The execution direction of the marking as well as the number of rotations (for example, $3,25 \times 2\pi$) is determined by the argument <code>Angle</code>. Therefore, <code>slsc_list_circle_2d_abs</code> can be used to mark arcs (alternatively to <code>slsc_list_dashed_arc_abs</code>). <code>slsc_list_dashed_circle_2d_abs</code> belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45.

Name of the function	<code>slsc_list_dashed_circle_2d_abs</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> The corresponding <code>slsc_list_para*</code> function of <code>slsc_list_dashed_circle_2d_abs</code> is <code>slsc_list_para_dashed_circle_2d_abs</code>. <code>slsc_cfg_set_rot_and_offset_2d</code> and <code>slsc_list_set_rot_and_offset_2d</code> change the target point coordinates (argument <code>Center</code>, see also page 268) of <code>slsc_list_dashed_circle_2d_abs</code>. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	<code>slsc_list_circle_2d_abs</code> , <code>slsc_list_para_dashed_circle_2d_abs</code>

Name of the function	<code>slsc_list_dashed_mark_abs</code>
Purpose	Like <code>slsc_list_mark_abs</code> , but the corresponding <code>[*]dashed[*]</code> Function. Therefore, offers the arguments <code>NSwitches</code> and <code>LaserSwitches</code> in addition, see Section "[*]dashed[*] Functions", page 91 .
Function signature	<code>uint32_t slsc_list_dashed_mark_abs(size_t Handle, const double* Target, size_t NSwitches, const double* LaserSwitches);</code>
Argument(s)	Handle Handle to a <code>syncAXIS</code> control instance.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
	NSwitches <code>NSwitches</code> is the size of the <code>LaserSwitches</code> array. It specifies how often the laser is to be switched on/off along the marking pattern section. Minimum value: 1. See Section "[*]dashed[*] Functions", page 91 .
	LaserSwitches <code>LaserSwitches</code> is an array of double values. The array specifies at which arc length values (in mm) a switching of "Laser Standby" Operation and "Laser Active" Operation has to occur. See Section "[*]dashed[*] Functions", page 91 .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions", page 279 .
Comment(s)	<ul style="list-style-type: none"> The behavior of mark vectors is defined in the configuration of the Trajectory planning, see <code>slsc_MarkConfig</code> (for example, <code>MarkSpeed</code>) and <code>slsc_GeometryConfig</code> (for example, <code>MaxBlendRadius</code>). <code>slsc_list_dashed_mark_abs</code> belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. The corresponding <code>slsc_list_para*</code> function of <code>slsc_list_dashed_mark_abs</code> is <code>slsc_list_para_dashed_mark_abs</code>. <code>slsc_cfg_set_rot_and_offset_2d</code> and <code>slsc_list_set_rot_and_offset_2d</code> change the target point coordinates (argument <code>Target</code>, see also page 268) of <code>slsc_list_dashed_mark_abs</code>. On the permissibility of <code>syncAXIS</code> control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of <code>syncAXIS-DLL V1.5.0</code> .
References	slsc_list_mark_abs , slsc_list_para_dashed_mark_abs

Name of the function	<code>slsc_list_end</code>
Purpose	Defines the end of a Job . Is 1 of 2 mandatory structure elements of a Job .
Function signature	<code>uint32_t slsc_list_end(size_t Handle);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • slsc_list_end terminates the Job <ul style="list-style-type: none"> – by a jump to the galvanometer scanner position 0,0 of the scan head – without changing the position of the positioning stage. That is, the positioning stage remains at the last set jump position or marking position. • To subsequently move the positioning stage to a desired position: <ul style="list-style-type: none"> – define a further Job – set the Operation mode (by slsc_cfg_set_mode) to StageOnly and – define a jump by slsc_list_jump_abs to the target position. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	slsc_cfg_set_mode , slsc_list_jump_abs , slsc_list_begin

Name of the function	slsc_list_jump_abs
Purpose	Defines a jump by absolute coordinate values.
Function signature	<code>uint32_t slsc_list_jump_abs(size_t Handle, const double* Target);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> The behavior of jumps is defined in the configuration of the Trajectory planning, see slsc_MarkConfig (for example, LaserMinOffTime and JumpSpeed). slsc_list_jump_abs belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. The corresponding slsc_list_para* function of slsc_list_jump_abs is slsc_list_para_jump_abs. slsc_cfg_set_rot_and_offset_2d and slsc_list_set_rot_and_offset_2d change the target point coordinates (argument Target, see also page 268) of slsc_list_jump_abs. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. double Target [2]; // Array of size 2 Target[0] = 2.0; x value Target[1] = 4.0; y value // Handle: see Code example at slsc_cfg_initialize_from_file slsc_list_jump_abs(Handle, Target);</pre>
Version info	Available as of syncAXIS-DLL V0.9.0.
References	slsc_list_para_jump_abs

Name of the function	slsc_list_jump_abs_min_time
Purpose	Like slsc_list_jump_abs . But additionally allows to specify a minimum duration for the jump.
Function signature	<code>uint32_t slsc_list_jump_abs_min_time(size_t Handle, const double* Target, double MinimalJumpTime);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
	MinimalJumpTime Minimum duration for the jump. In s. Only positive values are allowed.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> The behavior of jumps is defined in the configuration of the Trajectory planning, see slsc_MarkConfig (for example, LaserMinOffTime and JumpSpeed). slsc_list_jump_abs_min_time belongs to the “MIRROR” functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 “About the Point in Time when Output Signals are actually set”, page 45. The corresponding slsc_list_para* function of slsc_list_jump_abs_min_time is slsc_list_para_jump_abs_min_time. slsc_cfg_set_rot_and_offset_2d and slsc_list_set_rot_and_offset_2d change the target point coordinates (argument Target, see also page 268) of slsc_list_jump_abs_min_time. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. double Target [2]; // Array of size 2 Target[0] = 2.0; x value Target[1] = 4.0; y value // Handle: see Code example at slsc_cfg_initialize_from_file slsc_list_jump_abs_min_time(Handle, Target, 0.0001);</pre>
Version info	Available as of syncAXIS-DLL V1.6.0.
References	slsc_list_para_jump_abs_min_time

Name of the function	slsc_list_mark_abs
Purpose	Defines a mark vector by absolute coordinate values.
Function signature	<code>uint32_t slsc_list_mark_abs(size_t Handle, const double* Target);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> The behavior of mark vectors is defined in the configuration of the Trajectory planning, see slsc_MarkConfig (for example, MarkSpeed) and slsc_GeometryConfig (for example, MaxBlendRadius). slsc_list_mark_abs belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. The corresponding slsc_list_para* function of slsc_list_mark_abs is slsc_list_para_mark_abs. slsc_cfg_set_rot_and_offset_2d and slsc_list_set_rot_and_offset_2d change the target point coordinates (argument Target, see also page 268) of slsc_list_mark_abs. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. double Target [2]; // Array of size 2 Target[0] = 2.0; Target[1] = 4.0; // Handle: see Code example at slsc_cfg_initialize_from_file slsc_list_mark_abs(Handle, Target);</pre>
Version info	Available as of syncAXIS-DLL V0.9.0.
References	slsc_list_para_mark_abs , slsc_list_wait_with_laser_on

Name of the function	<code>slsc_list_multi_para_arc_abs</code>
Purpose	Like <code>slsc_list_arc_abs</code> . But offers the argument <code>MultiParaTarget</code> additionally, by which (per "ActiveChannel") a Ramp consisting of several sections is defined.
Function signature	<code>uint32_t slsc_list_multi_para_arc_abs(size_t Handle, const double* Mid, const double* Target, const slsc_MultiParaTarget* MultiParaTarget);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Mid Pointer to an array of dimension 2. Coordinates of a point on the circular arc between target point of the last function and <code>Target</code> . In mm.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
	MultiParaTarget Pointer to an array of dimension 2 or 1, see Comment(s) , page 250. 1 Ramp per "ActiveChannel" (there are at most 2 "ActiveChannel") consisting of several sections (ds), see structure slsc_MultiParaTarget .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • See <code>slsc_list_para_arc_abs</code>. • <code>slsc_list_multi_para_arc_abs</code> belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. • At the beginning of the Ramp, the achieved end value of the previous <code>slsc_list_[para/multi_para]*</code> function or (initially <code>ParaTargetDefault</code>) is used as start value (Factor Ip). • If the sum of the sections (ds) is <i>shorter</i> than the arc lasts, then the Ramp value achieved at the end of the last section value is kept unchanged until the end of the marking vector. • If the sum of the sections (ds) is <i>longer</i> than the arc lasts, then the Ramp is cancelled at this point in time. The Ramp value achieved until then is used as start value for the following Ramp. • See also Section "Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)", page 92. • See also Section "About Ramps", page 53. • <code>slsc_cfg_set_rot_and_offset_2d</code> and <code>slsc_list_set_rot_and_offset_2d</code> change the target point coordinates (argument <code>Target</code> and <code>Mid</code>, see also page 268) of <code>slsc_list_multi_para_arc_abs</code>. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	See Section "About Ramps" , page 53.
Version info	Available as of syncAXIS-DLL V0.11.0.
References	<code>slsc_list_para_arc_abs</code>

Name of the function	<code>slsc_list_multi_para_circle_2d_abs</code>
Purpose	Like <code>slsc_list_circle_2d_abs</code> . But offers the argument <code>MultiParaTarget</code> additionally, by which (per "ActiveChannel") a Ramp consisting of several sections is defined.
Function signature	<code>uint32_t slsc_list_multi_para_circle_2d_abs(size_t Handle, const double* Center, double Angle, const slsc_MultiParaTarget* MultiParaTarget);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Center Pointer to an array of dimension 2. Coordinates (x value and y value) of the circle center. In mm.
	Angle In rad. Positive values: marking is carried-out counterclockwise. Negative values: marking is carried-out clockwise.
	MultiParaTarget Pointer to an array of dimension 2 or 1, see Comment(s) , page 250. 1 Ramp per "ActiveChannel" (there are at most 2 "ActiveChannel") consisting of several sections (ds), see structure slsc_MultiParaTarget .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • See <code>slsc_list_para_circle_2d_abs</code>. • <code>slsc_list_multi_para_circle_2d_abs</code> belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. • At the beginning of the Ramp, the achieved end value of the previous <code>slsc_list_[para/multi_para]*</code> function or (initially <code>ParaTargetDefault</code>) is used as start value (Factor Ip). • If the sum of the sections (ds) is <i>shorter</i> than the total arc length ($\text{Radius} \times \text{Angle}$) lasts, then the Ramp value achieved at the end of the last section value is kept unchanged until the end of the marking vector. • If the sum of the sections (ds) is <i>longer</i> than the total arc length ($\text{Radius} \times \text{Angle}$) lasts, then the Ramp is cancelled at this point in time. The Ramp value achieved until then is used as start value for the following Ramp. • See also Section "Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)", page 92. • See also Section "About Ramps", page 53. • <code>slsc_cfg_set_rot_and_offset_2d</code> and <code>slsc_list_set_rot_and_offset_2d</code> change the target point coordinates (argument <code>Center</code>, see also page 268) of <code>slsc_list_multi_para_circle_2d_abs</code>. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0.
References	<code>slsc_list_para_circle_2d_abs</code>

Name of the function	<code>slsc_list_multi_para_dashed_arc_abs</code>
Purpose	Like <code>slsc_list_multi_para_arc_abs</code> , but the corresponding <code>[*]dashed[*]</code> Function. Therefore, offers the arguments <code>NSwitches</code> and <code>LaserSwitches</code> in addition, see Section "[*]dashed[*] Functions", page 91 .
Function signature	<code>uint32_t slsc_list_multi_para_dashed_arc_abs(size_t Handle, const double* Mid, const double* Target, size_t NSwitches, const double* LaserSwitches, const slsc_MultiParaTarget* MultiParaTarget);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Mid Pointer to an array of dimension 2. Coordinates of a point on the circular arc between target point of the last function and <code>Target</code> . In mm.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
	NSwitches NSwitches is the size of the LaserSwitches array. It specifies how often the laser is to be switched on/off along the marking pattern section. Minimum value: 1. See Section "[*]dashed[*] Functions", page 91.
	LaserSwitches LaserSwitches is an array of double values. The array specifies at which arc length values (in mm) a switching of "Laser Standby" Operation and "Laser Active" Operation has to occur. See Section "[*]dashed[*] Functions", page 91.
	MultiParaTarget Pointer to an array of dimension 2 or 1, see Comment(s), page 250 . 1 Ramp per "ActiveChannel" (there are at most 2 "ActiveChannel") consisting of several sections (<code>ds</code>), see structure <code>slsc_MultiParaTarget</code> .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions", page 279 .
Comment(s)	<ul style="list-style-type: none"> See <code>slsc_list_para_arc_abs</code>. <code>slsc_list_multi_para_dashed_arc_abs</code> belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. At the beginning of the Ramp, the achieved end value of the previous <code>slsc_list_[para/multi_para]*</code> function or (initially <code>ParaTargetDefault</code>) is used as start value (Factor Ip). If the sum of the sections (<code>ds</code>) is <i>shorter</i> than the arc lasts, then the Ramp value achieved at the end of the last section value is kept unchanged until the end of the marking vector. If the sum of the sections (<code>ds</code>) is <i>longer</i> than the arc lasts, then the Ramp is cancelled at this point in time. The Ramp value achieved until then is used as start value for the following Ramp.

Name of the function	<code>slsc_list_multi_para_dashed_arc_abs</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> • See also Section "Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)", page 92. • See also Section "About Ramps", page 53. • <code>slsc_cfg_set_rot_and_offset_2d</code> and <code>slsc_list_set_rot_and_offset_2d</code> change the target point coordinates (argument <code>Target</code> and <code>Mid</code>, see also page 268) of <code>slsc_list_multi_para_dashed_arc_abs</code>. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_list_para_arc_abs

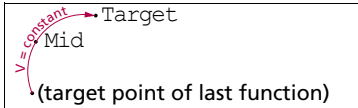
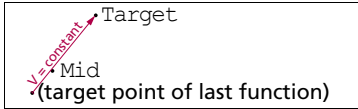
Name of the function	<code>slsc_list_multi_para_dashed_circle_2d_abs</code>
Purpose	Like <code>slsc_list_multi_para_circle_2d_abs</code> , but the corresponding <code>[*]dashed[*]</code> Function. Therefore, offers the arguments <code>NSwitches</code> and <code>LaserSwitches</code> in addition, see Section "[*]dashed[*] Functions", page 91 .
Function signature	<code>uint32_t slsc_list_multi_para_dashed_circle_2d_abs(size_t Handle, const double* Center, double Angle, size_t NSwitches, const double* LaserSwitches, const slsc_MultiParaTarget* MultiParaTarget);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Center Pointer to an array of dimension 2. Coordinates (x value and y value) of the circle center. In mm.
	Angle In rad. Positive values: marking is carried-out counterclockwise. Negative values: marking is carried-out clockwise.
	NSwitches NSwitches is the size of the LaserSwitches array. It specifies how often the laser is to be switched on/off along the marking pattern section. Minimum value: 1. See Section "[*]dashed[*] Functions", page 91.
	LaserSwitches LaserSwitches is an array of double values. The array specifies at which arc length values (in mm) a switching of "Laser Standby" Operation and "Laser Active" Operation has to occur. See Section "[*]dashed[*] Functions", page 91.
	MultiParaTarget Pointer to an array of dimension 2 or 1, see Comment(s), page 250. 1 Ramp per "ActiveChannel" (there are at most 2 "ActiveChannel") consisting of several sections (ds), see structure <code>slsc_MultiParaTarget</code>.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions", page 279 .
Comment(s)	<ul style="list-style-type: none"> See <code>slsc_list_para_circle_2d_abs</code>. <code>slsc_list_multi_para_dashed_circle_2d_abs</code> belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. At the beginning of the Ramp, the achieved end value of the previous <code>slsc_list_[para/multi_para]*</code> function or (initially <code>ParaTargetDefault</code>) is used as start value (Factor Ip). If the sum of the sections (ds) is <i>shorter</i> than the total arc length ($\text{Radius} \times \text{Angle}$) lasts, then the Ramp value achieved at the end of the last section value is kept unchanged until the end of the marking vector. If the sum of the sections (ds) is <i>longer</i> than the total arc length ($\text{Radius} \times \text{Angle}$) lasts, then the Ramp is cancelled at this point in time. The Ramp value achieved until then is used as start value for the following Ramp.

Name of the function	<code>slsc_list_multi_para_dashed_circle_2d_abs</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> • See also Section "Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)", page 92. • See also Section "About Ramps", page 53. • slsc_cfg_set_rot_and_offset_2d and slsc_list_set_rot_and_offset_2d change the target point coordinates (argument <code>Center</code>, see also page 268) of <code>slsc_list_multi_para_dashed_circle_2d_abs</code>. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_list_multi_para_circle_2d_abs

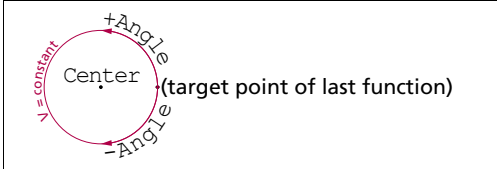
Name of the function	slsc_list_multi_para_dashed_mark_abs
Purpose	Like slsc_list_multi_para_mark_abs , but the corresponding [*]dashed[*] Function. Therefore, offers the arguments NSwitches and LaserSwitches in addition, see Section "[*]dashed[*] Functions" , page 91.
Function signature	<pre>uint32_t slsc_list_multi_para_dashed_mark_abs(size_t Handle, const double* Target, size_t NSwitches, const double* LaserSwitches, const slsc_MultiParaTarget* MultiParaTarget);</pre>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
	NSwitches NSwitches is the size of the LaserSwitches array. It specifies how often the laser is to be switched on/off along the marking pattern section. Minimum value: 1. See Section "[*]dashed[*] Functions" , page 91.
	LaserSwitches LaserSwitches is an array of double values. The array specifies at which arc length values (in mm) a switching of "Laser Standby" Operation and "Laser Active" Operation has to occur. See Section "[*]dashed[*] Functions" , page 91.
	MultiParaTarget Pointer to an array of dimension 2 or 1, see Comment(s) , page 250. 1 Ramp per "ActiveChannel" (there are at most 2 "ActiveChannel") consisting of several sections (ds), see structure slsc_MultiParaTarget .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • See slsc_list_para_mark_abs. • slsc_list_multi_para_dashed_mark_abs belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. • At the beginning of the Ramp, the achieved end value of the previous slsc_list_[para/multi_para]* function or (initially ParaTargetDefault) is used as start value (Factor Ip). • If the sum of the sections (ds) is <i>shorter</i> than the mark vector length lasts, then the Ramp value achieved at the end of the last section value is kept unchanged until the end of the marking vector. • If the sum of the sections (ds) is <i>longer</i> than the mark vector length lasts, then the Ramp is cancelled at this point in time. The Ramp value achieved until then is used as start value for the following Ramp.

Name of the function	slsc_list_multi_para_dashed_mark_abs
Comment(s) (cont'd)	<ul style="list-style-type: none"> • See also Section "Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)", page 92. • See also Section "About Ramps", page 53. • slsc_cfg_set_rot_and_offset_2d and slsc_list_set_rot_and_offset_2d change the target point coordinates (argument <code>Target</code>, see also page 268) of slsc_list_multi_para_dashed_mark_abs. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_list_multi_para_mark_abs

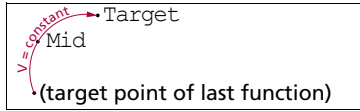
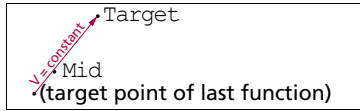
Name of the function	<code>slsc_list_multi_para_mark_abs</code>
Purpose	Like <code>slsc_list_mark_abs</code> . But offers the argument <code>MultiParaTarget</code> additionally, by which (per "ActiveChannel") a Ramp consisting of several sections is defined.
Function signature	<code>uint32_t slsc_list_multi_para_mark_abs(size_t Handle, const double* Target, const slsc_MultiParaTarget* MultiParaTarget);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
	MultiParaTarget Pointer to an array of dimension 2 or 1, see Comment(s), page 250 . 1 Ramp per "ActiveChannel" (there are at most 2 "ActiveChannel") consisting of several sections (<code>ds</code>), see structure <code>slsc_MultiParaTarget</code> .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • See <code>slsc_list_para_mark_abs</code>. • <code>slsc_list_multi_para_mark_abs</code> belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. • At the beginning of the Ramp, the achieved end value of the previous <code>slsc_list_[para/multi_para]*</code> function or (initially <code>ParaTargetDefault</code>) is used as start value (Factor Ip). • If the sum of the sections (<code>ds</code>) is <i>shorter</i> than the mark vector length lasts, then the Ramp value achieved at the end of the last section value is kept unchanged until the end of the marking vector. • If the sum of the sections (<code>ds</code>) is <i>longer</i> than the mark vector length lasts, then the Ramp is cancelled at this point in time. The Ramp value achieved until then is used as start value for the following Ramp. • See also Section "Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)", page 92. • See also Section "About Ramps", page 53. • <code>slsc_cfg_set_rot_and_offset_2d</code> and <code>slsc_list_set_rot_and_offset_2d</code> change the target point coordinates (argument <code>Target</code>, see also page 268) of <code>slsc_list_multi_para_mark_abs</code>. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0.
References	<code>slsc_list_para_mark_abs</code>

Name of the function	slsc_list_para_arc_abs
Purpose	Like slsc_list_arc_abs . But offers the argument ParaTarget additionally, by which a Ramp is defined (in the working field, the value/s of one/two "ActiveChannel" is/are varied linearly).
Function signature	<code>uint32_t slsc_list_para_arc_abs(size_t Handle, const double* Mid, const double* Target, const double* ParaTarget);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Mid Pointer to an array of dimension 2. Coordinates of a point on the circular arc between target point of the last function and Target . In mm.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
	ParaTarget Array of dimension 2 or 1, see Comment(s) , page 250. ParaTarget refers to the end of the following Ramp . Is used as Factor Ip for calculating the ActiveChannel values, see Section "About how ActiveChannel Values along a Contour are Calculated" , page 51.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> Like with slsc_list_arc_abs – a circular arc is defined by 3 points: <ul style="list-style-type: none"> First point = target of the last function. Mid = a point (on the circular arc) somewhere between first point and Target. Target = target point.  Like with slsc_list_arc_abs – a straight line is marked, if the 3 points result in (nearly) lying on a single line (are collinear).  Like with slsc_list_arc_abs – once slsc_list_para_arc_abs is executed, the laser is switched on, and then the circular arc is scanned with a constant speed. <ul style="list-style-type: none"> If the marking is smaller than approx. the half scan head working field: the applied speed is the currently set marking speed. If the marking is greater than approx. the half scan head working field: the applied speed is reduced to a speed to a suiting positioning stage speed. This is specified in the characteristic (see DynamicReductionFunction). slsc_list_para_arc_abs belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45.

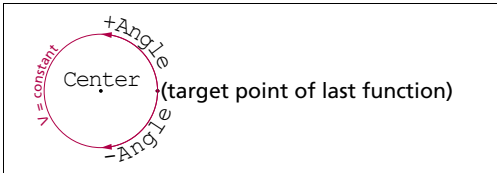
Name of the function	slsc_list_para_arc_abs
Comment(s) (cont'd)	<ul style="list-style-type: none"> Each slsc_list_[para/multi_para]* function works as its corresponding slsc_list* function, if no "ActiveChannel" has been entered, see Section "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48. See also Section "Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)", page 92. slsc_cfg_set_rot_and_offset_2d and slsc_list_set_rot_and_offset_2d change the target point coordinates (argument <code>Target</code> and <code>Mid</code>, see also page 268) of slsc_list_para_arc_abs. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0.
References	slsc_list_arc_abs , slsc_list_para_circle_2d_abs , slsc_list_para_disable , slsc_list_para_enable , slsc_list_para_jump_abs , slsc_list_para_mark_abs

Name of the function	<code>slsc_list_para_circle_2d_abs</code>
Purpose	Like <code>slsc_list_circle_2d_abs</code> . But offers the argument <code>ParaTarget</code> additionally, by which a Ramp is defined (in the working field, the value/s of one/two "ActiveChannel" is/are varied linearly).
Function signature	<code>uint32_t slsc_list_para_circle_2d_abs(size_t Handle, const double* Center, double Angle, const double* ParaTarget);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Center Pointer to an array of dimension 2. Coordinates (x value and y value) of the circle center. In mm.
	Angle In rad. Positive values: marking is carried-out counterclockwise. Negative values: marking is carried-out clockwise.
	ParaTarget Array of dimension 2 or 1, see Comment(s) , page 250 . <code>ParaTarget</code> refers to the end of the following Ramp . Is used as Factor Ip for calculating the ActiveChannel values, see Section "About how ActiveChannel Values along a Contour are Calculated" , page 51 .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279 .
Comment(s)	<ul style="list-style-type: none"> Like with <code>slsc_list_circle_2d_abs</code> - the circle is defined by 2 points: <ul style="list-style-type: none"> First point = target of the last Mark function or Jump command. Center = center of the circle.  Like with <code>slsc_list_circle_2d_abs</code> - once <code>slsc_list_para_circle_2d_abs</code> is executed, the laser is switched on and then the circle is scanned with a constant speed. The execution direction of the marking as well as the number of rotations (for example, $3,25 \times 2\pi$) is determined by the argument <code>Angle</code>. <code>slsc_list_para_circle_2d_abs</code> belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. Each <code>slsc_list_[para/multi_para]*</code> function works as its corresponding <code>slsc_list*</code> function, if no "ActiveChannel" has been entered, see Section "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48. Prior to <code>slsc_list_para_circle_2d_abs</code>, <code>slsc_list_para_enable</code> must have been called. Otherwise, <code>slsc_list_para_circle_2d_abs</code> works as <code>slsc_list_circle_2d_abs</code>. See also Section "Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)", page 92.

Name of the function	<code>slsc_list_para_circle_2d_abs</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> • <code>slsc_cfg_set_rot_and_offset_2d</code> and <code>slsc_list_set_rot_and_offset_2d</code> change the target point coordinates (argument <code>Center</code>, see also page 268) of <code>slsc_list_para_circle_2d_abs</code>. • On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0.
References	slsc_list_circle_2d_abs , slsc_list_para_arc_abs , slsc_list_para_disable , slsc_list_para_enable , slsc_list_para_jump_abs , slsc_list_para_mark_abs

Name of the function	slsc_list_para_dashed_arc_abs
Purpose	Like slsc_list_para_arc_abs , but the corresponding [*]dashed[*] Function. Therefore, offers the arguments NSwitches and LaserSwitches in addition, see Section "[*]dashed[*] Functions", page 91 .
Function signature	<code>uint32_t slsc_list_para_dashed_arc_abs(size_t Handle, const double* Mid, const double* Target, size_t NSwitches, const double* LaserSwitches, const double* ParaTarget);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Mid Pointer to an array of dimension 2. Coordinates of a point on the circular arc between target point of the last function and Target . In mm.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
	NSwitches NSwitches is the size of the LaserSwitches array. It specifies how often the laser is to be switched on/off along the marking pattern section. Minimum value: 1. See Section "[*]dashed[*] Functions", page 91.
	LaserSwitches LaserSwitches is an array of double values. The array specifies at which arc length values (in mm) a switching of "Laser Standby" Operation and "Laser Active" Operation has to occur. See Section "[*]dashed[*] Functions", page 91.
	ParaTarget Array of dimension 2 or 1, see Comment(s), page 250 . ParaTarget refers to the end of the following Ramp . Is used as Factor Ip for calculating the ActiveChannel values, see Section "About how ActiveChannel Values along a Contour are Calculated", page 51 .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions", page 279 .
Comment(s)	<ul style="list-style-type: none"> Like with slsc_list_arc_abs – a circular arc is defined by 3 points: <ul style="list-style-type: none"> First point = target of the last function. Mid = a point (on the circular arc) somewhere between first point and Target. Target = target point.  Like with slsc_list_arc_abs – a straight line is marked, if the 3 points result in (nearly) lying on a single line (are collinear).  Like with slsc_list_arc_abs – once slsc_list_para_dashed_arc_abs is executed, the laser is switched on, and then the circular arc is scanned with a constant speed. <ul style="list-style-type: none"> If the marking is smaller than approx. the half scan head working field: the applied speed is the currently set marking speed. If the marking is greater than approx. the half scan head working field: the applied speed is reduced to a speed to a suiting positioning stage speed. This is specified in the characteristic (see DynamicReductionFunction).

Name of the function	<code>slsc_list_para_dashed_arc_abs</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> • <code>slsc_list_para_dashed_arc_abs</code> belongs to the “MIRROR” functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 “About the Point in Time when Output Signals are actually set”, page 45. • Each <code>slsc_list_[para/multi_para]*</code> function works as its corresponding <code>slsc_list*</code> function, if no “ActiveChannel” has been entered, see Section “About Automatically Controlling the Laser by syncAXIS control (“Automatic Laser Control”)”, page 48. • See also Section “Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)”, page 92. • <code>slsc_cfg_set_rot_and_offset_2d</code> and <code>slsc_list_set_rot_and_offset_2d</code> change the target point coordinates (argument <code>Target</code> and <code>Mid</code>, see also page 268) of <code>slsc_list_para_dashed_arc_abs</code>. • On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_list_para_arc_abs

Name of the function	slsc_list_para_dashed_circle_2d_abs
Purpose	Like slsc_list_circle_2d_abs , but the corresponding [*]dashed[*] Function. Therefore, offers the arguments NSwitches and LaserSwitches in addition, see Section "[*]dashed[*] Functions" , page 91.
Function signature	<code>uint32_t slsc_list_para_dashed_circle_2d_abs(size_t Handle, const double* Center, double Angle, size_t NSwitches, const double* LaserSwitches, const double* ParaTarget);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Center Pointer to an array of dimension 2. Coordinates (x value and y value) of the circle center. In mm.
	Angle In rad. Positive values: marking is carried-out counterclockwise. Negative values: marking is carried-out clockwise.
	NSwitches NSwitches is the size of the LaserSwitches array. It specifies how often the laser is to be switched on/off along the marking pattern section. Minimum value: 1. See Section "[*]dashed[*] Functions" , page 91.
	LaserSwitches LaserSwitches is an array of double values. The array specifies at which arc length values (in mm) a switching of "Laser Standby" Operation and "Laser Active" Operation has to occur. See Section "[*]dashed[*] Functions" , page 91.
	ParaTarget Array of dimension 2 or 1, see Comment(s) , page 250. ParaTarget refers to the end of the following Ramp . Is used as Factor Ip for calculating the ActiveChannel values, see Section "About how ActiveChannel Values along a Contour are Calculated" , page 51.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> Like with slsc_list_circle_2d_abs - the circle is defined by 2 points: <ul style="list-style-type: none"> First point = target of the last Mark function or Jump command. Center = center of the circle.  Like with slsc_list_circle_2d_abs - once slsc_list_para_dashed_circle_2d_abs is executed, the laser is switched on and then the circle is scanned with a constant speed. The execution direction of the marking as well as the number of rotations (for example, $3,25 \times 2\pi$) is determined by the argument Angle. slsc_list_para_dashed_circle_2d_abs belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. Each slsc_list_[para/multi_para]* function works as its corresponding slsc_list* function, if no "ActiveChannel" has been entered, see Section "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48.

Name of the function	<code>slsc_list_para_dashed_circle_2d_abs</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> • Prior to <code>slsc_list_para_dashed_circle_2d_abs</code>, <code>slsc_list_para_enable</code> must have been called. Otherwise, <code>slsc_list_para_dashed_circle_2d_abs</code> works as <code>slsc_list_circle_2d_abs</code>. • See also Section "Functions for Defining Ramps (<code>slsc_list_[para/multi_para]*-Functions</code>)", page 92. • <code>slsc_cfg_set_rot_and_offset_2d</code> and <code>slsc_list_set_rot_and_offset_2d</code> change the target point coordinates (argument <code>Center</code>, see also page 268) of <code>slsc_list_para_dashed_circle_2d_abs</code>. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	<code>slsc_list_para_circle_2d_abs</code>

Name of the function	slsc_list_para_dashed_mark_abs
Purpose	Like slsc_list_para_dashed_mark_abs , but the corresponding [*]dashed[*] Function. Therefore, offers the arguments NSwitches and LaserSwitches in addition, see Section "[*]dashed[*] Functions" , page 91.
Function signature	<code>uint32_t slsc_list_para_dashed_mark_abs(size_t Handle, const double* Target, size_t NSwitches, const double* LaserSwitches, const double* ParaTarget);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
	NSwitches NSwitches is the size of the LaserSwitches array. It specifies how often the laser is to be switched on/off along the marking pattern section. Minimum value: 1. See Section "[*]dashed[*] Functions" , page 91.
	LaserSwitches LaserSwitches is an array of double values. The array specifies at which arc length values (in mm) a switching of "Laser Standby" Operation and "Laser Active" Operation has to occur. See Section "[*]dashed[*] Functions" , page 91.
	ParaTarget Array of dimension 2 or 1, see Comment(s) , page 250. ParaTarget refers to the end of the following Ramp . Is used as Factor Ip for calculating the ActiveChannel values, see Section "About how ActiveChannel Values along a Contour are Calculated" , page 51.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> Like with slsc_list_mark_abs – the behavior of mark vectors is defined in the configuration of the Trajectory planning, see slsc_MarkConfig (for example, MarkSpeed) and slsc_GeometryConfig (for example, MaxBlendRadius). slsc_list_para_dashed_mark_abs belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. Each slsc_list_[para/multi_para]* function works as its corresponding slsc_list* function, if no "ActiveChannel" has been entered, see Section "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48. Prior to slsc_list_para_dashed_mark_abs, slsc_list_para_enable must have been called. Otherwise, slsc_list_para_dashed_mark_abs works as slsc_list_mark_abs.

Name of the function	<code>slsc_list_para_dashed_mark_abs</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> • See also Section "Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)", page 92. • <code>slsc_cfg_set_rot_and_offset_2d</code> and <code>slsc_list_set_rot_and_offset_2d</code> change the target point coordinates (argument <code>Target</code>, see also page 268) of <code>slsc_list_para_dashed_mark_abs</code>. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_list_para_dashed_mark_abs

Name of the function	slsc_list_para_disable
Purpose	Switches the processing of the arguments <code>ParaTarget</code> (of slsc_list_para* functions) and <code>MultiParaTarget</code> (of slsc_list_multi_para* functions) off.
Function signature	<code>uint32_t slsc_list_para_disable(size_t Handle);</code>
Argument(s)	<code>Handle</code> Handle to a syncAXIS control instance.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • After calling slsc_list_para_disable, all slsc_list_[para/multi_para]*-functions work like their corresponding slsc_list_*-functions. All subsequent <code>ParaTarget</code> and <code>MultiParaTarget</code> values are set to 1 (Factor Ip = 1) (which means "no change" for the output value/s of the ActiveChannel/s). • The processing of the arguments <code>ParaTarget</code> (of slsc_list_para* functions) and <code>MultiParaTarget</code> (of slsc_list_multi_para* functions) is switched back on by slsc_list_para_enable. • See also Section "Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)", page 92. • On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0.
References	slsc_list_para_enable

Name of the function	slsc_list_para_enable
Purpose	Switches the processing of the arguments <code>ParaTarget</code> (of <code>slsc_list_para*</code> functions) and <code>MultiParaTarget</code> (of <code>slsc_list_multi_para*</code> functions) on.
Function signature	<code>uint32_t slsc_list_para_enable(size_t Handle, const double* ParaTargetDefault);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	<code>ParaTargetDefault</code> Array of dimension 2 or 1, see Comment(s) . Start value/s of the first Ramp .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> The dimension of the array for the Ramp value should correspond to the number of defined-as-active channels. In <code>syncAXISConfig.xml</code> under <code><cfg:Configuration></code> → <code><cfg:LaserConfig></code> → <code><cfg:AutomaticLaserControl></code> → <code><cfg:ActiveChannel></code> there is <ul style="list-style-type: none"> 2× the tag <code><cfg:Channel></code> = 2 channels are defined as active: Pointer to an array of dimension 2. 1× the tag <code><cfg:Channel></code> = 1 channel is defined as active: Pointer to an array of dimension 1. 0× the tag <code><cfg:Channel></code> = no "ActiveChannel" is defined. <code>ParaTargetDefault</code> is not evaluated! If <code>slsc_list_para_enable</code> is not called, all <code>slsc_list_[para/multi_para]*</code>-functions work like their corresponding <code>slsc_list*</code>-functions. The same applies, if the processing of the arguments <code>ParaTarget</code> (of <code>slsc_list_para*</code> functions) and <code>MultiParaTarget</code> (of <code>slsc_list_multi_para*</code> functions) has been switched off by <code>slsc_list_para_disable</code>. See also Section "Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)", page 92. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0.
References	<code>slsc_list_para_disable</code>

Name of the function	<code>slsc_list_para_jump_abs</code>
Purpose	Like <code>slsc_list_jump_abs</code> . But offers the argument <code>ParaTarget</code> additionally, by which a Ramp is defined (in the working field, the value/s of one/two "ActiveChannel" is/are varied linearly).
Function signature	<code>uint32_t slsc_list_para_jump_abs(size_t Handle, const double* Target, const double* ParaTarget);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
	ParaTarget Array of dimension 2 or 1, see Comment(s) , page 250. <code>ParaTarget</code> refers to the end of the following Ramp . Is used as Factor Ip for calculating the ActiveChannel values, see Section "About how ActiveChannel Values along a Contour are Calculated" , page 51.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> Like with <code>slsc_list_jump_abs</code> – the behavior of jumps is defined in the configuration of the Trajectory planning, see <code>slsc_MarkConfig</code> (for example, <code>LaserMinOffTime</code> and <code>JumpSpeed</code>). <code>slsc_list_para_jump_abs</code> belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. Each <code>slsc_list_[para/multi_para]*</code> function works as its corresponding <code>slsc_list*</code> function, if no "ActiveChannel" has been entered, see Section "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48. Prior to <code>slsc_list_para_jump_abs</code>, <code>slsc_list_para_enable</code> must have been called. Otherwise, <code>slsc_list_para_jump_abs</code> works as <code>slsc_list_jump_abs</code>. See also Section "Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)", page 92. <code>slsc_cfg_set_rot_and_offset_2d</code> and <code>slsc_list_set_rot_and_offset_2d</code> change the target point coordinates (argument <code>Target</code>, see also page 268) of <code>slsc_list_para_jump_abs</code>. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0.
References	<code>slsc_list_jump_abs</code> , <code>slsc_list_para_arc_abs</code> , <code>slsc_list_para_circle_2d_abs</code> , <code>slsc_list_para_disable</code> , <code>slsc_list_para_enable</code> , <code>slsc_list_para_mark_abs</code>

Name of the function	<code>slsc_list_para_jump_abs_min_time</code>
Purpose	Like <code>slsc_list_jump_abs_min_time</code> . But offers the argument <code>ParaTarget</code> additionally, by which a Ramp is defined (in the working field, the value/s of one/two “ActiveChannel” is/are varied linearly).
Function signature	<code>uint32_t slsc_list_para_jump_abs_min_time(size_t Handle, const double* Target, double MinimalJumpTime, const double* ParaTarget);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
	MinimalJumpTime Minimum duration for the jump. In s. Only positive values are allowed.
	ParaTarget Array of dimension 2 or 1, see Comment(s) , page 250. <code>ParaTarget</code> refers to the end of the following Ramp . Is used as Factor Ip for calculating the ActiveChannel values, see Section “About how ActiveChannel Values along a Contour are Calculated” , page 51.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> Like with <code>slsc_list_jump_abs_min_time</code> – the behavior of jumps is defined in the configuration of the Trajectory planning, see <code>slsc_MarkConfig</code> (for example, <code>LaserMinOffTime</code> and <code>JumpSpeed</code>). <code>slsc_list_para_jump_abs_min_time</code> belongs to the “MIRROR” functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 “About the Point in Time when Output Signals are actually set”, page 45. Each <code>slsc_list_[para/multi_para]*</code> function works as its corresponding <code>slsc_list*</code> function, if no “ActiveChannel” has been entered, see Section “About Automatically Controlling the Laser by syncAXIS control (“Automatic Laser Control”)”, page 48. Prior to <code>slsc_list_para_jump_abs_min_time</code>, <code>slsc_list_para_enable</code> must have been called. Otherwise, <code>slsc_list_para_jump_abs_min_time</code> works as <code>slsc_list_jump_abs_min_time</code>. See also Section “Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)”, page 92. <code>slsc_cfg_set_rot_and_offset_2d</code> and <code>slsc_list_set_rot_and_offset_2d</code> change the target point coordinates (argument <code>Target</code>, see also page 268) of <code>slsc_list_para_jump_abs_min_time</code>. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.6.0.
References	<code>slsc_list_jump_abs_min_time</code>

Name of the function	slsc_list_para_mark_abs
Purpose	Like slsc_list_mark_abs . But offers the argument <code>ParaTarget</code> additionally, by which a Ramp is defined (in the working field, the value/s of one/two "ActiveChannel" is/are varied linearly).
Function signature	<code>uint32_t slsc_list_para_mark_abs(size_t Handle, const double* Target, const double* ParaTarget);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Target Pointer to an array of dimension 2. Coordinates of the target point. In mm.
	ParaTarget Array of dimension 2 or 1, see Comment(s) , page 250. <code>ParaTarget</code> refers to the end of the following Ramp . Is used as Factor Ip for calculating the ActiveChannel values, see Section "About how ActiveChannel Values along a Contour are Calculated" , page 51.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> Like with slsc_list_mark_abs – the behavior of mark vectors is defined in the configuration of the Trajectory planning, see slsc_MarkConfig (for example, <code>MarkSpeed</code>) and slsc_GeometryConfig (for example, <code>MaxBlendRadius</code>). slsc_list_para_mark_abs belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. Each slsc_list_[para/multi_para]* function works as its corresponding slsc_list* function, if no "ActiveChannel" has been entered, see Section "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48. Prior to slsc_list_para_mark_abs, slsc_list_para_enable must have been called. Otherwise, slsc_list_para_mark_abs works as slsc_list_mark_abs. See also Section "Functions for Defining Ramps (slsc_list_[para/multi_para]*-Functions)", page 92. slsc_cfg_set_rot_and_offset_2d and slsc_list_set_rot_and_offset_2d change the target point coordinates (argument <code>Target</code>, see also page 268) of slsc_list_para_mark_abs. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0 .
References	slsc_list_mark_abs , slsc_list_para_arc_abs , slsc_list_para_circle_2d_abs , slsc_list_para_disable , slsc_list_para_enable , slsc_list_para_jump_abs

Name of the function	<code>slsc_list_para_playback_module</code>
Purpose	Like <code>slsc_list_playback_module</code> , however, parameter values on Ramps are applied, if there is a <code>slsc_list_para_enable</code> in advance.
Function signature	<code>uint32_t slsc_list_para_playback_module(size_t Handle, const char* ModuleFileName);</code>
Argument(s)	Handle Like <code>slsc_list_playback_module</code> .
	ModuleFileName Like <code>slsc_list_playback_module</code> .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_list_para_playback_module</code> behaves like <code>slsc_list_playback_module</code>, if <code>slsc_list_para_enable</code> has not been called in advance. Regarding the parameter values for automatic laser control, <code>slsc_list_para_playback_module</code> behaves mostly like other <code>slsc_list_para_[*]/slsc_list_multi_para_[*]</code> functions. The main exception is that these values cannot be redefined at replay time. Therefore, when the Module is replayed, these values start with the recorded values and <i>not</i> with the end values of the preceding function or with the <code>ParaTargetDefault</code> value specified in <code>slsc_list_para_enable</code>. When using <code>slsc_list_para_playback_module</code>, make sure that the <code>ActiveChannel</code> configuration in the <code>syncAXISConfig.xml</code> (see Chapter 2.9.2 "Definition of the Channels and ActiveChannel", page 48) matches in regards to recording time and replay time. See Chapter 2.11 "About Working with "Modules"", page 65.
Code example	–
Version info	Available as of syncAXIS-DLL V1.3.0.
References	<code>slsc_list_playback_module</code>

Name of the function	<code>slsc_list_playback_module</code>
Purpose	Integrates ("replays") a Module into the current Job . Parameter values on Ramps are not applied.
Function signature	<code>uint32_t slsc_list_playback_module(size_t Handle, const char* ModuleFileName);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	ModuleFileName Absolute file path of the Module file to be read in (*.slm). Pointer to a \0-terminated ANSI string, 1 byte per char.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • slsc_list_playback_module is allowed in simulation mode and (other than slsc_list_begin_module as well in) hardware mode. • slsc_list_playback_module is allowed in Operation mode "ScannerOnly", "StageOnly", "ScannerAndStage". • For slsc_list_playback_module, there is no corresponding Configuration function (slsc_cfg_*). • See also Section "Functions for "Modules"", page 95. • In the following cases, slsc_list_playback_module is rejected and the return value indicates that a bit is set: <ul style="list-style-type: none"> – The specified Module file cannot be opened Bit #06 (UnplausibleOrUnknownParameter) – The version number in Module file is incompatible Bit #06 (UnplausibleOrUnknownParameter) – The Job to be recorded has not been completely written to the Module file. Bit #06 (UnplausibleOrUnknownParameter) – The Operation mode is incompatible Bit #09 (NotAllowedInCurrentConfiguration) <ul style="list-style-type: none"> • A ScannerOnly (ScannerAndStage) recorded module must not be replayed in StageOnly Operation mode. • A StageOnly recorded module may only be replayed in StageOnly Operation mode. – JumpSpeed or MarkSpeed in the Module are greater than speed limit at replay time Bit #09 (NotAllowedInCurrentConfiguration) – Acceleration limit or jerk limit in the Module greater than the corresponding limit at replay time Bit #09 (NotAllowedInCurrentConfiguration)

Name of the function	slsc_list_playback_module
Comment(s) (cont'd)	<ul style="list-style-type: none"> The following cases merely trigger a new [WARN] log file line (see [WARN] log file lines): <ul style="list-style-type: none"> The lowest jump time in the Module is smaller than LaserPreTriggerTime at replay time. Some laser switching time points may not be set as expected. Delay values for scan device and positioning stage in the Module differ to those at replay time. With smaller Delays at replay time some "SIGNAL" functions for which negative TimeDelay values are specified may get ignored. The Module has been recorded in Operation mode ScannerOnly (ScannerAndStage) and is replayed in Operation mode ScannerAndStage (ScannerOnly). Only relevant for slsc_list_para_playback_module: different ActiveChannel configuration in the module for replay time than for recording time. On transitions ("Module boundaries") into the Module Trajectory: <ul style="list-style-type: none"> If the connection cannot be made directly, there is always a Sky Writing-like motion (no blending). If a Module begins (ends) with a jump, it is not combined with jumps following at replay time, but both jumps are executed separately. See Chapter 2.11 "About Working with "Modules"", page 65. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	See Figure 29, page 69 .
Version info	Available as of syncAXIS-DLL V1.3.0.
References	slsc_list_begin_module , slsc_list_para_playback_module

Name of the function	<code>slsc_list_set_approx_blend_limit</code>
Purpose	Changes the <code>ApproxBlendLimit</code> value, which is specified in the configuration of the Trajectory planning (see below). This change applies to all following Job functions (<code>slsc_list_*</code>) but only until the end of the Job .
Function signature	<code>uint32_t slsc_list_set_approx_blend_limit(size_t Handle, double ApproxBlendLimit);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	<code>ApproxBlendLimit</code> ApproxBlendLimit value. See also <code>slsc_GeometryConfig</code> .
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_list_set_approx_blend_limit</code> changes the configuration of the specified syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change applies as of the insert position but only until the end of the currently running Job. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	–

Name of the function	<code>slsc_list_set_calculation_dynamics_jump_scan_device</code>
Purpose	<p>Changes:</p> <ul style="list-style-type: none"> The maximum acceleration and jerk value of the intended scan device type. The values are used only in Trajectory planning calculations of the scan device motion – however, only for jumps but not markings <p>This change applies to all following Job functions (<code>slsc_list_*</code>) but only until the end of the Job.</p>
Function signature	<code>uint32_t slsc_list_set_calculation_dynamics_jump_scan_device(size_t Handle, double JumpAngularAcc, double JumpAngularJerk);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	JumpAngularAcc Like JumpAngularAcc.
	JumpAngularJerk Like JumpAngularJerk.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> ⚠ Caution! syncAXIS control uses the Acceleration value = <code>JumpAngularAcc</code> = <code>JumpAngularAcc</code> to plan trajectories for the Operation modes “ScannerOnly” and “ScannerAndStage”. Make sure that the entered values are correct. ⚠ Caution! syncAXIS control uses the Jerk value = <code>JumpAngularJerk</code> = <code>JumpAngularJerk</code> to plan trajectories for the Operation modes “ScannerOnly” and “ScannerAndStage”. Make sure that the entered values are correct. <code>slsc_list_set_calculation_dynamics_jump_scan_device</code> changes the configuration of the specified syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change applies as of the insert position but only until the end of the currently running Job. The corresponding Configuration function (<code>slsc_cfg_*</code>) of <code>slsc_list_set_calculation_dynamics_jump_scan_device</code> is <code>slsc_cfg_set_calculation_dynamics_jump_scan_device</code>. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.6.0.
References	<code>slsc_cfg_get_calculation_dynamics_jump_scan_device</code>, <code>slsc_cfg_set_calculation_dynamics_jump_scan_device</code>

Name of the function	<code>slsc_list_set_calculation_dynamics_mark_scan_device</code>
Purpose	<p>Changes:</p> <ul style="list-style-type: none"> The maximum acceleration and jerk value of the intended scan device type. The values are used only in Trajectory planning calculations of the scan device motion – however, only for markings but not jumps <p>This change applies to all following Job functions (<code>slsc_list_*</code>) but only until the end of the Job.</p>
Function signature	<code>uint32_t slsc_list_set_calculation_dynamics_mark_scan_device(size_t Handle, double MarkAngularAcc, double MarkAngularJerk);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	MarkAngularAcc Like MarkAngularAcc.
	MarkAngularJerk Like MarkAngularJerk.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> ⚠ Caution! syncAXIS control uses the Acceleration value = <code>MarkAngularAcc</code> = <code>MarkAngularAcc</code> to plan trajectories for the Operation modes “ScannerOnly” and “ScannerAndStage”. Make sure that the entered values are correct. ⚠ Caution! syncAXIS control uses the Jerk value = <code>MarkAngularJerk</code> = <code>MarkAngularJerk</code> to plan trajectories for the Operation modes “ScannerOnly” and “ScannerAndStage”. Make sure that the entered values are correct. <code>slsc_list_set_calculation_dynamics_mark_scan_device</code> changes the configuration of the specified syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change applies as of the insert position only until the end of the currently running Job. The corresponding Configuration function (<code>slsc_cfg_*</code>) of <code>slsc_list_set_calculation_dynamics_mark_scan_device</code> is <code>slsc_cfg_set_calculation_dynamics_mark_scan_device</code>. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.6.0.
References	<code>slsc_cfg_get_calculation_dynamics_mark_scan_device</code> , <code>slsc_cfg_set_calculation_dynamics_mark_scan_device</code>

Name of the function	<code>slsc_list_set_contour_dependent_speed_control_2d</code>
Purpose	Switches on/off the “Contour-dependent speed calculation”. Furthermore, it can be changed how the syncAXIS control instance internally determines speeds along curves (“left” or “right” of the curve mid-line; distance to it). Once the “Automatic Laser Control” is activated, these results are used to correspondingly set, for example, the laser spot distances equidistant. This change applies to all following Job functions (<code>slsc_list_*</code>) but only until the end of the Job .
Function signature	<code>uint32_t slsc_list_set_contour_dependent_speed_control_2d(size_t Handle, int32_t Direction, double SpotRadius);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Direction 0: “Contour-dependent speed calculation” = off. Speeds are determined on the curve mid-line. Is also the default status after syncAXIS control instance initialization by slsc_cfg_initialize_from_file . +1: “Contour-dependent speed calculation” = on. Speeds are determined right of the curve mid-line. -1: “Contour-dependent speed calculation” = on. Speeds are determined left of the curve mid-line.
	SpotRadius Radius of the laser spot in the working plane. In mm. The value specifies how far to the right or left of the curve mid-line the speeds are determined.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_list_set_contour_dependent_speed_control_2d changes the configuration of the specified syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change applies as of the insert position but only until the end of the currently running Job. slsc_list_set_contour_dependent_speed_control_2d has no effect (no error is returned), if the “Automatic Laser Control” is not switched on (for example, no ActiveChannel is entered in <code>syncAXISConfig.xml</code>). See also Chapter 2.9.5 “About the “Contour-dependent speed calculation””, page 60. The corresponding Configuration function (<code>slsc_cfg_*</code>) of slsc_list_set_contour_dependent_speed_control_2d is slsc_cfg_set_contour_dependent_speed_control_2d. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0.
References	slsc_cfg_set_contour_dependent_speed_control_2d , slsc_list_set_laser_on_move

Name of the function	<code>slsc_list_set_free_variable</code>
Purpose	Like <code>slsc_ctrl_set_free_variable</code> .
Function signature	<code>uint32_t slsc_list_set_free_variable(size_t Handle, uint32_t Number, uint32_t Value, double TimeDelay);</code>
Argument(s)	Handle Handle to a <code>syncAXIS</code> control instance.
	Number Number of the free variable on the RTC6 to be set. Allowed value range: [0...7]. Only the three least significant bits are evaluated.
	Value Value of the free variable to be set on the RTC6.
	TimeDelay Relative point in time between 2 Job functions (<code>slsc_list_*</code>), when the change is going to be applied (reference point: point in time at <i>marking execution</i> when the target point of the first Job function is reached, see Figure 14, page 46). In s. Only positive values are allowed.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> In simulation mode, <code>slsc_ctrl_get_free_variable</code>, <code>slsc_ctrl_set_free_variable</code>, as well as <code>slsc_list_set_free_variable</code> have no effect. For excessive <code>TimeDelay</code> values, the change is applied with <code>slsc_list_end</code> at latest. The change <ul style="list-style-type: none"> – is persistent – applies to all <code>Jobs</code> that follow <code>slsc_list_set_free_variable</code> belongs to the "SIGNAL" functions. See also Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. <code>slsc_list_set_free_variable</code> is a direct implementation of the RTC6 command <code>set_free_variable_list</code> in <code>syncAXIS</code> control. However, <code>set_free_variable_list</code> does not provide the configuration parameter <code>TimeDelay</code>. The functions for free variables (<code>slsc_ctrl_set_free_variable</code>, <code>slsc_list_set_free_variable</code> and <code>slsc_ctrl_get_free_variable</code>) can be used, for example, to determine and count increments (within <code>Jobs</code>). For further information on free variables, refer to the RTC6 Manual, Chapter 6.9.1 "Free Variables", page 134. The current value of a free variable can be queried by <code>slsc_ctrl_get_free_variable</code>.

Name of the function	<code>slsc_list_set_free_variable</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> The corresponding Control function (<code>slsc_ctrl_*</code>) of <code>slsc_list_set_free_variable</code> is <code>slsc_ctrl_set_free_variable</code>. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	<p>Available as of syncAXIS-DLL V1.1.2.</p> <p>Latest change with syncAXIS-DLL V1.2.4: <code>TimeDelay</code> values for V1.1 are shifted syncAXIS-DLL-internally by a certain delay in V1.2 (for example, in Operation mode "ScannerOnly" by 0.00125 s).</p>
References	slsc_ctrl_get_free_variable , slsc_ctrl_set_free_variable

Name of the function	<code>slsc_list_set_jump_speed</code>
Purpose	Changes the jump speed. This change applies to all following Job functions (<code>slsc_list_*</code>) but only until the end of the Job .
Function signature	<code>uint32_t slsc_list_set_jump_speed(size_t Handle, double JumpSpeed);</code>
Argument(s)	<code>Handle</code> Handle to a syncAXIS control instance.
	<code>JumpSpeed</code> Jump speed. In mm/s.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_list_set_jump_speed</code> changes the configuration of the specified syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change applies as of the insert position (but other than with the similar RTC command) but only until the end of the currently running Job. The corresponding Configuration function (<code>slsc_cfg_*</code>) of <code>slsc_list_set_jump_speed</code> is <code>slsc_cfg_set_jump_speed</code>. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	slsc_cfg_set_jump_speed

Name of the function	slsc_list_set_laser_on_move
Purpose	Delays the “ Laser Active ” Operation by exactly the amount of time needed to travel the specified path length (PathLength) on the current marking section. This change applies to all following Job functions (slsc_list_*) but only until the end of the Job.
Function signature	uint32_t slsc_list_set_laser_on_move(size_t Handle, double PathLength);
Argument(s)	Handle Handle to a syncAXIS control instance.
	PathLength Path length on the marking sections after which the laser is to be actually switched on. In mm. Allowed values: ≥ 0 .
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> • slsc_list_set_laser_on_move changes the configuration of the specified syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change applies as of the insert position but only until the end of the currently running Job. • For a PathLength value < 0 the return value indicates that Bit #06 is set (UnplausibleOrUnknownParameter). • Use case for slsc_list_set_laser_on_move: see Section “About slsc_list_set_laser_on_move”, page 88. • On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.5.0.
References	slsc_list_set_contour_dependent_speed_control_2d

Name of the function	<code>slsc_list_set_laser_pulses</code>
Purpose	Like <code>slsc_ctrl_set_laser_pulses</code> .
Function signature	<code>uint32_t slsc_list_set_laser_pulses(size_t Handle, double HalfPeriod, double PulseLength, double TimeDelay);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	HalfPeriod <i>Half</i> of the output period. In s. Allowed value range: [0...671].
	PulseLength Pulse length of the laser signals LASER1 and LASER2. In s. Allowed value range: [0...671].
	TimeDelay Relative point in time between 2 Job functions (<code>slsc_list_*</code>), when the change is going to be applied (reference point: point in time at <i>marking execution</i> when the target point of the first Job function is reached, see Figure 14, page 46). In s. Only positive values are allowed.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> In simulation mode, <code>slsc_ctrl_set_laser_pulses</code> and <code>slsc_list_set_laser_pulses</code> have no effect. <code>slsc_list_set_laser_pulses</code> changes the configuration of the specified syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. <code>HalfPeriod</code> and <code>PulseLength</code> change what has been set during initialization (by the attributes of the same name in <code>syncAXISConfig.xml</code>-tag <code><cfg:LaserOutput Unit="s" HalfPeriod="..." PulseLength="..." /></code>). <code>slsc_ctrl_set_laser_pulses</code> and <code>slsc_list_set_laser_pulses</code> are provided for those Sky Writings who (due to the laser they use) cannot use the "Automatic Laser Control" to achieve equidistant spot distances and instead want to influence the pulse output via <code>HalfPeriod</code> and <code>PulseLength</code>. If the "Automatic Laser Control" is active and <code>SpotDistance</code> is an "ActiveChannel", see Chapter 2.9.2 "Definition of the Channels and ActiveChannel", page 48, then: <ul style="list-style-type: none"> <code>HalfPeriodV</code> is not effective <code>PulseLength</code> is effective (that is, pulse lengths of laser signal LASER1 and LASER2 are changed) For excessive <code>TimeDelay</code> values, the change is applied with <code>slsc_list_end</code> at latest. The change <ul style="list-style-type: none"> is persistent applies to all Jobs that follow Related RTC6 command: <code>set_laser_pulses</code>. However, it does not provide the configuration parameter <code>TimeDelay</code>. <code>slsc_list_set_laser_pulses</code> belongs to the "SIGNAL" functions. See also Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45.

Name of the function	slsc_list_set_laser_pulses
Comment(s) (cont'd)	<ul style="list-style-type: none"> The corresponding Control function (slsc_ctrl_*) of slsc_list_set_laser_pulses is slsc_ctrl_set_laser_pulses. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.2.4.
References	slsc_ctrl_set_laser_pulses

Name of the function	slsc_list_set_mark_speed
Purpose	Changes the marking speed. This change applies to all following Job functions (slsc_list_*) but only until the end of the Job .
Function signature	<code>uint32_t slsc_list_set_mark_speed(size_t Handle, double MarkSpeed);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
	MarkSpeed Mark speed. In mm/s.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_list_set_mark_speed changes the configuration of the specified syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change applies as of the insert position (but other than with the similar RTC command) but only until the end of the currently running Job. The corresponding Configuration function (slsc_cfg_*) of slsc_list_set_mark_speed is slsc_cfg_set_mark_speed. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0.
References	slsc_cfg_set_mark_speed

Name of the function	<code>slsc_list_set_matrix_and_offset</code>
Purpose	Changes target point coordinates according to a transformation matrix and an offset value. This change applies to all following Job functions (<code>slsc_list_*</code>) but only until the end of the Job.
Function signature	<code>uint32_t slsc_list_set_matrix_and_offset(size_t Handle, const double* Matrix, const double* Offset);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Matrix Pointer to an array of dimension 4. Coefficients $m_{11} \dots m_{22}$ of a (2×2) transformation matrix.
	Offset Pointer to an array of dimension 2. x value and y value by which target points are moved in the working field. In mm.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_list_set_matrix_and_offset</code> changes (for example, like <code>slsc_list_set_rot_and_offset_2d</code>) the configuration of the specified syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change applies as of the insert position but only until the end of the currently running Job. Target point coordinates of Job functions (<code>slsc_list_*</code>): see list bullet on page 268. <code>slsc_list_set_matrix_and_offset</code> (like <code>slsc_cfg_set_matrix_and_offset</code>) calculates the new target points according to (transformation matrix \times target point) + offset: With suitable transformation matrix coefficients (argument <code>Matrix</code>), for example, scaling, rotating or flipping of marking patterns can be achieved. See also Section “Functions for Changing Target Point Coordinates”, page 92. The corresponding Configuration function (<code>slsc_cfg_*</code>) of <code>slsc_list_set_matrix_and_offset</code> is <code>slsc_cfg_set_matrix_and_offset</code>. On the permissibility of syncAXIS control functions in the Mode “Manual Positioning”, see Chapter 2.12 “About the Mode “Manual Positioning””, page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.1.0.
References	<code>slsc_cfg_set_matrix_and_offset</code>

Name of the function	<code>slsc_list_set_min_mark_speed</code>
Purpose	Changes the minimal marking speed, see MinimalMarkSpeed . This change applies to all following Job functions (<code>slsc_list_*</code>) but only until the end of the Job .
Function signature	<code>uint32_t slsc_list_set_min_mark_speed(size_t Handle, double MinimalMarkSpeed);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
	MinimalMarkSpeed Minimal marking speed. In mm/s.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_list_set_min_mark_speed</code> changes the configuration of the specified syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change applies as of the insert position (but other than with the similar RTC command) but only until the end of the currently running Job. For <code>slsc_list_set_min_mark_speed</code>, there is no corresponding Configuration function (<code>slsc_cfg_*</code>) of. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.2.6.
References	–

Name of the function	<code>slsc_list_set_rot_and_offset_2d</code>
Purpose	Changes target point coordinates by an angle and an offset value. This change applies to all following Job functions (<code>slsc_list_*</code>) but only until the end of the Job .
Function signature	<code>uint32_t slsc_list_set_rot_and_offset_2d(size_t Handle, double Angle, const double* Offset);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Angle Angle (about the origin 0,0) by which target points are rotated in the working field. In rad. Positive values: rotation is counterclockwise. Negative values: rotation is clockwise.
	Offset Pointer to an array of dimension 2. x value and y value by which target points are moved in the working field. In mm.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions” , page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_list_set_rot_and_offset_2d</code> changes the configuration of the specified syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. The change applies as of the insert position but only until the end of the currently running Job. Target point coordinates of Job functions (<code>slsc_list_*</code>) are: <ul style="list-style-type: none"> – Mid and Target of <code>slsc_list_arc_abs</code> – Mid and Target of <code>slsc_list_dashed_arc_abs</code> – Mid and Target of <code>slsc_list_multi_para_arc_abs</code> – Mid and Target of <code>slsc_list_multi_para_dashed_arc_abs</code> – Mid and Target of <code>slsc_list_para_arc_abs</code> – Mid and Target of <code>slsc_list_para_dashed_arc_abs</code> – Center of <code>slsc_list_circle_2d_abs</code> – Center of <code>slsc_list_dashed_circle_2d_abs</code> – Center of <code>slsc_list_multi_para_circle_2d_abs</code> – Center of <code>slsc_list_multi_para_dashed_circle_2d_abs</code> – Center of <code>slsc_list_para_circle_2d_abs</code> – Center of <code>slsc_list_para_dashed_circle_2d_abs</code> – Target of <code>slsc_list_dashed_mark_abs</code> – Target of <code>slsc_list_jump_abs</code> – Target of <code>slsc_list_jump_abs_min_time</code> – Target of <code>slsc_list_mark_abs</code> – Target of <code>slsc_list_multi_para_dashed_mark_abs</code> – Target of <code>slsc_list_multi_para_mark_abs</code> – Target of <code>slsc_list_para_dashed_mark_abs</code> – Target of <code>slsc_list_para_jump_abs</code> – Target of <code>slsc_list_para_jump_abs_min_time</code> – Target of <code>slsc_list_para_mark_abs</code>

Name of the function	<code>slsc_list_set_rot_and_offset_2d</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> <code>slsc_list_set_rot_and_offset_2d</code> calculates the new target points as follows: (rotation matrix \times target point) + offset, thus concretely: <ul style="list-style-type: none"> new target point x value = $((x \times \cos \alpha) - (y \times \sin \alpha)) + x_{\text{Offset value}}$ new target point y value = $((x \times \sin \alpha) + (y \times \cos \alpha)) + y_{\text{Offset value}}$ Example: <div data-bbox="715 629 1332 1090" data-label="Figure"> <p> $P1 = \text{target point of, for example, } \text{slsc_list_jump_abs}$ $\text{Offset } (x=0, y=-2)$ $\alpha = \pi/2$ $R_\alpha = \text{Rotation matrix}_\alpha = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix}$ </p> </div> <ul style="list-style-type: none"> The corresponding Configuration function (<code>slsc_cfg_*</code>) of <code>slsc_list_set_rot_and_offset_2d</code> is <code>slsc_cfg_set_rot_and_offset_2d</code>. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70. A Module can be positioned and rotated anywhere in the space by <code>slsc_list_set_rot_and_offset_2d</code>.
Code example	–
Version info	Available as of syncAXIS-DLL V0.11.0.
References	slsc_cfg_set_rot_and_offset_2d , slsc_list_arc_abs , slsc_list_circle_2d_abs , slsc_list_jump_abs , slsc_list_mark_abs

Name of the function	<code>slsc_list_suppress_spotdistance_control</code>
Purpose	Only if "Automatic Laser Control" is active with <code>SpotDistance</code> as an "ActiveChannel": supplementary function that must precede <code>slsc_list_wait_with_laser_on</code> .
Function signature	<code>uint32_t slsc_list_suppress_spotdistance_control(size_t Handle, double TimeDelay);</code>
Argument(s)	Handle Handle to a <code>syncAXIS</code> control instance.
	TimeDelay Relative point in time between 2 Job functions (<code>slsc_list_*</code>), when the change is going to be applied (reference point: point in time at <i>marking execution</i> when the target point of the first Job function is reached, see Figure 14, page 46). In s. Only positive values are allowed.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> • If the "Automatic Laser Control" is active and <code>SpotDistance</code> is an "ActiveChannel", see Chapter 2.9.2 "Definition of the Channels and ActiveChannel", page 48, then: <code>slsc_list_suppress_spotdistance_control</code> must be called prior to <code>slsc_list_wait_with_laser_on</code>. See Section "Special Case: SpotDistance as an "ActiveChannel"\"", page 87 for further details. • <code>slsc_list_suppress_spotdistance_control</code> and <code>slsc_list_unsuppress_spotdistance_control</code> require <code>SpotDistance</code> to be set as "ActiveChannel". Otherwise, the return value indicates that Bit #09 is set (<code>NotAllowedInCurrentConfiguration</code>). • <code>slsc_list_suppress_spotdistance_control</code> <code>syncAXIS</code>-DLL-internally suppresses the functionality that creates equidistant spot spacings. If this functionality is already suppressed, <code>slsc_list_suppress_spotdistance_control</code> has no effect. • For excessive <code>TimeDelay</code> values, the change is applied with <code>slsc_list_end</code> at latest. • The change <ul style="list-style-type: none"> – is persistent – applies to all Jobs that follow • <code>slsc_list_suppress_spotdistance_control</code> belongs to the "SIGNAL" functions. See also Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. • The complementary function of <code>slsc_list_suppress_spotdistance_control</code> is <code>slsc_list_unsuppress_spotdistance_control</code>. • For <code>slsc_list_suppress_spotdistance_control</code>, there is neither a corresponding Configuration function (<code>slsc_cfg_*</code>) nor Control function (<code>slsc_ctrl_*</code>).

Name of the function	<code>slsc_list_suppress_spotdistance_control</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> <code>slsc_list_suppress_spotdistance_control</code> and <code>slsc_list_unsuppress_spotdistance_control</code> are allowed in all Operation modes (see enum <code>slsc_OperationMode</code>). On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	Available as of syncAXIS-DLL V1.2.7.
References	<code>slsc_list_unsuppress_spotdistance_control</code> , <code>slsc_list_wait_with_laser_on</code>

Name of the function	<code>slsc_list_unsuppress_spotdistance_control</code>
Purpose	Cancels the effect of <code>slsc_list_suppress_spotdistance_control</code> .
Function signature	<code>uint32_t slsc_list_unsuppress_spotdistance_control(size_t Handle, double TimeDelay);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	TimeDelay Relative point in time between 2 Job functions (<code>slsc_list_*</code>), when the change is going to be applied (reference point: point in time at <i>marking execution</i> when the target point of the first Job function is reached, see Figure 14, page 46). In s. Only positive values are allowed.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> See <code>slsc_list_suppress_spotdistance_control</code>. See Section "Special Case: SpotDistance as an "ActiveChannel"", page 87. <code>slsc_list_unsuppress_spotdistance_control</code> belongs to the "SIGNAL" functions. See also Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45.
Code example	–
Version info	Available as of syncAXIS-DLL V1.2.7.
References	<code>slsc_list_suppress_spotdistance_control</code>

Name of the function	<code>slsc_list_wait_with_laser_off</code>
Purpose	Like <code>slsc_list_wait_with_laser_on</code> , but the laser is switched off.
Function signature	<code>uint32_t slsc_list_wait_with_laser_off(size_t Handle, double Time);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Time Duration in which the laser control signal LASERON is switched off. The mirrors remain in their last position. In s.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_list_wait_with_laser_off</code> is rejected, if <code>Time</code> is shorter than a certain time resulting from values specified in the <code>syncAXISConfig.xml</code> (under <code><cfg:Configuration></code> → <code><cfg:TrajectoryConfig></code> → <code><cfg:MarkConfig></code> → <code><cfg:LaserSwitchConfig></code>). The following applies: <ul style="list-style-type: none"> With <code>LaserPreTriggerTime > 0</code>: <code>Time < LaserMinOffTime + LaserPreTriggerTime</code> With <code>LaserPreTriggerTime < 0</code>: <code>Time < LaserMinOffTime</code> Then, the return value indicates that Bit #06 is set (<code>UnplausibleOrUnknownParameter</code>). <code>slsc_list_wait_with_laser_off</code> belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. <code>slsc_list_wait_with_laser_off</code> behaves like a <code>slsc_list_mark_abs</code> with velocity 0 and switched-off laser. The complementary function of <code>slsc_list_wait_with_laser_off</code> is <code>slsc_list_wait_with_laser_on</code>. On the permissibility of <code>syncAXIS</code> control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning" ", page 70.
Code example	–
Version info	Available as of <code>syncAXIS-DLL V1.2.4</code> .
References	<code>slsc_list_wait_with_laser_on</code> , <code>slsc_list_mark_abs</code>

Name of the function	slsc_list_wait_with_laser_on
Purpose	Defines a waiting time with which the laser spot is to wait at the last defined target point with the laser switched on.
Function signature	<code>uint32_t slsc_list_wait_with_laser_on(size_t Handle, double Time);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Time Duration in which the laser control signal LASERON is switched on. The mirrors remain in their last position. In s.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> slsc_list_wait_with_laser_on is rejected, if Time < 0. Then, the return value indicates that Bit #06 is set (UnplausibleOrUnknownParameter). slsc_list_wait_with_laser_on belongs to the "MIRROR" functions. Therefore, it is relevant for setting output signals, see Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. slsc_list_wait_with_laser_on behaves like a slsc_list_mark_abs with velocity 0. As of syncAXIS-DLL ≥ V1.2.6, the following applies: with MinimalMarkSpeed = 0, the transition from slsc_list_wait_with_laser_on to slsc_list_mark_abs is now seamless because the Sky Writing-like motion (where the Laser is switched off) is omitted there. This makes it possible to start cutting immediately after penetration of the workpiece (for example, sheet metal). If the "Automatic Laser Control" is active and SpotDistance is an "ActiveChannel", see Chapter 2.9.2 "Definition of the Channels and ActiveChannel", page 48, then: slsc_list_suppress_spotdistance_control must be called prior to slsc_list_wait_with_laser_on. slsc_list_wait_with_laser_on can be used, for example, if the quality of the laser spot is to be measured by external sensors. The complementary function of slsc_list_wait_with_laser_on is slsc_list_wait_with_laser_off. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	<pre>// C++ code section for educational purposes only. // Do not execute this code on actual XL SCAN systems without prior modification and simulation! // Observe the safety notices and disclaimer on page 16. size_t JobID = 0; // Create JobID variable, // initialize with 0. slsc_list_begin(Handle, &JobID); // Job start. double Target1[2] = {0, 0}; // Array of size 2 for target point 1. slsc_list_jump_abs(Handle, Target1); // Jump to target point 1. slsc_list_wait_with_laser_on(Handle, 0.5); // As if it would be a mark vector with // velocity 0 => set laser control signal LASERON // and stay at current location // for half a sec. slsc_list_end(Handle); // Job end.</pre>

Name of the function	<code>slsc_list_wait_with_laser_on</code>
Version info	Available as of syncAXIS-DLL V0.11.0. Latest change with syncAXIS-DLL V1.2.6: behavior changed.
References	slsc_list_wait_with_laser_off , slsc_list_mark_abs , slsc_list_suppress_spotdistance_control

Name of the function	<code>slsc_list_write_analog_x</code>
Purpose	Writes a output value to the 12-Bit-analog output port ANALOG OUT1 or ANALOG OUT2 of all RTC6 boards.
Function signature	<code>uint32_t slsc_list_write_analog_x(size_t Handle, slsc_AnalogOutput Channel, double Value, double TimeDelay);</code>
Argument(s)	Handle Handle to a syncAXIS control instance .
	Channel Analog output port ANALOG OUT1 or ANALOG OUT2 ("channel"). = 1: ANALOG OUT1. = 2: ANALOG OUT2. Allowed value range: [1, 2]. See enum slsc_AnalogOutput .
	Value Output value at the ANALOG OUT1 or ANALOG OUT2 analog output port. Value = 0 corresponds to an output value of 0 V. Value = 1 corresponds to an output value of 10 V.
	TimeDelay Relative point in time between 2 Job functions (slsc_list_*), when the change is going to be applied (reference point: point in time at <i>marking execution</i> when the target point of the first Job function is reached, see Figure 14, page 46). In s. Only positive values are allowed.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions", page 279 .
Comment(s)	<ul style="list-style-type: none"> <code>slsc_list_write_analog_x</code> has <i>no</i> effect with activated "Automatic Laser Control", if the analog output port specified at <code>Channel</code> is defined as "ActiveChannel" (see Chapter 2.9.2 "Definition of the Channels and ActiveChannel", page 48), that is, the output value of this analog output port is dictated by the automatic laser control. The ANALOG OUT1 signal is outputted with <ul style="list-style-type: none"> – RTC6 PCI Express Boards (as RTC5 boards): LASER connector, pin 08 The ANALOG OUT2 signal is outputted with <ul style="list-style-type: none"> – RTC6 PCI Express Boards (as RTC5 boards): LASER connector, pin 15, as well as MARKING ON THE FLY socket connector, pin 14 <code>slsc_list_write_analog_x</code> changes the configuration of the specified syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. For excessive <code>TimeDelay</code> values, the change is applied with slsc_list_end at latest.

Name of the function	<code>slsc_list_write_analog_x</code>
Comment(s) (cont'd)	<ul style="list-style-type: none"> The change <ul style="list-style-type: none"> is persistent applies to all Jobs that follow Related RTC6 command: <code>write_da_x_list</code>. However, it does not provide the configuration parameter <code>TimeDelay</code>. <code>slsc_list_write_analog_x</code> belongs to the "SIGNAL" functions. See also Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. The corresponding Control function (<code>slsc_ctrl_*</code>) of <code>slsc_list_write_analog_x</code> is <code>slsc_ctrl_write_analog_x</code>. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	<p>Available as of syncAXIS-DLL V0.11.0.</p> <p>Change with syncAXIS-DLL V1.1.0: data type of <code>Channel</code>.</p> <p>Latest change with syncAXIS-DLL V1.2.4: <code>TimeDelay</code> values for V1.1 are shifted syncAXIS-DLL-internally by a certain delay in V1.2 (for example, in Operation mode "ScannerOnly" by 0.00125 s).</p>
References	<code>slsc_ctrl_write_analog_x</code>

Name of the function	slsc_list_write_digital_out
Purpose	Writes a 16-bit output value to the 16-bit digital output port DIGITAL OUT 0...DIGITAL OUT 15 of all RTC6 boards.
Function signature	<code>uint32_t slsc_list_write_digital_out(size_t Handle, uint16_t Value, double TimeDelay);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Value 16-bit output value (DIGITAL OUT0...DIGITAL OUT15) at the 16-bit digital output port.
	TimeDelay Relative point in time between 2 Job functions (slsc_list_*), when the change is going to be applied (reference point: point in time at <i>marking execution</i> when the target point of the first Job function is reached, see Figure 14, page 46). In s. Only positive values are allowed.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions" , page 279.
Comment(s)	<ul style="list-style-type: none"> The DIGITAL OUT 0...DIGITAL OUT 15 signal is outputted with <ul style="list-style-type: none"> RTC6 PCI Express Boards (as RTC5 boards): EXTENSION 1 socket connector, pin 01...pin 31 (odd-numbered pins only) slsc_list_write_digital_out changes the configuration of the specified syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. For excessive <code>TimeDelay</code> values, the change is applied with slsc_list_end at latest. The change <ul style="list-style-type: none"> is persistent applies to all Jobs that follow Related RTC6 command: write_io_port_list. However, it does not provide the configuration parameter <code>TimeDelay</code>. slsc_list_write_digital_out belongs to the "SIGNAL" functions. See also Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. The corresponding Control function (slsc_ctrl_*) of slsc_list_write_digital_out is slsc_ctrl_write_digital_out. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.
Code example	–
Version info	<p>Available as of syncAXIS-DLL V0.9.0.</p> <p>Change with syncAXIS-DLL V1.1.0: data type of <code>Value</code>.</p> <p>Latest change with syncAXIS-DLL V1.2.4: <code>TimeDelay</code> values for V1.1 are shifted syncAXIS-DLL-internally by a certain delay in V1.2 (for example, in Operation mode "ScannerOnly" by 0.00125 s).</p>
References	slsc_ctrl_write_digital_out, slsc_list_write_digital_out_mask

Name of the function	slsc_list_write_digital_out_mask
Purpose	Writes only those bits of the <i>Value</i> -values to the 16-bit digital output port of all RTC6, which are specified in the user-defined bit mask (<i>Mask</i> parameter).
Function signature	<code>uint32_t slsc_list_write_digital_out_mask(size_t Handle, uint16_t Value, uint16_t Mask, double TimeDelay);</code>
Argument(s)	Handle Handle to a syncAXIS control instance.
	Value 16-bit output value (DIGITAL OUT0 ... DIGITAL OUT15).
	Mask 16-bit mask (for DIGITAL OUT0 ... DIGITAL OUT15).
	TimeDelay Relative point in time between 2 Job functions (<i>slsc_list_*</i>), when the change is going to be applied (reference point: point in time at <i>marking</i> execution when the target point of the first Job function is reached, see Figure 14, page 46). In s. Only positive values are allowed.
Return value	See Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions", page 279.
Comment(s)	<ul style="list-style-type: none"> The DIGITAL OUT 0 ... DIGITAL OUT 15 signal is outputted with <ul style="list-style-type: none"> – RTC6 PCI Express Boards (as RTC5 boards): EXTENSION 1 socket connector, pin 01...pin 31 (odd-numbered pins only) The parameter <i>Mask</i> defines <i>which</i> bits of the 16-bit digital output port (see slsc_list_write_digital_out) are changed, whereas the argument <i>Value</i> defines <i>how</i> they are changed. All bits of the 16-bit digital output port which are not set in <i>Mask</i> remain unchanged. These are outputted again as previously. For <i>Mask</i> = 0xFFFF ("set all bits"), slsc_list_write_digital_out_mask behaves like slsc_list_write_digital_out. slsc_list_write_digital_out_mask changes the configuration of the specified syncAXIS control instance. In the process, the syncAXIS control instance is not reinitialized. For excessive <i>TimeDelay</i> values, the change is applied with slsc_list_end at latest. The change <ul style="list-style-type: none"> – is persistent – applies to all Jobs that follow Related RTC6 command: write_io_port_mask_list. However, it does not provide the configuration parameter <i>TimeDelay</i>. slsc_list_write_digital_out_mask belongs to the "SIGNAL" functions. See also Chapter 2.7.2 "About the Point in Time when Output Signals are actually set", page 45. The corresponding Control function (<i>slsc_ctrl_*</i>) of slsc_list_write_digital_out_mask is slsc_ctrl_write_digital_out_mask. On the permissibility of syncAXIS control functions in the Mode "Manual Positioning", see Chapter 2.12 "About the Mode "Manual Positioning"", page 70.

Name of the function	<code>slsc_list_write_digital_out_mask</code>
Code example	–
Version info	Available as of syncAXIS-DLL V0.9.0. Change with syncAXIS-DLL V1.1.0: data type of Value, Mask. Latest change with syncAXIS-DLL V1.2.4: TimeDelay values for V1.1 are shifted syncAXIS-DLL- internally by a certain delay in V1.2 (for example, in Operation mode “ScannerOnly” by 0.00125 s).
References	<code>slsc_ctrl_write_digital_out_mask</code> , <code>slsc_list_write_digital_out</code>

Name of the function	<code>slsc_util_reset_pcie</code>
Purpose	Carries out a “hard” reset of all found RTC6 PCI Express Boards.
Function signature	<code>uint32_t slsc_util_reset_pcie(const char* PathToProgramFile);</code>
Argument(s)	<code>PathToProgramFile</code> Absolute file path or the relative file path from the execution directory to the RTC6 files. These must be from the syncAXIS control-software package.
Return value	See Chapter 4 “Standard Return Values of the syncAXIS-DLL Functions”, page 279.
Comment(s)	<ul style="list-style-type: none"> <code>slsc_util_reset_pcie</code> is a Utility Function, see also Chapter 3.1.4 “Utility Functions (<code>slsc_util_*</code>)”, page 101. Utility Functions (<code>slsc_util_*</code>) may only be called outside syncAXIS control operation. ⚠ Warning! Risk of injury due to laser radiation! <code>slsc_util_reset_pcie</code> can lead to states of the RTC6 board(s) in which the laser could emit unexpectedly! Make sure that the laser is switched off before calling <code>slsc_util_reset_pcie</code>! <code>slsc_util_reset_pcie</code> is provided for hard reset, if one of the RTC6 PCI Express Boards is in a state that is not fixed by a call of <code>slsc_cfg_initialize_from_file</code> (do not use <code>iSCANcfg.exe</code> here). <code>slsc_util_reset_pcie</code> does not reference a syncAXIS control instance (= does not have an Handle argument). Therefore, no syncAXIS control instance settings are observed, in particular there is no simulation mode. Notice! <code>slsc_util_reset_pcie</code> basically triggers the RTC6 command <code>load_program_file</code> on all found RTC6 PCI Express Boards, even several times in certain circumstances. No consideration is given to running syncAXIS control instances and is crashing these!
Code example	–
Version info	Available as of syncAXIS-DLL V1.3.0.
References	–

4 Standard Return Values of the syncAXIS-DLL Functions

Most⁽¹⁾ syncAXIS-DLL functions provide a return value with uniform meaning ("standard return value"; not to be confused with **Error Codes with slsc_ctrl_get_error, Log File and Console, page 282**).

This return value is an unsigned 32-bit value (scheme in hexadecimal notation: 0x nn nn nn nn, see column **Bit mask** in the following table).

- The return value is 0, if the function was executed successfully ("everything is OK").
- The return value is ... 0, if the function could not have been successfully executed. At the same time, it contains the coded error cause, see following table.

Remarks: return values ... 0 result from the fact that a certain bit is set if a certain error occurs, see the following table. Example: **Bit #03** is set, all remaining are not. Value of **Bit #03** is 8, therefore the resulting **Bit mask** is 0x 00 00 00 08 (**NotAllowedInExecuting**).

With syncAXIS control only one error bit is set at a time. That is, never several bits are set, even if several errors have been occurred.

(1) For example, **slsc_cfg_get_sync_axis_version** does not.

Bit mask	Bit	Short name	Description
0x 00 00 00 00	Bit #00 = 0 (LSB)	OK	The specified function has been successfully executed.
0x 00 00 00 01	Bit #00 = 1 (LSB)	InErrorState	The specified syncAXIS control instance is in an error state. This error state was probably not caused by this very function but already before!
0x 00 00 00 02	Bit #01 = 1	ErrorOccurred	An error has occurred which cannot be specified in more detail.
0x 00 00 00 04	Bit #02 = 1	NotAllowedWithoutInitialization	The specified Handle does not exist. Also: The specified function is not possible at this position (yet). The syncAXIS control instance is not yet initialized. Example: In the program source code, slsc_list_begin is entered before slsc_cfg_initialize_from_file .

Bit mask	Bit	Short name	Description
0x 00 00 00 08	Bit #03 = 1	NotAllowedInExecuting	The specified function (for example, slsc_cfg_initialize_from_file) is not possible at this position because an execution is currently in progress.
0x 00 00 00 10	Bit #04 = 1	BUFFER_FULL	The Input buffer of the specified syncAXIS control instance has no free capacity at the moment. Possible action in the program code: send the desired function once again after a short waiting loop. See also Chapter 2.7.1 "About the Buffers of the syncAXIS control Instances" , page 42.
0x 00 00 00 20	Bit #05 = 1	NotReadyForExecution	It was tried to start a Job execution. However, (from the perspective of the syncAXIS control instance) the RTC6 board is not ready for execution.
0x 00 00 00 40	Bit #06 = 1	UnplausibleOrUnknownParameter	The specified function has been tried to apply a parameter that is not possible in this way. Example: slsc_list_set_mark_speed(5000) , but the maximum speed of the galvanometer scanner is actually only 500.
0x 00 00 00 80	Bit #07 = 1	JobStructureNotValid	The Job does not have a valid structure.
0x 00 00 01 00	Bit #08 = 1	Undefined	Reserved.
0x 00 00 02 00	Bit #09 = 1	NotAllowedInCurrentConfiguration	The specified function is not allowed with the current configuration.
0x 00 00 04 00	Bit #10 = 1	Undefined	Reserved.
0x 00 00 08 00	Bit #11 = 1	NotAllowedInCurrentMode	The specified function is not allowed in the current Operation mode .
0x 00 00 10 00	Bit #12 = 1	InvalidPosition	The specified function fails due to an invalid position. Example: at slsc_ctrl_start_execution , the current position of the positioning stage does not match the position calculated for the begin of the Job .
0x 00 00 20 00	Bit #13 = 1	Timeout	The specified function exceeds a specified time limit and therefore fails.
0x 00 00 40 00	Bit #14 = 1	XmlLoadError	XML file not found or is invalid.
0x 00 00 80 00	Bit #15 = 1	NotEnoughMemory	The initialization of the syncAXIS control instance failed because there was not enough free memory on the PC.
0x 00 01 00 00	Bit #16 = 1	Undefined	Reserved.
0x 00 02 00 00	Bit #17 = 1	Undefined	Reserved.
0x 00 04 00 00	Bit #18 = 1	HandshakeFailed	The initialization of the positioning stage failed.

Bit mask	Bit	Short name	Description
0x 00 08 00 00	Bit #19 = 1	Undefined	Reserved.
0x 00 10 00 00	Bit #20 = 1	UnknownException	An unknown exception occurred during the execution of the specified function.
0x 00 20 00 00	Bit #21 = 1	Undefined	Reserved.
0x 00 40 00 00	Bit #22 = 1	Undefined	Reserved.
0x 00 80 00 00	Bit #23 = 1	Undefined	Reserved.
0x 01 00 00 00	Bit #24 = 1	Undefined	Reserved.
0x 02 00 00 00	Bit #25 = 1	Undefined	Reserved.
0x 04 00 00 00	Bit #26 = 1	Undefined	Reserved.
0x 08 00 00 00	Bit #27 = 1	Undefined	Reserved.
0x 10 00 00 00	Bit #28 = 1	UnknownDevice	The specified device is not known.
0x 20 00 00 00	Bit #29 = 1	Undefined	Reserved.
0x 40 00 00 00	Bit #30 = 1	MaxInstancesReached	An additional syncAXIS control instance cannot be created The max. number of syncAXIS control instances which are concurrently allowed on a PC is coded on the Dongle .
0x 80 00 00 00	Bit #31 = 1 (MSB)	InvalidOrMissingDongle	The Dongle is either not valid for syncAXIS control or not plugged-in.



5 Error Codes with `slsc_ctrl_get_error`, Log File and Console

- See also [Chapter 4 "Standard Return Values of the syncAXIS-DLL Functions"](#), page 279.

The following error codes

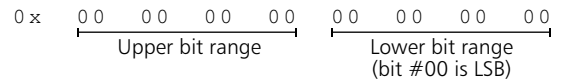
- 0x 00 00 00 02 00 00 00 01
EXEC_AUTOSTOP
- 0x 00 00 00 02 00 00 00 02
EXEC_BUFFER_UNDERRUN
- 0x 00 00 00 06 00 00 00 01
INIT_ACS_TCPIP

refer to:

- `slsc_ctrl_get_error`, argument `ErrorCode`
- Log file (`EnableFileLogging` true), see also **Chapter 2.8 “About the Logging in syncAXIS control”**, page 47
- Console (`EnableConsoleLogging` true)

Notes

- For the specific errors the error constants should be predefined.
- For interpretation of the 64 bit values:



The values of the upper 32 bits (Bit #63...32) determine the meaning of the lower 32 bits (Bit #31...Bit #00 = LSB).

Example: 0x00 00 00 00 nn nn nn nn

=> In the lower bit range, an RTC6 error is coded. Several bits are set, if multiple errors occurred (Note: other than with the RTC control command **get_error**, here *no* cumulative error code is returned!).

- With syncAXIS-DLL *no* error codes are generated where simultaneously RTC errors and syncAXIS control errors are coded.

Code	0x 00 00 00 02 00 00 00 01
Error constant	EXEC_AUTOSTOP
Keyword sequence	Autostop detected at master. Headstatus NOK! MarkingInfo-Flags ...
Cause	<ul style="list-style-type: none"> 1. Cause in context "Initializing the syncAXIS control instance" At the respective axis, there is no drive connected or it is not powered. => See Remedy on 1. Cause in context "Initializing the syncAXIS control instance". 2. Cause in context "running operation" and "emergency stop" During running operation, an "emergency stop" has been triggered by the user (kill switch), Laser, scan device or positioning stage controller. => See Remedy on 2. Cause in context "running operation" and "emergency stop". 3. Cause in context "running operation" and non-adherence to the positioning stage limits on ACS-side During running operation, the ACS Motion Controller has detected an exceedance of the set positioning stage limits. => See Remedy on 3. Cause in context "running operation" and non-adherence to the positioning stage limits on ACS-side. 4. Cause in context "running operation" and EXEC_BUFFER_UNDERRUN EXEC_BUFFER_UNDERRUN usually triggers an EXEC_AUTOSTOP as well. In the case of an EXEC_BUFFER_UNDERRUN the positioning stage is not decelerated in a controlled manner. Instead, the positioning stage is simply "hard stopped" which is associated with the violation of positioning stage dynamic limits. => See Remedy on 4. Cause in context "running operation" and EXEC_BUFFER_UNDERRUN. 5. Cause in context "running operation" and scan head-related Fehler During running operation, the scan head does not send the PWROK signal ("Power OK"), see excelliSCAN Manual. => See Remedy on 5. Cause in context "running operation" and scan head-related Fehler.

Code	0x 00 00 00 02 00 00 00 01
Remedy	<ul style="list-style-type: none"> • Remedy on 1. Cause in context “Initializing the syncAXIS control instance” <ol style="list-style-type: none"> (1) Read out ACS error in ACS SPiiPlus MMI Application Studio. It reads roughly: “Axis <n> disabled: Error 5027 (Motion termination error), Servo Processor Alarm”. (2) Put the affected axis into operation with ACS SPiiPlus MMI Application Studio. (3) Make sure that axes configuration is correct in <code>syncAXISConfig.xml</code>, for example, <pre><cfg:StageAxisX>0</cfg:StageAxisX> <cfg:StageAxisY>1</cfg:StageAxisY> <cfg:SlecEtherCATNodeID>0</cfg:SlecEtherCATNodeID></pre> • Remedy on 2. Cause in context “running operation” and “emergency stop” <ol style="list-style-type: none"> (1) Read out ACS error in ACS SPiiPlus MMI Application Studio. It reads roughly: “Error 5028 (Motion Termination error), Safe Torque Off” (2) Reset the effect of the emergency stop switch according to your safety concept. (3) Initialize the syncAXIS control instance again. • Remedy on 3. Cause in context “running operation” and non-adherence to the positioning stage limits on ACS-side <ol style="list-style-type: none"> (1) syncAXIS-DLL checks the dynamic limits and responds as set in <code>DynamicViolationReaction</code>. With <code>WarningOnly</code>, a [WARN] log file line is written, see [WARN] log file lines, by which you can see if the exceedance has been detected by syncAXIS-DLL. With <code>AbortImmediately</code> and <code>StopAndReport</code>, the Job is cancelled without you getting an error from syncAXIS-DLL. (2) Read out ACS error in ACS SPiiPlus MMI Application Studio. With “Error 5015 (Motion termination error), Software Right Limit” the maximum allowed position has been exceeded <ul style="list-style-type: none"> • With “Error 5076 (Motion termination error), Driver Alarm: Drive Saturation” or “Error 5023 (Motion termination error), Critical Position Error” the velocity limit has been exceeded • With “Error Error 5076 (Motion termination error), Driver Alarm: Drive Saturation” or “Error 5023 (Motion termination error), Critical Position Error” the acceleration limit has been exceeded • With “Error 5023 (Motion termination error), Critical Position Error” the jerk limit has been exceeded (3) If one of the errors mentioned in (2) is the cause of the <code>EXEC_AUTOSTOP</code>, you may possibly (only within the safe zone and depending on the process requirements) increase the maximum allowed position error in ACS Motion Controller.

Code	0x 00 00 00 02 00 00 00 01
Remedy (cont'd)	<p>(4) Otherwise you have to adjust the Job. In simulation mode (see Chapter 2.5 "About the syncAXIS control Simulation Mode", page 31) you must first identify at which point of the Job the dynamic limits are violated. As described in Chapter 2.6 "About Optimizing syncAXIS control-based User Programs", page 36, you have to make sure that the violation no longer occurs by iteratively changing the values for velocities, dynamics and <code>FilterBandwidth</code>.</p> <ul style="list-style-type: none"> • Remedy on 4. Cause in context "running operation" and EXEC_BUFFER_UNDERRUN Since <code>EXEC_BUFFER_UNDERRUN</code> and <code>EXEC_AUTOSTOP</code> are often associated, see Section "Avoiding Buffer Underruns", page 42. • Remedy on 5. Cause in context "running operation" and scan head-related Fehler <ul style="list-style-type: none"> – Check whether the scan head answers using <code>iSCANcfg.exe</code> from the syncAXIS control-software package. If it does not, check the cabling. – Contact SCANLAB depending on the returned scan head error – If the scan head temperature was too high, redefine the Job with lower dynamic requirements. – Improve the cooling and the thermal connectivity of the scan head.
Comment(s)	An indication of the error cause may give the MarkingInfo flags that are also outputted. These correspond to the return value of the RTC6 command <code>get_marking_info</code> . Example: If flags are set for the scan head port to which the positioning stage is connected, then this indicates a problem in the control of this positioning stage.

Code	0x 00 00 00 02 00 00 00 02
Error constant	EXEC_BUFFER_UNDERRUN
Keyword sequence	Buffer underrun detected on RTC ...
Cause	Buffer underrun, page 11.
Remedy	See Section "Avoiding Buffer Underruns", page 42.
Comment(s)	<ul style="list-style-type: none"> EXEC_BUFFER_UNDERRUN usually triggers an EXEC_AUTOSTOP as well. In the case of an EXEC_BUFFER_UNDERRUN the positioning stage is <i>not</i> decelerated in a controlled manner. Instead, the positioning stage is simply "hard stopped" which is associated with the violation of positioning stage dynamic limits.

Code	0x 00 00 00 06 00 00 00 01
Error constant	INIT_ACS_TCPIP
Keyword sequence	Establishing TCP/IP communication to the ACS controller failed ...
Cause	<ul style="list-style-type: none"> 1. Cause The entry in <code>syncAXISConfig.xml</code> under <code><cfg:Configuration></code> → <code><cfg:GeneralConfig></code> → <code><cfg:ACSController></code> is missing or wrong. => See Remedy on 1. Cause. 2. Cause Bootling of ACS Motion Controller is not yet complete. => See Remedy on 2. Cause. 3. Cause Connection to the ACS Motion Controller is faulty. => See Remedy on 3. Cause. 4. Cause In ACS SPiiPlus MMI Application Studio, "Network Error" messages are opened. => See Remedy on 4. Cause. 5. Cause Problems with ACS User Mode Driver (UMD). => See Remedy on 5. Cause. 6. Cause XL SCAN option is not enabled on ACS Motion Controller. => See Remedy on 6. Cause.
Remedy	<ul style="list-style-type: none"> Remedy on 1. Cause (1) Identify the ACS Motion Controller IP address with ACS SPiiPlus MMI Application Studio (for example, #SI" in ACS Communication Terminal). (2) Correct the entry in <code>syncAXISConfig.xml</code> under <code><cfg:Configuration></code> → <code><cfg:GeneralConfig></code> → <code><cfg:ACSController></code>. Remedy on 2. Cause (1) A restart can take up to 5 minutes. Wait accordingly. (2) If the error persists: See Remedy on 3. Cause.

Code	0x 00 00 00 06 00 00 00 01
Remedy (cont'd)	<ul style="list-style-type: none"> • Remedy on 3. Cause <ol style="list-style-type: none"> (1) Check, if a connection to ACS Motion Controller via ACS SPiiPlus MMI Application Studio is possible. (2) If not: Find and remove errors concerning the network connection. Possible causes: Defective hardware (network card, network cable, router, WLAN components), wrong configuration, blocking by a firewall. (3) If you still cannot establish a connection: Contact ACS Support. • Remedy on 4. Cause <ol style="list-style-type: none"> (1) Try to trace the error with “#LOG” in ACS Communication Terminal. (2) Find and remove errors concerning the network connection. Possible causes: Defective hardware (network card, network cable, router, WLAN components), wrong configuration, blocking by a firewall. (3) In ACS SPiiPlus MMI Application Studio, increase the “Communication Timeout”. (4) If the “Network Error” still persists: Contact ACS support. • Remedy on 5. Cause <ul style="list-style-type: none"> – Restart ACS User Mode Driver (UMD). • Remedy on 6. Cause <ol style="list-style-type: none"> (1) Type “#SI” in ACS Communication Terminal. (2) “XL SCAN units” must be > 0. If “XL SCAN units” = 0: Contact ACS Support and request an upgrade.
Comment(s)	<ul style="list-style-type: none"> • –

6 Structures

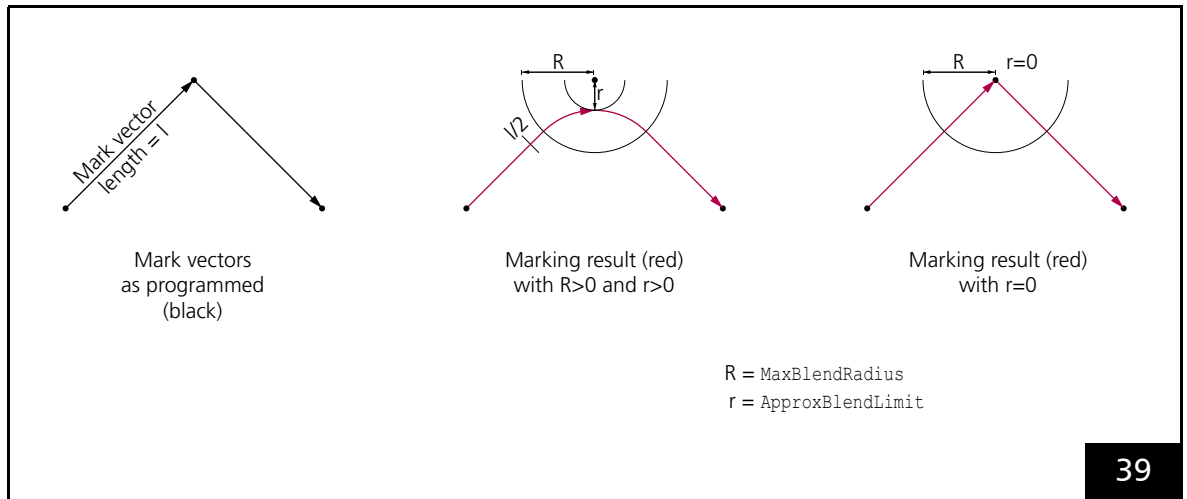
In this chapter:

- Structure **slsc_GeometryConfig**
- Structure **slsc_MarkConfig**
- Structure **slsc_MultiParaTarget**
- Structure **slsc_ParaSection**
- Structure **slsc_TrajectoryConfig**
- Structure **VersionInfo**

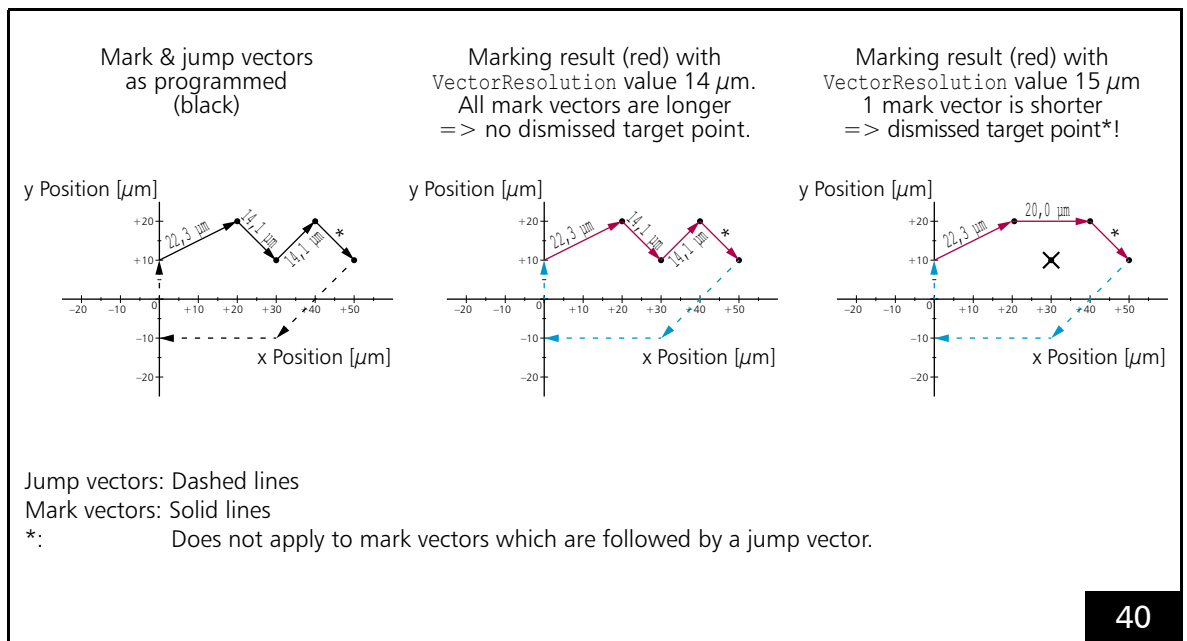
Name of the structure	slsc_GeometryConfig	
Description	This structure defines (as part of the Trajectory planning configuration) the behavior of the blending curves (\leq V1.4.0: of splines as well).	
Used by	This structure is used with: <ul style="list-style-type: none"> • Structure slsc_TrajectoryConfig 	
Syntax	<pre> struct slsc_GeometryConfig { double MaxBlendRadius; double ApproxBlendLimit; slsc_BlendModes BlendMode; double VectorResolution; bool AutoCyclicGeometry; double SplineConversionLengthLimit; slsc_SplineModes SplineMode; }; </pre>	
Argument(s)	double MaxBlendRadius	<p>See 'R' in Figure 39, page 292: Radius of the circle around the corner point (between two vectors) in which the blending curve is to be positioned. In the marking result, blending curves are also limited by the vector length: $R_{actual} = \min(R, l/2)$.</p> <p>If the values specified with <code>MaxBlendRadius</code> and/or <code>ApproxBlendLimit</code> cannot be adhered to, then the syncAXIS control instance executes a jump (similar but different as with RTC-Sky Writing). Important: For <code>BlendMode = slsc_BlendModes_MinimalBlending</code>, the <code>MaxBlendRadius</code> value is used as the limit for the blending curve start.</p> <p>The corresponding <code>syncAXISConfig.xml</code> tag is <code>MaxBlendRadius</code>.</p>

Name of the structure	slsc_GeometryConfig		
Argument(s) (cont'd)	double	ApproxBlendLimit	<p>See 'r' in Figure 39, page 292: maximum tolerable mathematical distance of the blending curve to the corner point.</p> <p>If the values specified with <code>MaxBlendRadius</code> and/or <code>ApproxBlendLimit</code> cannot be adhered to, then the syncAXIS control instance executes a jump (similar but different as with RTC-Sky Writing).</p> <p>By slsc_list_set_approx_blend_limit, the <code>ApproxBlendLimit</code> value can be changed for a particular Job. The change applies as of the insert position but only until the end of the currently running Job.</p> <p>The corresponding <code>syncAXISConfig.xml</code> tag is <code>ApproxBlendLimit</code>.</p>
	slsc_BlendModes	BlendMode	The Blend mode to be applied.
	double	VectorResolution	<p>See Figure 40, page 292: If the target points of two consecutive Mark functions (slsc_list_mark_abs, slsc_list_multi_para_mark_abs, slsc_list_para_mark_abs) have a smaller distance than the <code>VectorResolution</code> value, then these target points are regarded by the syncAXIS control instance as identical. That is, with Mark vector → Mark vector sequences, target points are dismissed under certain circumstances.</p> <p>The intended use of <code>VectorResolution</code> is as follows: users shall be able to specify target points with a certain input inaccuracy (for example, with floating points or, if data is automatically read-in). Therefore, reasonable values are in the micrometer range, for example 0.02 mm.</p> <p>The corresponding <code>syncAXISConfig.xml</code> tag is <code>VectorResolution</code>.</p>

Name of the structure	slsc_GeometryConfig	
Argument(s) (cont'd)	<code>bool</code> <code>AutoCyclicGeometry</code>	<p>Deprecated. Only use <code>false</code>.</p> <p>The corresponding <code>syncAXISConfig.xml</code> tag is <code>AutoCyclicGeometry</code>.</p>
	<code>double</code> <code>SplineConversionLengthLimit</code>	<p>Deprecated. Only use / leave the default value.</p> <p>The corresponding <code>syncAXISConfig.xml</code> tag is <code>SplineConversionLengthLimit</code>.</p>
	<code>slsc_SplineModes</code> <code>SplineMode</code>	<p>Deprecated. Only use <code>slsc_SplineModes_Deactivated</code>.</p> <p>The corresponding <code>syncAXISConfig.xml</code> tag is <code>SplineMode</code>.</p>
Comment(s)	<ul style="list-style-type: none"> In the syncAXIS-DLL, there is the following priority in regards to the calculation of trajectories: blending curve to corner point > Sky Writing-like motion. 	
Version info	Available as of syncAXIS-DLL V0.11.0.	



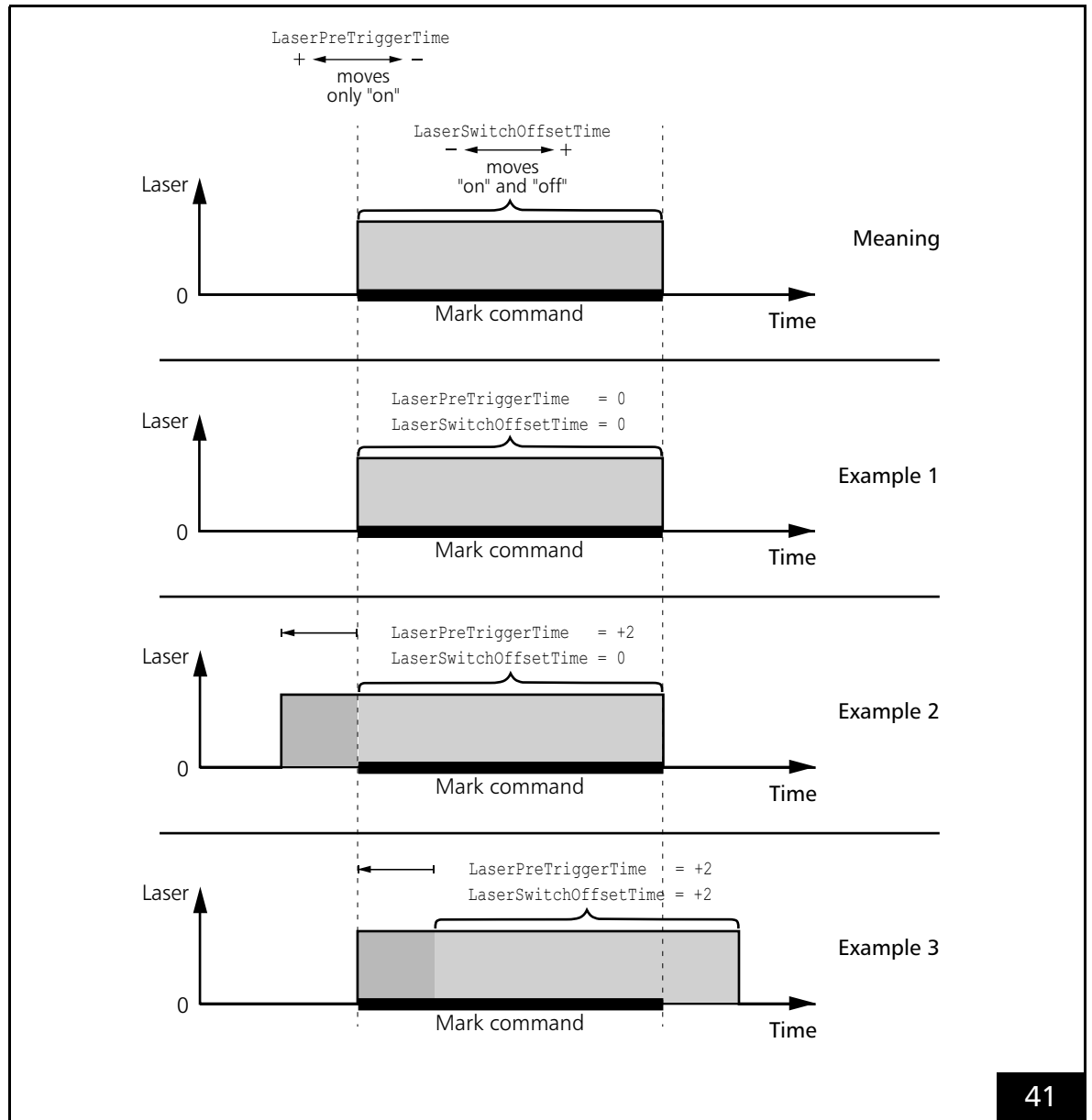
Structure **slsc_GeometryConfig**: On the mode of action of the arguments **MaxBlendRadius** and **ApproxBlendLimit**.



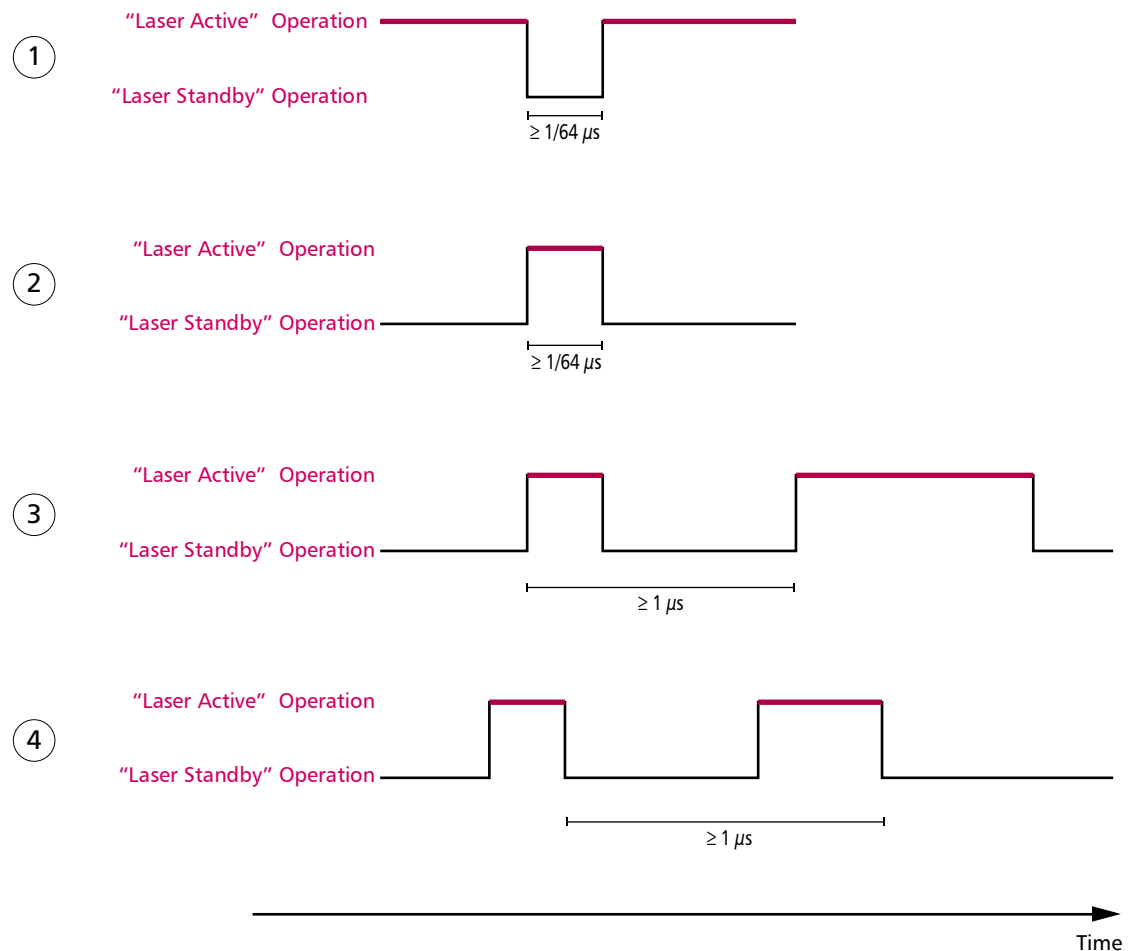
Structure **slsc_GeometryConfig**: On the mode of action of the argument **VectorResolution** in $\geq V1.2$.

Name of the structure	slsc_MarkConfig	
Description	This structure defines the basic settings of the Trajectory planning for markings, for example, laser switching times.	
Used by	This structure is used with: <ul style="list-style-type: none"> Structure slsc_TrajectoryConfig 	
Syntax	<pre> struct slsc_MarkConfig { double LaserPreTriggerTime; double LaserSwitchOffsetTime; double LaserMinOffTime; double JumpSpeed; double MarkSpeed; double MinimalMarkSpeed; }; </pre>	
Argument(s)	double LaserPreTriggerTime	See Figure 41, page 295 : Time to trigger the laser signal in advance, if a Mark Segment is executed. Unit: s. The corresponding <code>syncAXISConfig.xml</code> tag is <code>LaserPreTriggerTime</code> . Related RTC6 command: set_laser_delays (similarities to LaserOn delay and LaserOff delay exist, see RTC6 Manual, Chapter 7.2.1 "Laser Delays", page 144).
	double LaserSwitchOffsetTime	See Figure 41, page 295 : Time shift for the laser signals output. Unit: s. The corresponding <code>syncAXISConfig.xml</code> tag is <code>LaserSwitchOffsetTime</code> . Related RTC6 command: set_laser_delays (similarities to LaserOn delay and LaserOff delay exist, see RTC6 Manual, Chapter 7.2.1 "Laser Delays", page 144).
	double LaserMinOffTime	See Figure 42, page 296 ((1)) : Shortest "Laser Standby" Operation duration. Unit: s. The corresponding <code>syncAXISConfig.xml</code> tag is <code>LaserMinOffTime</code> .

Name of the structure	slsc_MarkConfig		
Argument(s) (cont'd)	double	JumpSpeed	Desired combined jump speed (that is, combined scanner and positioning stage motion) during Job execution. In StageOnly mode, this speed is strictly adhered to as maximum speed, otherwise the value may be slightly exceeded temporarily. As of $\geq V1.5.0$, the following applies: In Operation mode "StageOnly", this speed is strictly adhered to as the maximum speed. In the Operation mode "ScannerOnly" and "ScannerAndStage", the value can even be temporarily exceeded slightly. The corresponding <code>syncAXISConfig.xml</code> tag is <code>JumpSpeed</code> .
	double	MarkSpeed	The highest desired combined marking speed (that is, combined scanner and positioning stage motion) during Job execution. There may be cases with blending curves at corner points where the <code>syncAXIS control instance</code> must apply a lower speed. Then at least the value of <code>MinimalMarkSpeed</code> is used. The corresponding <code>syncAXISConfig.xml</code> tag is <code>MarkSpeed</code> .
	double	MinimalMarkSpeed	The lowest desired spot velocity that shall be reached in corners of the contour. With blending curves at corner points – in order to make rounded edges possible – the <code>syncAXIS control instance</code> is allowed to slow down to this speed. See <code>MarkSpeed</code> . If a exact defined corner blending cannot be performed with the known scanner and positioning stage dynamics with at least this minimal marking speed, a Sky Writing-like motion is executed. The corresponding <code>syncAXISConfig.xml</code> tag is <code>MinimalMarkSpeed</code> .
Comment(s)	• –		
Version info	Available as of syncAXIS-DLL V0.11.0.		



Structure **slsc_MarkConfig**: On the mode of action of the arguments **LaserPreTriggerTime** and **LaserSwitchOffsetTime**.



Legend

1. The shortest "Laser Standby" Operation duration (and thus the shortest duration of **Jump Segments**) is:
 - The `LaserMinOffTime` value
 - syncAXIS-DLL extends **Jump Segments** and Sky Writing-like motions accordingly, if they are shorter than specified above. The smallest allowed `LaserMinOffTime` value is $1/64 \mu s$.
2. The shortest "Laser Active" Operation duration (and thus the shortest duration of **Mark Segments**) is:
 - Always $1/64 \mu s$
3. "Laser Active" Operation starts must always be at least $1 \mu s$ apart
Otherwise, the following applies:
 - With **[*]dashed[*] Functions**, syncAXIS-DLL rejects them
 - With **Jump Segments**, syncAXIS-DLL extends their duration accordingly
4. "Laser Standby" Operation starts must always be at least $1 \mu s$ apart
Otherwise, the following applies:
 - Same as 3

Name of the structure	slsc_MultiParaTarget		
Description	This structure defines a Ramp , which consists of several sections (these are defined with slsc_ParaSection).		
Used by	This structure is used with: <ul style="list-style-type: none"> • slsc_list_multi_para_arc_abs • slsc_list_multi_para_circle_2d_abs • slsc_list_multi_para_dashed_arc_abs • slsc_list_multi_para_dashed_circle_2d_abs • slsc_list_multi_para_dashed_mark_abs • slsc_list_multi_para_mark_abs 		
Syntax	<pre>struct slsc_MultiParaTarget { slsc_ParaSection* Targets; size_t NumParaTargets; };</pre>		
Argument(s)	slsc_ParaSection*	Targets	Pointer to an array of variable size (which is specified in NumParaTargets).
	size_t	NumParaTargets	Number of elements in Targets.
Comment(s)	<ul style="list-style-type: none"> • See also Section “About Ramps”, page 53. 		
Version info	Available as of syncAXIS-DLL V0.11.0.		

Name of the structure	slsc_ParaSection	
Description	This structure defines one section of the Ramp (which is defined by slsc_MultiParaTarget).	
Used by	This structure is used with: <ul style="list-style-type: none"> Structure slsc_MultiParaTarget 	
Syntax	<pre>struct slsc_ParaSection { double ds; double MultiParaTarget; };</pre>	
Argument(s)	double ds	Length of the section (with respect to the vector length/arc length). 0 is also allowed (=step change). In mm.
	double MultiParaTarget	Factor Ip at the end of ds. No unit. On Factor Ip , see Section "About how ActiveChannel Values along a Contour are Calculated" , page 51.
Comment(s)	<ul style="list-style-type: none"> This structure is also suitable for defining Ramps with sawtooth and square profile, see also Section "About Ramps", page 53. 	
Version info	Available as of syncAXIS-DLL V0.11.0.	

Name of the structure	slsc_TrajectoryConfig		
Description	This structure defines the configuration of the Trajectory planning.		
Used by	This structure is used with: <ul style="list-style-type: none"> • slsc_cfg_delete_trajectory_config • slsc_cfg_get_trajectory_config • slsc_cfg_set_trajectory_config 		
Syntax	<pre>struct slsc_TrajectoryConfig { slsc_MarkConfig MarkConfig; slsc_GeometryConfig GeometryConfig; };</pre>		
Argument(s)	slsc_MarkConfig	MarkConfig	Configures the basic settings for markings.
	slsc_GeometryConfig	GeometryConfig	Configures the behavior of blending curves (≤ V1.4.0: of splines as well).
Comment(s)	• –		
Version info	Available as of syncAXIS-DLL V0.11.0.		

Name of the structure	VersionInfo		
Description	This structure defines the three number blocks of the syncAXIS-DLL version ("Version n.n.n").		
Used by	This structure is used with: <ul style="list-style-type: none"> • slsc_cfg_get_sync_axis_version 		
Syntax	<pre>struct VersionInfo { uint32_t Major; uint32_t Minor; uint32_t Revision; };</pre>		
Argument(s)	uint32_t	Major	Major version of the syncAXIS-DLL.
	uint32_t	Minor	Minor version of the syncAXIS-DLL.
	uint32_t	Revision	Revision of the syncAXIS-DLL.
Comment(s)	• –		
Version info	Available as of syncAXIS-DLL V0.1.0.		

7 Enumerated Types enum

In this chapter:

- enum **slsc_AnalogOutput**
- enum **slsc_BlendModes**
- enum **slsc_DynamicsMonitoringLevel**
- enum **slsc_DynamicViolationReaction**
- enum **slsc_ExecState**
- enum **slsc_JobCharacteristic**
- enum **slsc_ListHandlingMode**
- enum **slsc_MeasurementSignal**
- enum **slsc_OperationMode**
- enum **slsc_OperationStatus**
- enum **slsc_PositionType**
- enum **slsc_ScanDevice**
- enum **slsc_SimulationSetting**
- enum **slsc_SplineModes**
- enum **slsc_Stage**

Name of the enum	slsc_AnalogOutput	
Description	This enum defines the choices for: <ul style="list-style-type: none"> The analog output ports 	
Used by	This enum is used with: <ul style="list-style-type: none"> slsc_ctrl_write_analog_x slsc_list_write_analog_x 	
Syntax	<pre>enum slsc_AnalogOutput { slsc_AnalogOutput_1 =0, slsc_AnalogOutput_2 =1, };</pre>	
Enumeration constant(s)	slsc_AnalogOutput_1	Analog output port 1.
	slsc_AnalogOutput_2	Analog output port 2.
Version info	Available as of syncAXIS-DLL V0.11.0.	

Name of the enum	slsc_BlendModes	
Description	This enum defines the choices for: <ul style="list-style-type: none"> The blend mode 	
Used by	This enum is used with: <ul style="list-style-type: none"> slsc_GeometryConfig 	
Syntax	<pre>enum slsc_BlendModes { slsc_BlendModes_Deactivated = 0, slsc_BlendModes_VariableBlending = 1, slsc_BlendModes_MinimalBlending = 2, slsc_BlendModes_FixedBlending = 3, };</pre>	
Enumeration constant(s)	slsc_BlendModes_Deactivated	This parameter is effective with successions of: <ul style="list-style-type: none"> mark vector → mark vector mark vector → circle circle → mark vector The syncAXIS-DLL inserts a Sky Writing-like motion at each of these successions ("corner point") which the system cannot travel (that is, if the dynamic values there were no longer in the permissible range). See syncAXISConfig.xml tag BlendMode . Mandatory notation, if used there: Deactivated
	slsc_BlendModes_VariableBlending	In the latest syncAXIS-DLL, this parameter is only effective with successions of mark vector → mark vector. The syncAXIS-DLL tries to insert a blending curve at each corner point of mark vector → mark vector. These blending curves are always calculated to have the <i>greatest</i> distance. That is, with MaxBlendRadius (with structure slsc_GeometryConfig) the value min (R, 1/2) is always <i>observed</i> . slsc_BlendModes_VariableBlending has a shorter computing time than slsc_BlendModes_MinimalBlending. See syncAXISConfig.xml tag BlendMode . Mandatory notation, if used there: VariableBlending

Name of the enum	slsc_BlendModes	
Enumeration constant(s) (cont'd)	slsc_BlendModes_MinimalBlending	<p>In the latest syncAXIS-DLL, this parameter is only effective with successions of mark vector → mark vector.</p> <p>The syncAXIS-DLL tries to insert a blending curve at each corner point of mark vector → mark vector. These blending curves are always calculated to have the <i>smallest, but still executable</i> distance.</p> <p>slsc_BlendModes_MinimalBlending has a longer computing time than slsc_BlendModes_VariableBlending.</p> <p>See syncAXISConfig.xml tag BlendMode. Mandatory notation, if used there:</p> <p>MinimalBlending</p>
	slsc_BlendModes_FixedBlending	<p>Deprecated.</p> <p>In the latest syncAXIS-DLL, this parameter is only effective with successions of:</p> <p>mark vector → mark vector</p> <p>syncAXIS-DLL converts these consecutive lines into a composite spline. Note, this is a totally different behavior as with slsc_BlendModes_VariableBlending and slsc_BlendModes_MinimalBlending).</p> <p>See syncAXISConfig.xml tag BlendMode. Mandatory notation, if used there:</p> <p>FixedBlending</p>
Version info	Available as of syncAXIS-DLL V0.4.2.	

Name of the enum	slsc_DynamicsMonitoringLevel	
Description	This enum defines the choices for: <ul style="list-style-type: none"> The dynamics parameters to be monitored 	
Used by	This enum is used with: <ul style="list-style-type: none"> slsc_cfg_get_scan_device_dynamic_monitoring_level slsc_cfg_get_stage_dynamic_monitoring_level slsc_cfg_set_scan_device_dynamic_monitoring_level slsc_cfg_set_stage_dynamic_monitoring_level 	
Syntax	<pre>enum slsc_DynamicsMonitoringLevel { slsc_DynamicsMonitoringLevel_Deactivated =0, slsc_DynamicsMonitoringLevel_Position =1, slsc_DynamicsMonitoringLevel_Velocity =2, slsc_DynamicsMonitoringLevel_Acceleration =3, slsc_DynamicsMonitoringLevel_Jerk =4, };</pre>	
Enumeration constant(s)	slsc_DynamicsMonitoringLevel_Deactivated	No monitoring is to take place. See syncAXISConfig.xml tag MonitoringLevel . Mandatory notation, if used there: Deactivated
	slsc_DynamicsMonitoringLevel_Position	Exceedances of X and Y position values (working field limits) are to be monitored. See syncAXISConfig.xml tag MonitoringLevel . Mandatory notation, if used there: Position
	slsc_DynamicsMonitoringLevel_Velocity	Like slsc_DynamicsMonitoringLevel_Position and additionally: exceedances of velocity (dynamic limits) are to be monitored. See syncAXISConfig.xml tag MonitoringLevel . Mandatory notation, if used there: Velocity
	slsc_DynamicsMonitoringLevel_Acceleration	Like slsc_DynamicsMonitoringLevel_Velocity and additionally: exceedances of acceleration (dynamic limits) are to be monitored. See syncAXISConfig.xml tag MonitoringLevel . Mandatory notation, if used there: Acceleration
	slsc_DynamicsMonitoringLevel_Jerk	Like slsc_DynamicsMonitoringLevel_Acceleration and additionally: exceedances of jerk (dynamic limits) are to be monitored. See syncAXISConfig.xml tag MonitoringLevel . Mandatory notation, if used there: Jerk
Version info	Available as of syncAXIS-DLL V1.5.0.	

Name of the enum	slsc_DynamicViolationReaction
Description	<p>This enum defines the choices for:</p> <ul style="list-style-type: none"> The reaction when a limit value exceedance occurs
Used by	<p>This enum is used with:</p> <ul style="list-style-type: none"> slsc_cfg_get_dynamic_violation_reaction slsc_cfg_set_dynamic_violation_reaction
Syntax	<pre>enum slsc_DynamicViolationReaction { slsc_DynamicViolationReaction_WarningOnly =0, slsc_DynamicViolationReaction_AbortImmediately =1, slsc_DynamicViolationReaction_StopAndReport =2, };</pre>
Enumeration constant(s)	<p>slsc_DynamicViolationReaction_WarningOnly [WARN] log file lines are to be generated only. See syncAXISConfig.xml tag DynamicViolationReaction. Mandatory notation, if used there:</p> <p>WarningOnly</p>
	<p>slsc_DynamicViolationReaction_AbortImmediately It is to be aborted immediately. This will (especially in Operation mode "StageOnly" and "ScannerAndStage") cause the process to abort ("emergency stop"). ACS control (if correspondingly configured) enters an error state. See syncAXISConfig.xml tag DynamicViolationReaction. Mandatory notation, if used there:</p> <p>AbortImmediately</p>
	<p>slsc_DynamicViolationReaction_StopAndReport Before aborting, an attempt is first made to perform a deceleration movement in order to stop (This deceleration movement is visible when the corresponding simulation file is displayed in syncAXIS Viewer.). Then, the syncAXIS control instance enters an error state. Therefore, users must reinitialize the syncAXIS control instance. In contrast, the ACS control usually does not enter an error state. See syncAXISConfig.xml tag DynamicViolationReaction. Mandatory notation, if used there:</p> <p>StopAndReport</p>
Version info	Available as of syncAXIS-DLL V1.5.0.

Name of the enum	slsc_ExecState
Description	<p>This enum defines the choices for:</p> <ul style="list-style-type: none"> The status of the Execution Layer
Used by	<p>This enum is used with:</p> <ul style="list-style-type: none"> slsc_ctrl_get_exec_state
Syntax	<pre>enum slsc_ExecState { slsc_ExecState_Idle =0, slsc_ExecState_ReadyForExecution =1, slsc_ExecState_Executing =2, slsc_ExecState_NotInitOrError =3, };</pre>
Enumeration constant(s)	slsc_ExecState_Idle The RTC6 board is idle.
	slsc_ExecState_ReadyForExecution The RTC6 board is ready to execute Jobs .
	slsc_ExecState_Executing The RTC6 board is executing a Job .
	slsc_ExecState_NotInitOrError The RTC6 board is not initialized or an error is present.
Version info	Available as of syncAXIS-DLL V0.11.0.

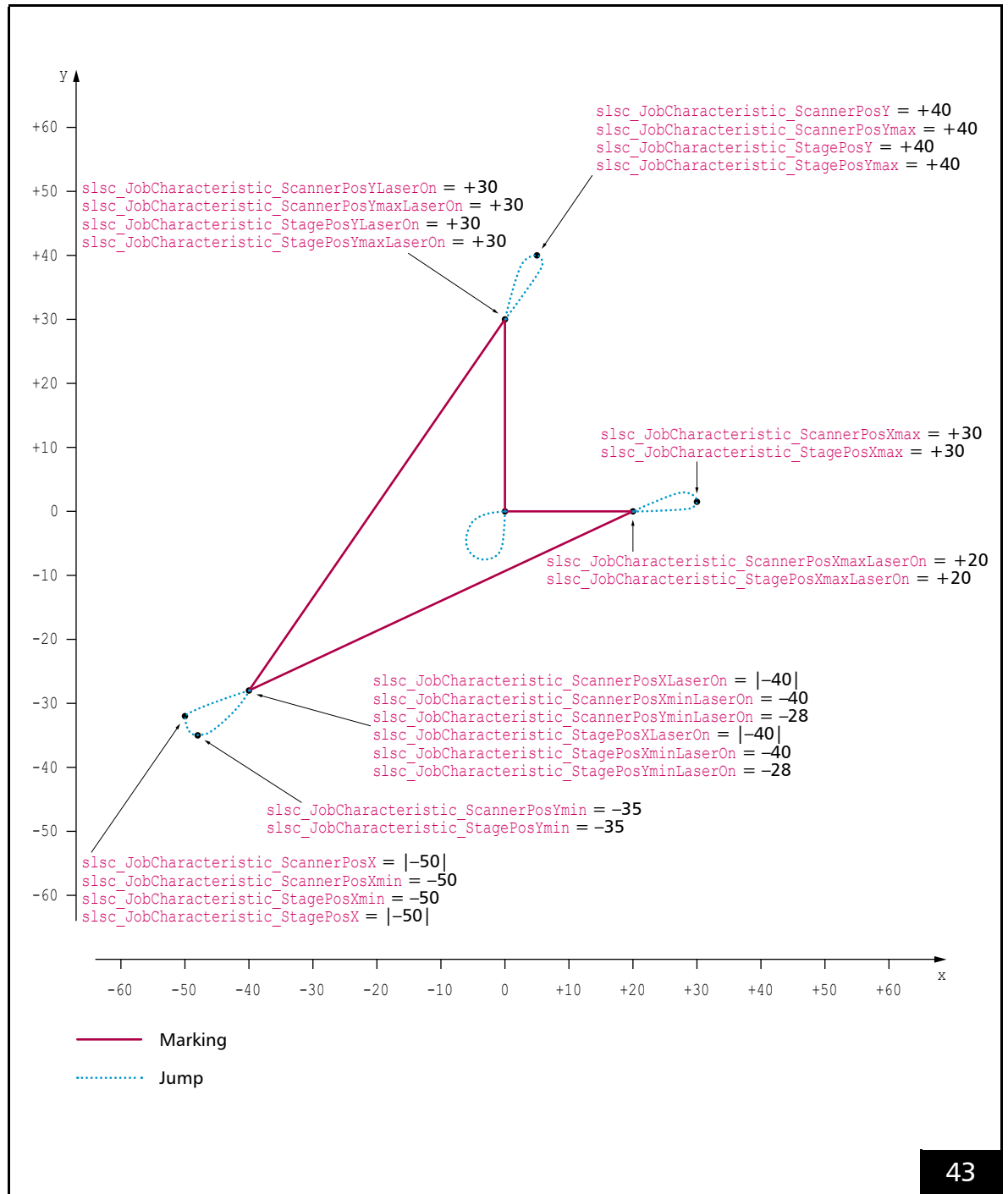
Name of the enum	slsc_JobCharacteristic
Description	<p>This enum defines the choices for:</p> <ul style="list-style-type: none"> The Job characteristic ("Key") <p>It refers to the Job-ID specified with slsc_ctrl_get_job_characteristic (that is, is "Job-related").</p> <p>Because the Job characteristic values are calculated by the Trajectory planning, the status of this Job-ID must have been reached "Calculation: Finished" (see Figure 12, page 43).</p> <p>See also Figure 43, page 311.</p>
Used by	<p>This enum is used with:</p> <ul style="list-style-type: none"> slsc_ctrl_get_job_characteristic
Syntax	<pre>enum slsc_JobCharacteristic { slsc_JobCharacteristic_ScannerPosX = 0, slsc_JobCharacteristic_ScannerPosY = 1, slsc_JobCharacteristic_StagePosX = 2, slsc_JobCharacteristic_StagePosY = 3, slsc_JobCharacteristic_ScannerVelX = 4, slsc_JobCharacteristic_ScannerVelY = 5, slsc_JobCharacteristic_StageVelX = 6, slsc_JobCharacteristic_StageVelY = 7, slsc_JobCharacteristic_ScannerAccX = 8, slsc_JobCharacteristic_ScannerAccY = 9, slsc_JobCharacteristic_StageAccX = 10, slsc_JobCharacteristic_StageAccY = 11, slsc_JobCharacteristic_StageJerkX = 12, slsc_JobCharacteristic_StageJerkY = 13, slsc_JobCharacteristic_MotionMicroSteps = 14, slsc_JobCharacteristic_ScannerPosXLaserOn = 30, slsc_JobCharacteristic_ScannerPosYLaserOn = 31, slsc_JobCharacteristic_StagePosXLaserOn = 32, slsc_JobCharacteristic_StagePosYLaserOn = 33, slsc_JobCharacteristic_MinimalMarkSpeed = 50, slsc_JobCharacteristic_MaximalMarkSpeed = 51, slsc_JobCharacteristic_InsertedSkywritings = 200, slsc_JobCharacteristic_ScannerPosXmax = 251, slsc_JobCharacteristic_ScannerPosXmin = 252, slsc_JobCharacteristic_ScannerPosYmax = 253, slsc_JobCharacteristic_ScannerPosYmin = 254, slsc_JobCharacteristic_ScannerPosXmaxLaserOn = 255, slsc_JobCharacteristic_ScannerPosXminLaserOn = 256, slsc_JobCharacteristic_ScannerPosYmaxLaserOn = 257, slsc_JobCharacteristic_ScannerPosYminLaserOn = 258, slsc_JobCharacteristic_StagePosXmax = 259, slsc_JobCharacteristic_StagePosXmin = 260, slsc_JobCharacteristic_StagePosYmax = 261, slsc_JobCharacteristic_StagePosYmin = 262, slsc_JobCharacteristic_StagePosXmaxLaserOn = 263, slsc_JobCharacteristic_StagePosXminLaserOn = 264, slsc_JobCharacteristic_StagePosYmaxLaserOn = 265, slsc_JobCharacteristic_StagePosYminLaserOn = 266, };</pre>

Name of the enum	slsc_JobCharacteristic	
Enumeration constant(s)	slsc_JobCharacteristic_ScannerPosX	<p>Max. distance of the scan head to zero position (without offset, see Chapter 8.3 "About Transformations in syncAXIS control V1.2.4 and Higher", page 332). In x direction. In mm.</p> <p>Notes</p> <ul style="list-style-type: none"> Defined offsets (see Chapter 8.3 "About Transformations in syncAXIS control V1.2.4 and Higher", page 332) are included in the values and are not compensated. Value calculated in the Trajectory planning. The actual executed value can deviate (for example, because of the RTC6 correction file).
	slsc_JobCharacteristic_ScannerPosY	<p>Max. distance of the scan head to zero position (without offset, see Chapter 8.3 "About Transformations in syncAXIS control V1.2.4 and Higher", page 332). In y direction. In mm.</p> <p>The Notes at slsc_JobCharacteristic_ScannerPosX apply.</p>
	slsc_JobCharacteristic_StagePosX	<p>Max. distance of the positioning stage to zero position (without offset, see Chapter 8.3 "About Transformations in syncAXIS control V1.2.4 and Higher", page 332). In x direction. In mm.</p> <p>Notes</p> <ul style="list-style-type: none"> Defined offsets (see Chapter 8.3 "About Transformations in syncAXIS control V1.2.4 and Higher", page 332) are included in the values and are not compensated. Value calculated in the Trajectory planning. The actual executed value can deviate (for example, because of downstream operations, for example in the ACS control system such as error corrections).
	slsc_JobCharacteristic_StagePosY	<p>Max. distance of the positioning stage to zero position (without offset, see Chapter 8.3 "About Transformations in syncAXIS control V1.2.4 and Higher", page 332). In y direction. In mm.</p> <p>The Notes at slsc_JobCharacteristic_StagePosX apply.</p>
	slsc_JobCharacteristic_ScannerVelX	<p>Absolute value of the max. scan head velocity. In x direction. In mm/s.</p> <p>The Notes at slsc_JobCharacteristic_ScannerPosX apply.</p>
	slsc_JobCharacteristic_ScannerVelY	<p>Absolute value of the max. scan head velocity. In y direction. In mm/s.</p> <p>The Notes at slsc_JobCharacteristic_ScannerPosX apply.</p>

Name of the enum	slsc_JobCharacteristic	
Enumeration constant(s) (cont'd)	slsc_JobCharacteristic_StageVelX	<i>Absolute value of the max. positioning stage velocity.</i> In x direction. In mm/s. The Notes at slsc_JobCharacteristic_StagePosX apply.
	slsc_JobCharacteristic_StageVelY	<i>Absolute value of the max. positioning stage velocity.</i> In y direction. In mm/s. The Notes at slsc_JobCharacteristic_StagePosX apply.
	slsc_JobCharacteristic_ScannerAccX	<i>Absolute value of the max. scan head acceleration.</i> In x direction. In mm/s ² . The Notes at slsc_JobCharacteristic_ScannerPosX apply.
	slsc_JobCharacteristic_ScannerAccY	<i>Absolute value of the max. scan head acceleration.</i> In y direction. In mm/s ² . The Notes at slsc_JobCharacteristic_ScannerPosX apply.
	slsc_JobCharacteristic_StageAccX	<i>Absolute value of the max. positioning stage acceleration.</i> In x direction. In mm/s ² . The Notes at slsc_JobCharacteristic_StagePosX apply.
	slsc_JobCharacteristic_StageAccY	<i>Absolute value of the max. positioning stage acceleration.</i> In y direction. In mm/s ² . The Notes at slsc_JobCharacteristic_StagePosX apply.
	slsc_JobCharacteristic_StageJerkX	<i>Absolute value of the max. positioning stage jerk.</i> In x direction. In mm/s ³ . The Notes at slsc_JobCharacteristic_StagePosX apply.
	slsc_JobCharacteristic_StageJerkY	<i>Absolute value of the max. positioning stage jerk.</i> In y direction. In mm/s ³ . The Notes at slsc_JobCharacteristic_StagePosX apply.
	slsc_JobCharacteristic_MotionMicroSteps	Number of micro vectors that make up the Job . Value calculated in the Trajectory planning . Hence, slsc_JobCharacteristic_MotionMicroSteps correspond to the minimum duration of the Job (1 micro vector = 10 μs). But the actual execution time may be longer (due to subsequent operations in the system, therefore, a quantitative statement cannot be made here). In particular, slsc_JobCharacteristic_MotionMicroSteps is useful to evaluate the effects of parameter permutations (in the context of optimizing steps).
	slsc_JobCharacteristic_ScannerPosXLaserOn	<i>Absolute value of the max. scan head position.</i> In x direction. With laser switched on. In mm. The Notes at slsc_JobCharacteristic_ScannerPosX apply.
	slsc_JobCharacteristic_ScannerPosYLaserOn	<i>Absolute value of the max. scan head position.</i> In y direction. With laser switched on. In mm. The Notes at slsc_JobCharacteristic_ScannerPosX apply.

Name of the enum	slsc_JobCharacteristic	
Enumeration constant(s) (cont'd)	slsc_JobCharacteristic_StagePosXLaserOn	<i>Absolute value</i> of the max. positioning stage position. In x direction. With laser switched on. In mm. The Notes at slsc_JobCharacteristic_StagePosX apply.
	slsc_JobCharacteristic_StagePosYLaserOn	<i>Absolute value</i> of the max. positioning stage position. In y direction. With laser switched on. In mm. The Notes at slsc_JobCharacteristic_StagePosX apply.
Enumeration constant(s) (cont'd)	slsc_JobCharacteristic_MinimalMarkSpeed	The minimum marking speed during the Job . Only those parts are taken into account in which the laser is switched on (laser spot speed).
	slsc_JobCharacteristic_MaximalMarkSpeed	The maximum marking speed during the Job . Only those parts are taken into account in which the laser is switched on (laser spot speed).
	slsc_JobCharacteristic_InsertedSkywritings	Applies to the following successions in the Job only: <ul style="list-style-type: none"> mark vector → mark vector mark vector → arc arc → mark vector arc → arc Number of successions in which a Sky Writing-like motion has been inserted (because a direct crossing would violate the dynamic limits and also a blending cannot be implemented).
	slsc_JobCharacteristic_ScannerPosXmax	Max. scan head position. In x direction. In mm. The Notes at slsc_JobCharacteristic_ScannerPosX apply.
	slsc_JobCharacteristic_ScannerPosXmin	Min. scan head position. In x direction. In mm. The Notes at slsc_JobCharacteristic_ScannerPosX apply.
	slsc_JobCharacteristic_ScannerPosYmax	Max. scan head position. In y direction. In mm. The Notes at slsc_JobCharacteristic_ScannerPosX apply.
	slsc_JobCharacteristic_ScannerPosYmin	Min. scan head position. In y direction. In mm. The Notes at slsc_JobCharacteristic_ScannerPosX apply.
	slsc_JobCharacteristic_ScannerPosXmaxLaserOn	Max. scan head position. In x direction. With laser switched on. In mm. The Notes at slsc_JobCharacteristic_ScannerPosX apply.
	slsc_JobCharacteristic_ScannerPosXminLaserOn	Min. scan head position. In x direction. With laser switched on. In mm. The Notes at slsc_JobCharacteristic_ScannerPosX apply.
	slsc_JobCharacteristic_ScannerPosYmaxLaserOn	Max. scan head position. In y direction. With laser switched on. In mm. The Notes at slsc_JobCharacteristic_ScannerPosX apply.

Name of the enum	slsc_JobCharacteristic	
Enumeration constant(s) (cont'd)	slsc_JobCharacteristic_ScannerPosYminLaserOn	Min. scan head position. In y direction. With laser switched on. In mm. The Notes at slsc_JobCharacteristic_ScannerPosX apply.
	slsc_JobCharacteristic_StagePosXmax	Max. positioning stage position. In x direction. In mm. The Notes at slsc_JobCharacteristic_StagePosX apply.
	slsc_JobCharacteristic_StagePosXmin	Min. positioning stage position. In x direction. In mm. The Notes at slsc_JobCharacteristic_StagePosX apply.
	slsc_JobCharacteristic_StagePosYmax	Max. positioning stage position. In y direction. In mm. The Notes at slsc_JobCharacteristic_StagePosX apply.
	slsc_JobCharacteristic_StagePosYmin	Min. positioning stage position. In y direction. In mm. The Notes at slsc_JobCharacteristic_StagePosX apply.
	slsc_JobCharacteristic_StagePosXmaxLaserOn	Max. positioning stage position. In x direction. With laser switched on. In mm. The Notes at slsc_JobCharacteristic_StagePosX apply.
	slsc_JobCharacteristic_StagePosXminLaserOn	Min. positioning stage position. In x direction. With laser switched on. In mm. The Notes at slsc_JobCharacteristic_StagePosX apply.
	slsc_JobCharacteristic_StagePosYmaxLaserOn	Max. positioning stage position. In y direction. With laser switched on. In mm. The Notes at slsc_JobCharacteristic_StagePosX apply.
	slsc_JobCharacteristic_StagePosYminLaserOn	Min. positioning stage position. In y direction. With laser switched on. In mm. The Notes at slsc_JobCharacteristic_StagePosX apply.
Version info	Available as of syncAX/IS-DLL V1.5.0.	



enum **slsc_JobCharacteristic**. Example value returns.

Name of the enum	slsc_ListHandlingMode	
Description	This enum defines the choices for: <ul style="list-style-type: none"> The return behavior of the Job functions (slsc_list_*) 	
Used by	This enum is used with: <ul style="list-style-type: none"> slsc_cfg_set_list_handling_mode slsc_cfg_set_list_handling_mode_with_context 	
Syntax	<pre>enum slsc_ListHandlingMode { slsc_ListHandlingMode_ReturnAtOnce =0, slsc_ListHandlingMode_RepeatWhileBufferFull =1, slsc_ListHandlingMode_RepeatWhilePredicate =2, };</pre>	
Enumeration constant(s)	slsc_ListHandlingMode_ReturnAtOnce	No list handling. See syncAXISConfig.xml tag InitialListHandlingMode . Mandatory notation, if used there: ReturnAtOnce
	slsc_ListHandlingMode_RepeatWhileBufferFull	It is tried to execute the Job function until the return value indicates that Bit #04 is not set any more (no longer BUFFER_FULL). Errors are returned nevertheless. See syncAXISConfig.xml tag InitialListHandlingMode . Mandatory notation, if used there: RepeatWhileBufferFull
	slsc_ListHandlingMode_RepeatWhilePredicate	It is tried to execute the Job function until the return value of the Predicate function is no longer true. See syncAXISConfig.xml tag InitialListHandlingMode . Mandatory notation, if used there: RepeatWhilePredicate
Version info	Available as of syncAXIS-DLL V0.11.0.	

Name of the enum	slsc_MeasurementSignal	
Description	This enum defines the choices for: <ul style="list-style-type: none"> The measurement signals 	
Used by	This enum is used with: <ul style="list-style-type: none"> slsc_ctrl_get_value 	
Syntax	<pre>enum slsc_MeasurementSignal { slsc_MeasurementSignal_Status =0, slsc_MeasurementSignal_Sample =1, slsc_MeasurementSignal_AnalogOutput_1 =2, slsc_MeasurementSignal_AnalogOutput_2 =3, slsc_MeasurementSignal_Errors =4, };</pre>	
Enumeration constant(s)	slsc_MeasurementSignal_Status	Deprecated. As of syncAXIS-DLL V1.2.0, enum slsc_PositionType is available. [Meaning in ≤ V1.1: Reserved.]
	slsc_MeasurementSignal_Sample	Deprecated. As of syncAXIS-DLL V1.2.0, enum slsc_PositionType is available. [Meaning in ≤ V1.1: Position value (commanded by the RTC6) of the specified axis. With scan head: deflection in the image field. In mm. With positioning stage: position value. In mm.]
	slsc_MeasurementSignal_AnalogOutput_1	Value at the analog output port 1.
	slsc_MeasurementSignal_AnalogOutput_2	Value at the analog output port 2.
	slsc_MeasurementSignal_Errors	Reserved.
Version info	Available as of syncAXIS-DLL V0.11.0.	

Name of the enum	slsc_OperationMode	
Description	<p>This enum defines the choices for:</p> <ul style="list-style-type: none"> The Operation mode of the syncAXIS control instance 	
Used by	<p>This enum is used with:</p> <ul style="list-style-type: none"> slsc_cfg_get_mode slsc_cfg_set_mode 	
Syntax	<pre>enum slsc_OperationMode { slsc_OperationMode_ScannerOnly =0, slsc_OperationMode_StageOnly =1, slsc_OperationMode_ScannerAndStage =2, };</pre>	
Enumeration constant(s)	slsc_OperationMode_ScannerOnly	<p>Only scan head, no positioning stage.</p> <p>The speed limits are given by JumpSpeed and MarkSpeed. In this Operation mode, however, the dynamic is restricted by the scanner dynamic limits.</p> <p>See syncAXISConfig.xml tag InitialOperationMode. Mandatory notation, if used there:</p> <p>ScannerOnly</p>
	slsc_OperationMode_StageOnly	<p>Only positioning stage, no scan head.</p> <p>The speed limits are given by JumpSpeed and MarkSpeed. In this Operation mode, however, the dynamic is restricted by the <i>reduced</i> positioning stage dynamic limits. This speed limit is limited in particular by the maximum positioning stage speed.</p> <p>See syncAXISConfig.xml tag InitialOperationMode. Mandatory notation, if used there:</p> <p>StageOnly</p>
	slsc_OperationMode_ScannerAndStage	<p>Both, scan head and positioning stage.</p> <p>The speed limits are given by JumpSpeed and MarkSpeed. In this Operation mode, however, the dynamic is restricted by the scanner dynamic limits and is influenced by further Heuristics.</p> <p>See syncAXISConfig.xml tag InitialOperationMode. Mandatory notation, if used there:</p> <p>ScannerAndStage</p>
Version info	Available as of syncAXIS-DLL V0.11.0.	

Name of the enum	slsc_OperationStatus	
Description	This enum defines the choices for: <ul style="list-style-type: none"> For the operation status of the syncAXIS control instance 	
Used by	This enum is used with: <ul style="list-style-type: none"> slsc_cfg_get_operation_status 	
Syntax	<pre>enum slsc_OperationStatus { slsc_OperationStatus_Green =0, slsc_OperationStatus_Yellow =1, slsc_OperationStatus_Red =2, };</pre>	
Enumeration constant(s)	slsc_OperationStatus_Green	Operation status "green": the syncAXIS control instance is running and no errors occurred.
	slsc_OperationStatus_Yellow	Operation status "yellow": the syncAXIS control instance is not yet running because the initialization is still in progress.
	slsc_OperationStatus_Red	Operation status "red": the syncAXIS control instance is not running and at least one error occurred.
Version info	Available as of syncAXIS-DLL V0.11.0.	

Name of the enum	slsc_PositionType	
Description	This enum defines the choices for: <ul style="list-style-type: none"> The position type 	
Used by	This enum is used with: <ul style="list-style-type: none"> slsc_ctrl_get_scan_device_position slsc_ctrl_get_stage_position 	
Syntax	<pre>enum slsc_PositionType { slsc_PositionType_Set = 0, slsc_PositionType_Actual = 1, };</pre>	
Enumeration constant(s)	slsc_PositionType_Set	Set position.
	slsc_PositionType_Actual	Actual position.
Version info	Available as of syncAXIS-DLL V1.2.4.	

Name of the enum	slsc_ScanDevice	
Description	<p>This enum defines the choices for:</p> <ul style="list-style-type: none"> The scan device to which a particular setting (for example, a transformation) is to be applied 	
Used by	<p>This enum is used with:</p> <ul style="list-style-type: none"> slsc_cfg_set_part_displacement 	
Syntax	<pre>enum slsc_ScanDevice { slsc_ScanDevice_None = 0, slsc_ScanDevice1 = 1, slsc_ScanDevice2 = 2, slsc_ScanDevice3 = 3, slsc_ScanDevice4 = 4, };</pre>	
Enumeration constant(s)	slsc_ScanDevice_None	<p>No scan device.</p> <p>See syncAXISConfig.xml tag HeadA, HeadB, ScanDevice. Mandatory notation, if used there:</p> <p>None</p>
	slsc_ScanDevice1	<p>Scan device 1.</p> <p>See syncAXISConfig.xml tag HeadA, HeadB, ScanDevice. Mandatory notation, if used there:</p> <p>ScanDevice1</p>
	slsc_ScanDevice2	<p>Scan device 2.</p> <p>See syncAXISConfig.xml tag HeadA, HeadB, ScanDevice. Mandatory notation, if used there:</p> <p>ScanDevice2</p>
	slsc_ScanDevice3	<p>Scan device 3.</p> <p>See syncAXISConfig.xml tag HeadA, HeadB, ScanDevice. Mandatory notation, if used there:</p> <p>ScanDevice3</p>
	slsc_ScanDevice4	<p>Scan device 4.</p> <p>See syncAXISConfig.xml tag HeadA, HeadB, ScanDevice. Mandatory notation, if used there:</p> <p>ScanDevice4</p>
Version info	Available as of syncAXIS-DLL V1.3.0.	

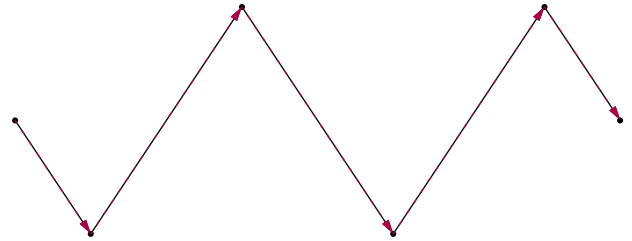
Name of the enum	slsc_SimulationSetting
Description	<p>This enum defines the choices for:</p> <ul style="list-style-type: none"> The Simulation Setting of the syncAXIS control instance
Used by	<p>This enum is used with:</p> <ul style="list-style-type: none"> slsc_cfg_get_simulation_setting slsc_cfg_set_simulation_setting
Syntax	<pre>enum slsc_SimulationSetting { slsc_SimulationSetting_SimulationMode =0, slsc_SimulationSetting_HardwareMode =1, };</pre>
Enumeration constant(s)	<div> slsc_SimulationSetting_SimulationMode <div> <p>The syncAXIS control instance does not attempt to connect to any hardware and merely simulates Jobs on the PC.</p> <p>See syncAXISConfig.xml tag SimulationMode true.</p> </div> </div>
	<div> slsc_SimulationSetting_HardwareMode <div> <p>The syncAXIS control instance is going to connect to all hardware and to physically execute Jobs.</p> <p>See syncAXISConfig.xml tag SimulationMode false.</p> </div> </div>
Version info	Available as of syncAXIS-DLL V1.5.0.

Name of the enum	slsc_SplineModes	
Description	This enum defines the choices for: <ul style="list-style-type: none"> The spline mode (\leq V1.4.0) 	
Used by	This enum is used with: <ul style="list-style-type: none"> Structure slsc_GeometryConfig 	
Syntax	<pre>enum slsc_SplineModes { slsc_SplineModes_Deactivated = 0, slsc_SplineModes_Interpolating = 1, slsc_SplineModes_Approximating = 2, };</pre>	
Enumeration constant(s)	slsc_SplineModes_Deactivated	Recommended setting. No splines, see Figure 44, page 319 . See syncAXISConfig.xml tag SplineMode . Mandatory notation, if used there: Deactivated
	slsc_SplineModes_Interpolating	Deprecated. Interpolating splines were used, see Figure 44, page 319 . See syncAXISConfig.xml tag SplineMode . Mandatory notation, if used there: Interpolating
	slsc_SplineModes_Approximating	Deprecated. Approximating splines were used, see Figure 44, page 319 . See syncAXISConfig.xml tag SplineMode . Mandatory notation, if used there: Approximating
Version info	Available as of syncAXIS-DLL V0.11.0.	

The 3 syncAXIS control spline modes. Only with sequences of > 2 mark vectors.
Mark vectors as programmed: black.
Marking result: red.

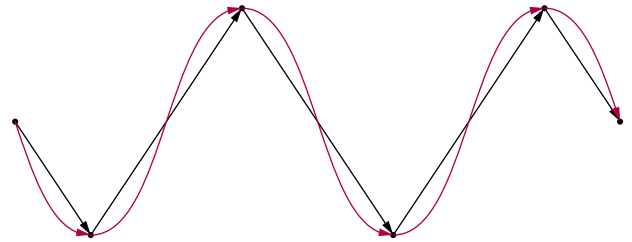
≥ V1.5.0: Recommended.

`slsc_SplineModes_Deactivated`
As trajectories, no splines are calculated.
The marking result exhibits no curves.



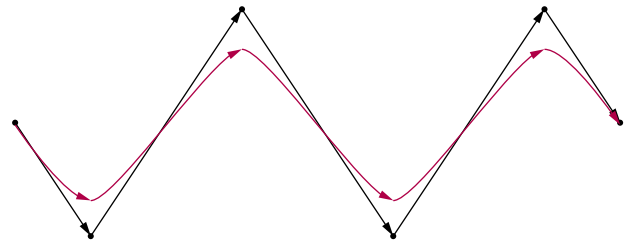
≥ V1.5.0: Deprecated.

`slsc_SplineModes_Interpolating`
As trajectories, splines are calculated.
The marking result exhibits curves.
The curve runs through its interpolation points.



≥ V1.5.0: Deprecated.

`slsc_SplineModes_Approximating`
As trajectories, splines are calculated.
The marking result exhibits curves.
The curve runs through its interpolation points
(control points are next to the curve).



enum **slsc_SplineModes**: On the mode of action of the arguments `slsc_SplineModes_Deactivated`, `slsc_SplineModes_Interpolating` and `slsc_SplineModes_Approximating`.

Name of the enum	slsc_Stage	
Description	This enum defines the choices for: <ul style="list-style-type: none"> The positioning stage which is to be used 	
Used by	This enum is used with: <ul style="list-style-type: none"> slsc_cfg_select_stage 	
Syntax	<pre>enum slsc_Stage { slsc_Stage_None = 0, slsc_Stage1 = 1, slsc_Stage2 = 2, slsc_Stage3 = 3, slsc_Stage4 = 4, };</pre>	
Enumeration constant(s)	slsc_Stage_None	No positioning stage. See syncAXISConfig.xml tag HeadA , HeadB , Stage . Mandatory notation, if used there: None
	slsc_Stage1	Positioning stage 1. See syncAXISConfig.xml tag HeadA , HeadB , Stage . Mandatory notation, if used there: Stage1
	slsc_Stage2	Positioning stage 2. See syncAXISConfig.xml tag HeadA , HeadB , Stage . Mandatory notation, if used there: Stage2
	slsc_Stage3	Positioning stage 3. See syncAXISConfig.xml tag HeadA , HeadB , Stage . Mandatory notation, if used there: Stage3
	slsc_Stage4	Positioning stage 4. See syncAXISConfig.xml tag HeadA , HeadB , Stage . Mandatory notation, if used there: Stage4
Version info	Available as of syncAXIS-DLL V1.2.4.	

8 Appendix A: Using syncAXIS control V1.2.4 and Higher with XL SCAN Multi-Head Systems

Notes

- The term **Multi-Head** refers to an XL SCAN system where
 - 1 **syncAXIS control instance** controls
 - more than one excelliSCAN scan head⁽¹⁾ (in this Appendix “multi” refers to n=4, see **Figure 45, page 322**) and
 - 1 positioning stage.
 This requires a **Dongle** for syncAXIS control which has been explicitly configured for the corresponding number of scan heads⁽²⁾.

8.1 About this Appendix

This Appendix is intended exclusively for system integrators who already know how to implement an XL SCAN single-head system and know how to use it together with syncAXIS control.

It applies along with:

- “Installation of SCANLAB XL SCAN Components and Initial Operation of the XL SCAN System” Manual
- Main part of this Manual syncAXIS-DLL – Application Programming Interface

Notice!

Carefully read the document “syncAXIS control Software License Agreement” before installing and using syncAXIS control. 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.



Caution!

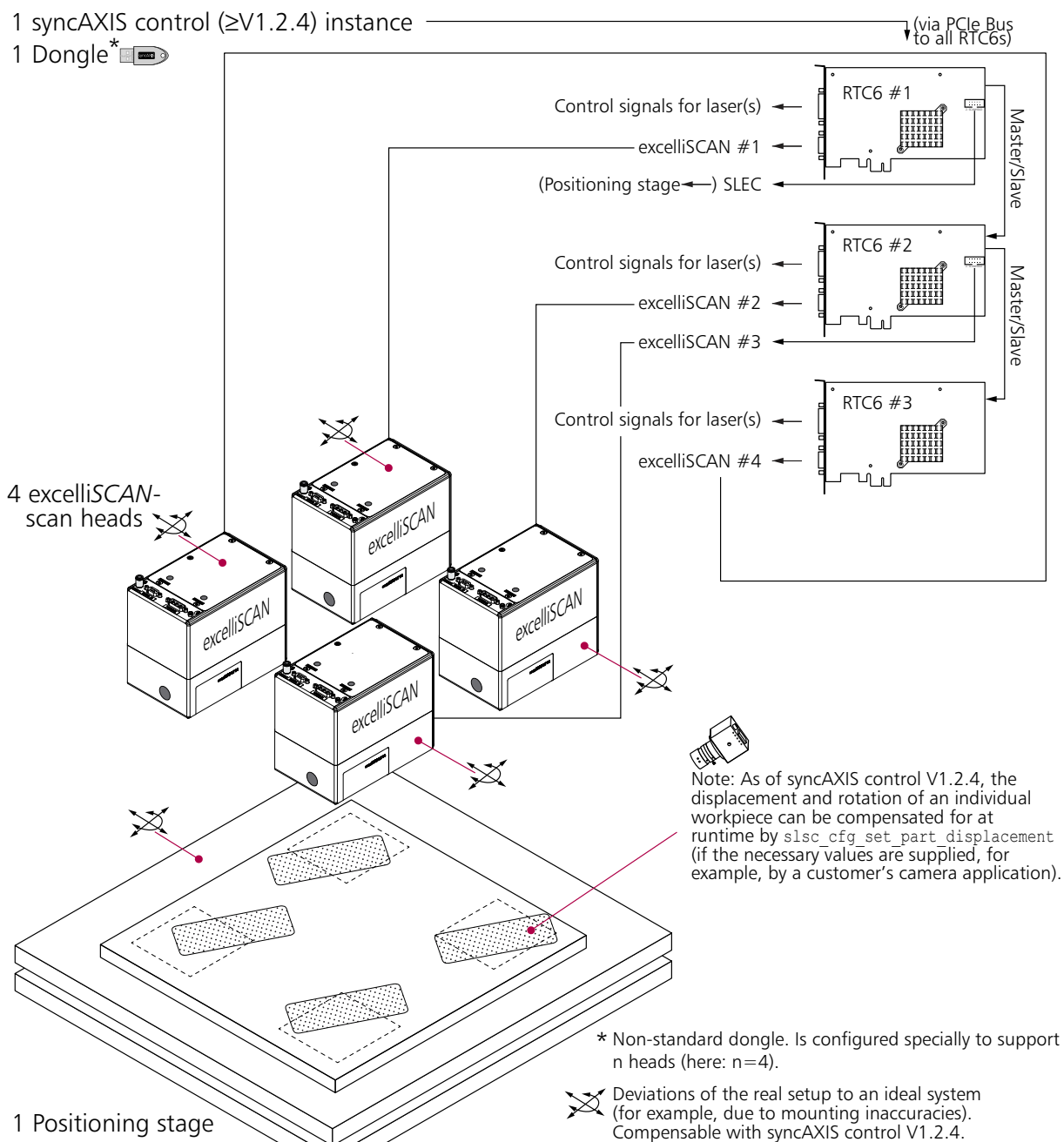
Read and observe all safety instructions in these manuals

- “Installation of SCANLAB XL SCAN Components and Initial Operation of the XL SCAN System” Manual
- “syncAXIS-DLL – Application Programming Interface” 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) All scan heads are supposed to mark the same pattern.

(2) That is, a standard **Dongle** is *not* sufficient.



Example of a 4-head setup (schematic illustration): 1 **syncAXIS control instance**, 4 excelliSCAN-heads, 1 positioning stage. In the `syncAXISConfig.xml`, the arrangement of the components must be configured accordingly, see also **Figure 47, page 326**.

The main part of “Installation of SCANLAB XL SCAN Components and Initial Operation of the XL SCAN System” Manual refers to an XL SCAN system with only 1 excelliSCAN scan head (and only 1 RTC6 board) and 1 positioning stage. It applies analogously to multi-head systems, but with the following differences:

- 3 additional excelliSCAN scan heads must be installed and made ready for operation.
- 2 additional RTC6 boards must be installed⁽¹⁾. Note, the connection for the 2nd scan head of one of the RTC6 boards is already occupied because it is used for the connection to the SLEC.
- The RTC6 boards must be connected as Master/Slave, see [RTC6 Manual](#). This requires 2 Master/Slave connection cables.
- 3 additional SL2-100 data cables must be provided and the cabling between the additional excelliSCANs and the additional RTC6 boards must be carried out with them.
- Note: the “Installation_Project” from previous software packages can be used (without changes) for both single head and multi-head systems. Only the [syncAXISConfig.xml](#), as it has been used until now (under syncAXIS control \leq V1.1), must be adapted for use with syncAXIS control \geq V1.2.4, see [Chapter 8.2 “Usage of syncAXIS control V1.2.4 and Higher”](#), page 324.
- For each additional excelliSCAN scan head, an optimized *.ct5 file must be created⁽²⁾. Note, in multi-head systems with syncAXIS control \geq V1.2.4, these *no* longer need to ensure that the (orthogonal) axes of the scan head and positioning stage coincide. This can be achieved via corresponding matrices in the [syncAXISConfig.xml](#), see [Figure 48](#), page 328 and [Figure 49](#), page 329.
- syncAXIS control \geq V1.2.4 must be used. On this topic, see [Chapter 8.2 “Usage of syncAXIS control V1.2.4 and Higher”](#), page 324.

Notice!

With syncAXIS control \geq V1.2.4 there are extended possibilities to comfortably configure systems in which the system components are arranged differently relative to each other than shown in [Figure 45](#), page 322. Further information can be found in [Chapter 8.3 “About Transformations in syncAXIS control V1.2.4 and Higher”](#), page 332.

(1) 1 additional SSHC slot cover required.

(2) The correction files supplied by the factory are not suitable to achieve precision results.

8.2 Usage of syncAXIS control V1.2.4 and Higher

8.2.1 Prerequisites for this Appendix

- The excelliSCAN scan heads are installed and ready for operation
- There is an optimized correction file (*.ct5) for each of the excelliSCAN scan heads.

8.2.2 Adapting syncAXISConfig.xml for syncAXIS control V1.2.4 and Higher

In particular observe the safety notices in “Installation of SCANLAB XL SCAN Components and Initial Operation of the XL SCAN System” Manual, Chapter 3 “Checking and Adapting the syncAXISConfig.xml”! That chapter applies analogously. However, to control the Multi-Head XL SCAN system (see page 14) it is mandatory to use syncAXIS control \geq V1.2.4. This results in several deviations and additional steps:

- syncAXIS control \geq V1.2.4 does not use syncAXISSysConfig.xml anymore.
- With syncAXIS control \geq V1.2.4, XML configuration files (syncAXISConfig.xml) are now validated against an XML scheme \geq syncAXIS_1_2.xsd⁽¹⁾. This means that every syncAXISConfig.xml for syncAXIS control \leq V1.1.n is no longer usable with syncAXIS control \geq V1.2.4. In order to be able to continue using them, the user must adapt them beforehand. Adapting is a 2 step process (observe the sequence of steps) which is described in the following:
 - (1) Technical adaptation
 - (2) Content adaptation

Step 1 of 2: Adapting syncAXISConfig.xml Technically

Change the structure of the syncAXISConfig.xml so that it is valid against the XML schema syncAXIS_1_2.xsd.

For this purpose, SCANLAB provides you with an executable file *.exe in the software package (this creates a syncAXISConfig.xml with the valid structure and takes over existing values).

If you need assistance with the adjustment, contact SCANLAB.

For further assistance you will also find a configuration example for a 2-head system in the software package (under configuration > syncAXISConfig_MultiHead.Template.xml).

Step 2 of 2: Adapting syncAXISConfig.xml in Regards to Content

More configuration settings are possible and necessary (because of the extensions of the XML schema syncAXIS_1_2.xsd to support multi-head systems). These are content changes and must therefore be entered manually by users, for example, the individual multi-head components and their properties (for example, at which RTC6 scan head connector is connected which scan device).

Notes

- Upon initializing the syncAXIS control instance
 - all RTC6 boards specified in syncAXISConfig.xml, section <cfg:RTCCfg>, see 2, must be available
 - all scan devices specified under <cfg:RTCCfg> must also be defined under <cfg:ScanDeviceConfig>
 - the positioning stage specified under <cfg:RTCCfg> must also be defined under <cfg:StageConfig>
 - all scan devices must be available
 - the positioning stage must be available
 Any additional (not listed under <cfg:RTCCfg>) entries for devices under <cfg:ScanDeviceConfig> and <cfg:StageConfig> have no effect.

(1) This also differs considerably from previous versions because there are many more configuration options.

(1) Section `<cfg:GeneralConfig>`, see [Figure 46](#),
[page 326](#):

This section has been restructured for
syncAXIS control \geq V1.2.4.

Deleted: `<cfg:SysConfigFilePath>`.

New: `<cfg:SimulationConfig>`.

Changed: `<cfg:LogConfig>` renamed (previously
`<cfg:LogConfiguration>`).

(2) Section `<cfg:RTCConfig>`, see [Figure 47](#), [page 326](#):

This section has been restructured for
syncAXIS control \geq V1.2.4.

New: More than 1 RTC6 board can be entered.

For each RTC6 board the name of the scan device
must be specified at scan head output A and B.

Moved: `<cfg:CorrectionFilePath ...>` (now in section
`<cfg:ScanDeviceConfig>`).

Note: Which of the RTC6 boards is master or slave
does not have to be entered (that is, the 'first'
RTC6 board in `syncAXISConfig.xml` does not neces-
sarily have to be the master, but can also be one
of the slave boards). The master board is determi-
nated by the physical cabling.

```
<cfg:GeneralConfig>
  <cfg:ACSController>127.0.0.1</cfg:ACSController>
  <cfg:InitialOperationMode>ScannerAndStage</cfg:InitialOperationMode>
  <cfg:InitialListHandlingMode>RepeatWhileBufferFull</cfg:InitialListHandlingMode>
  <cfg:BaseDirectoryPath>${BASE_PATH}</cfg:BaseDirectoryPath>
  <cfg:SimulationConfig>
    <cfg:SimulationMode>true</cfg:SimulationMode>
    <cfg:SimOutputFileDirectory>[BaseDirectoryPath]/Simulate/</cfg:SimOutputFileDirectory>
    <cfg:BinaryOutput>false</cfg:BinaryOutput>
    <cfg:DisableFileOutput>false</cfg:DisableFileOutput>
  </cfg:SimulationConfig>
  <cfg:LogConfig>
    <cfg:LogfilePath>[BaseDirectoryPath]/Log</cfg:LogfilePath>
    <cfg:LogLevel>Warn</cfg:LogLevel>
    <cfg:EnableConsoleLogging>true</cfg:EnableConsoleLogging>
    <cfg:EnableFileLogging>false</cfg:EnableFileLogging>
    <cfg:MaxLogfileSize>26214400</cfg:MaxLogfileSize>
    <cfg:MaxBackupFileCount>0</cfg:MaxBackupFileCount>
  </cfg:LogConfig>
</cfg:GeneralConfig>
```

Optional.

46

syncAXIS control V1.2.4: [syncAXISConfig.xml](#), example section for general settings <cfg:GeneralConfig>. See [page 325](#).

```
<cfg:RTCCConfig>
  <cfg:BoardIdentificationMethod>BySerialNumber</cfg:BoardIdentificationMethod>
  <cfg:ProgramFileDirectory>[BaseDirectoryPath]/RTC6</cfg:ProgramFileDirectory>
  <cfg:Boards>
    <cfg:RTC6>
      <cfg:SerialNumber>123457</cfg:SerialNumber>
      <cfg:HeadA>ScanDevice1</cfg:HeadA>
      <cfg:HeadB>Stage1</cfg:HeadB>
    </cfg:RTC6>
    <cfg:RTC6>
      <cfg:SerialNumber>123456</cfg:SerialNumber>
      <cfg:HeadA>ScanDevice2</cfg:HeadA>
      <cfg:HeadB>ScanDevice3</cfg:HeadB>
    </cfg:RTC6>
    <cfg:RTC6>
      <cfg:SerialNumber>123455</cfg:SerialNumber>
      <cfg:HeadA>ScanDevice4</cfg:HeadA>
      <cfg:HeadB>None</cfg:HeadB>
    </cfg:RTC6>
  </cfg:Boards>
</cfg:RTCCConfig>
```

47

syncAXIS control V1.2.4: [syncAXISConfig.xml](#), example section for the RTC6 boards <cfg:RTCCConfig>. See [page 325](#).

(3) Section `<cfg:ScanDeviceConfig>`, see [Figure 48](#), [page 328](#):

This is a new section in syncAXIS control \geq V1.2.4 (scan heads are generically referred to here as “scan devices”).

- Several scan devices can be specified now
- Name, matrix, offset and correction files must be specified per scan device.
- Furthermore, for workpieces below each of these scan devices, a matrix and offset can be specified.

Make sure that the matrices you specify can be inverted!

(4) Section `<cfg:StageConfig>`, see [Figure 49](#), [page 329](#):

This section has been renamed for syncAXIS control \geq V1.2.4 (previously: `<cfg:StageDynamics>`).

Make sure that the matrices you specify can be inverted!

(5) Section `<cfg:LaserConfig>`

Users do not need to make any changes here for syncAXIS control \geq V1.2.4.

Note: the laser signals are outputted identically on both RTC6 boards.

(6) Section `<cfg:TrajectoryConfig>`

The sub-section `<cfg:HeuristicConfig>` has to be deleted from this section `<cfg:TrajectoryConfig>` and to be added to section `<cfg:MotionDecompositionConfig>`, see [7](#).

(7) Section `<cfg:MotionDecompositionConfig>`, see [Figure 50](#), [page 330](#).

This section is new for syncAXIS control \geq V1.2.4 and must be correspondingly configured by users.

Notes: the sub-section `<cfg:HeuristicConfig>` derives from the section `<cfg:TrajectoryConfig>`, see [6](#). The tag `<cfg:FilterBandwidth>` corresponds to the tag `<cfg:Bandwidth>` in `syncAXIS SysConfig.xml` for syncAXIS control \leq V1.1.

(8) Section `<cfg:IOConfig>`

Users do not need to make any changes here for syncAXIS control \geq V1.2.4.

```

<cfg:ScanDeviceConfig>
  <cfg:FieldLimits>
    <cfg:XAxis Unit="mm" Max="150" Min="-150" />
    <cfg:YAxis Unit="mm" Max="150" Min="-150" />
  </cfg:FieldLimits>
  <cfg:DynamicLimits>
    <cfg:Velocity Unit="rad/s">90</cfg:Velocity>
    <cfg:Acceleration Unit="rad/s^2">1.1314e5</cfg:Acceleration>
    <cfg:Jerk Unit="rad/s^3">4e9</cfg:Jerk>
  </cfg:DynamicLimits>
  <cfg:DefaultCorrectionFile>0</cfg:DefaultCorrectionFile>
  <cfg:MaxGalvoAngle Unit="rad">10.3</cfg:MaxGalvoAngle>
  <cfg:FocalLength Unit="mm">160</cfg:FocalLength>
  <cfg:Delay Unit="s">0.00125</cfg:Delay>
  <cfg:ScanDeviceList>
    <cfg:ScanDevice Name="ScanDevice1">
      <cfg:Alignment>
        <cfg:Matrix>
          <cfg:T11>1</cfg:T11>
          <cfg:T12>0</cfg:T12>
          <cfg:T21>0</cfg:T21>
          <cfg:T22>1</cfg:T22>
        </cfg:Matrix>
        <cfg:Offset X="0" Y="0" />
      </cfg:Alignment>
      <cfg:BasePartDisplacement>
        <cfg:Matrix>
          <cfg:T11>1</cfg:T11>
          <cfg:T12>0</cfg:T12>
          <cfg:T21>0</cfg:T21>
          <cfg:T22>1</cfg:T22>
        </cfg:Matrix>
        <cfg:Offset X="0" Y="0" />
      </cfg:BasePartDisplacement>
      <cfg:CorrectionFileList>
        <cfg:CorrectionFilePath CalibrationFactor="-1">D2_584_imprvd_4_SD1_1.ct5</cfg:CorrectionFilePath>
        <cfg:CorrectionFilePath CalibrationFactor="-1">D2_584_factory.ct5</cfg:CorrectionFilePath>
      </cfg:CorrectionFileList>
    </cfg:ScanDevice>
    <cfg:ScanDevice Name="ScanDevice2">
      ...etc. ...
    </cfg:ScanDevice>
  </cfg:ScanDeviceList>
</cfg:ScanDeviceConfig>

```

Optional. Usually does not need to be changed.

Optional. Usually does not need to be changed.

Optional. Usually does not need to be changed.

Name of the 1st scan head

Coefficients m11...m22 of a (2 × 2) transformation matrix for the 1st scan head

Offset for the 1st scan head

Coefficients m11...m22 of a (2 × 2) transformation matrix for workpieces supposed to be processed by 1st scan head

Offset for these workpieces

Correction files (up to 4 tags)

Note: on runtime an additional transformation is possible by **slsc_cfg_set_part_displacement**.


```

<cfg:StageConfig>
  <cfg:Delay Unit="s">0.0014951</cfg:Delay>
  <cfg:FieldLimits>
    <cfg:XDirection Unit="mm" Max="150" Min="-150" />
    <cfg:YDirection Unit="mm" Max="150" Min="-150" />
  </cfg:FieldLimits>
  <cfg:DynamicLimits>
    <cfg:Velocity Unit="mm/s">1000</cfg:Velocity>
    <cfg:Acceleration Unit="mm/s^2">10000</cfg:Acceleration>
    <cfg:Jerk Unit="mm/s^3">100000</cfg:Jerk>
  </cfg:DynamicLimits>
  <cfg:CalculationDynamics>
    <cfg:Velocity Unit="mm/s">500</cfg:Velocity>
    <cfg:Acceleration Unit="mm/s^2">5000</cfg:Acceleration>
    <cfg:Jerk Unit="mm/s^3">50000</cfg:Jerk>
  </cfg:CalculationDynamics>
  <cfg:StageList>
    <cfg:Stage Name="Stage1">
      <cfg:FieldLimits>
        <cfg:XDirection Unit="mm" Max="150" Min="-150" />
        <cfg:YDirection Unit="mm" Max="150" Min="-150" />
      </cfg:FieldLimits>
      <cfg:DynamicLimits>
        <cfg:Velocity Unit="mm/s">1000</cfg:Velocity>
        <cfg:Acceleration Unit="mm/s^2">10000</cfg:Acceleration>
        <cfg:Jerk Unit="mm/s^3">100000</cfg:Jerk>
      </cfg:DynamicLimits>
      <cfg:CalculationDynamics>
        <cfg:Velocity Unit="mm/s">500</cfg:Velocity>
        <cfg:Acceleration Unit="mm/s^2">5000</cfg:Acceleration>
        <cfg:Jerk Unit="mm/s^3">50000</cfg:Jerk>
      </cfg:CalculationDynamics>
      <cfg:StageAxisX>0</cfg:StageAxisX>
      <cfg:StageAxisY>1</cfg:StageAxisY>
      <cfg:SlecEtherCATNodeID>0</cfg:SlecEtherCATNodeID>
      <cfg:Alignment>
        <cfg:Matrix>
          <cfg:T11>1</cfg:T11>
          <cfg:T12>0</cfg:T12>
          <cfg:T21>0</cfg:T21>
          <cfg:T22>1</cfg:T22>
        </cfg:Matrix>
        <cfg:Offset X="0" Y="0" />
      </cfg:Alignment>
    </cfg:Stage>
    ...etc. ...
  </cfg:StageList>
</cfg:StageConfig>

```

Mandatory. Values are taken over from the original xml.

Mandatory. Values are taken over from the original xml.

Mandatory. These values must be calculated and entered manually: CalculationDynamics (new) = ReducedStageDynamicFactor (syncAXISysConfig.xml) × DynamicLimits (syncAXISConfig.xml).

Name of the positioning stage

Optional. Overwrites the global stage configuration (that is, values from above).

Optional. Overwrites the global stage configuration (that is, values from above).

Optional. Overwrites the global stage configuration (that is, values from above).

Optional. For stage setups where ACS x axis index ≠ 0.

Optional. For stage setups where ACS y axis index ≠ 1.

Optional. For stage setups where SLEC-ID ≠ 0.

Coefficients m11...m22 of a (2 × 2) transformation matrix for the positioning stage

Offset for the positioning stage

```
<cfg:MotionDecompositionConfig>
  <cfg:FilterBandwidth>2</cfg:FilterBandwidth>
  <cfg:HeuristicConfig>
    <cfg:DynamicReductionFunction Units="mm and mm/s">
      <cfg:DataPoint Length="0.0" Velocity="2000" />
      <cfg:DataPoint Length="27.0" Velocity="2000" />
      <cfg:DataPoint Length="27.01" Velocity="700" />
      <cfg:DataPoint Length="54.0" Velocity="700" />
    </cfg:DynamicReductionFunction>
  </cfg:HeuristicConfig>
</cfg:MotionDecompositionConfig>
```

☐ Mandatory. Value is taken over from [syncAXISysConfig.xml](#).

50

syncAXIS control ≥ V1.2.4: [syncAXISConfig.xml](#), example section for filter bandwidth and **Heuristic** `<cfg:MotionDecompositionConfig>`. See [page 327](#).

8.2.3 Further Notes on the Use of syncAXIS control V1.2.4 and Higher

- “Jobs” that have been designed for syncAXIS control \leq V1.2.4 are compatible to syncAXIS control \geq V1.2.4 and do not need to be changed.
- syncAXIS control V1.2.4 allows developing software for Multi-Head systems. The syncAXIS-DLL provides the assignment of motions to scan heads and positioning stage. Code for multi-head systems is to be written in the same way as code for single-head systems (that is, software developers do not have to consider fundamental differences).
- syncAXIS control V1.2.4 offers extended possibilities to align scan head and positioning stage each other (even after the optimized correction file *.ct5 has been created). Among other things, there is the possibility (for example, to be able to compensate hardware setup errors) to compensate relative rotations to each other. Even axis inversions can be carried out. For this, in the `syncAXISConfig.xml`, there are in
 - Section `<cfg:ScanDeviceConfig>`, see [Figure 48, page 328](#), the Alignment tags Matrix and Offset (for the individual scan devices)⁽¹⁾. Make sure this matrix is invertible!
 - Section `<cfg:StageConfig>`, see [Figure 49, page 329](#), the Alignment tags Matrix and Offset (for the individual positioning stages)⁽²⁾. Make sure this matrix is invertible!
- With syncAXIS control V1.2.4, it is also possible to adjust – independently for each scan device – the set trajectory of each individual workpiece. (in `syncAXISConfig.xml` under `<cfg:Configuration>` → `<cfg:ScanDeviceConfig>` → `<cfg:ScanDeviceList>` → `<cfg:ScanDevice ...>` → `<cfg:BasePartDisplacement>`, see [Figure 48, page 328](#)). This results in a basic coordinate system for the individual scan devices (for example, to compensate for the different positions of different clamping systems). See also [Chapter 8.3 “About Transformations in syncAXIS control V1.2.4 and Higher”, page 332](#)).
- By `slsc_cfg_set_part_displacement`, it is possible to change the set trajectory individually for each individual workpiece (individually for each scan device) via API during the runtime of the user program. (must occur before the start of the `Job` calculation, that is, before `slsc_list_begin*` is called. This transformation is in addition to the coordinate transformation entered in `<cfg:BasePartDisplacement>`. For this purpose, only the desired matrix and offset values (for example, captured by image recognition) need to be passed to this function. See also [Chapter 8.3 “About Transformations in syncAXIS control V1.2.4 and Higher”, page 332](#).

(1) Exported as RTC6 commands `set_matrix`.

(2) The control values for the positioning stage are calculated within syncAXIS-DLL.

8.3 About Transformations in syncAXIS control V1.2.4 and Higher

In syncAXIS control \geq V1.2.4, there are several ways to change the marking pattern and to influence the control.

Depending on the application (example: a camera image evaluation determines the actual workpiece position) and the system setup present, certain transformations may be necessary.

In syncAXIS control, there are the following transformation types, see also [Figure 51, page 333](#) and [Figure 52, page 334](#):

- Transformation to change target point coordinates of the incoming pattern (even as of V1.0):
 - See [Figure 52, page 334](#), (1)
 - Via API by [slsc_cfg_set_matrix_and_offset](#)
 - Via API by [slsc_list_set_matrix_and_offset](#)
 - Via API by [slsc_cfg_set_rot_and_offset_2d](#)
 - Via API by [slsc_list_set_rot_and_offset_2d](#)

The effects are visible in the simulation result.
- Transformation to change the set trajectory of a marking pattern for individual scan devices (as of V1.2.4)
 - See [Figure 51, page 333](#), (2).
 - See [Figure 52, page 334](#), (2).
 - In [syncAXISConfig.xml](#), tag


```
<cfg:BasePartDisplacement>
```

, see also [page 331](#)
 - Via API by [slsc_cfg_set_part_displacement](#)

The effects are visible in the simulation result.

- Transformation to change the control values for the positioning stage (as of V1.2.4)
 - See [Figure 51, page 333](#), (3).
 - See [Figure 52, page 334](#), (3).
 - In [syncAXISConfig.xml](#), tag for


```
Stage1 -> <Alignment>
```

With this, it is possible:

 - to invert the axes
 - to align the positioning stages with each other (in multi-positioning stage-systems)

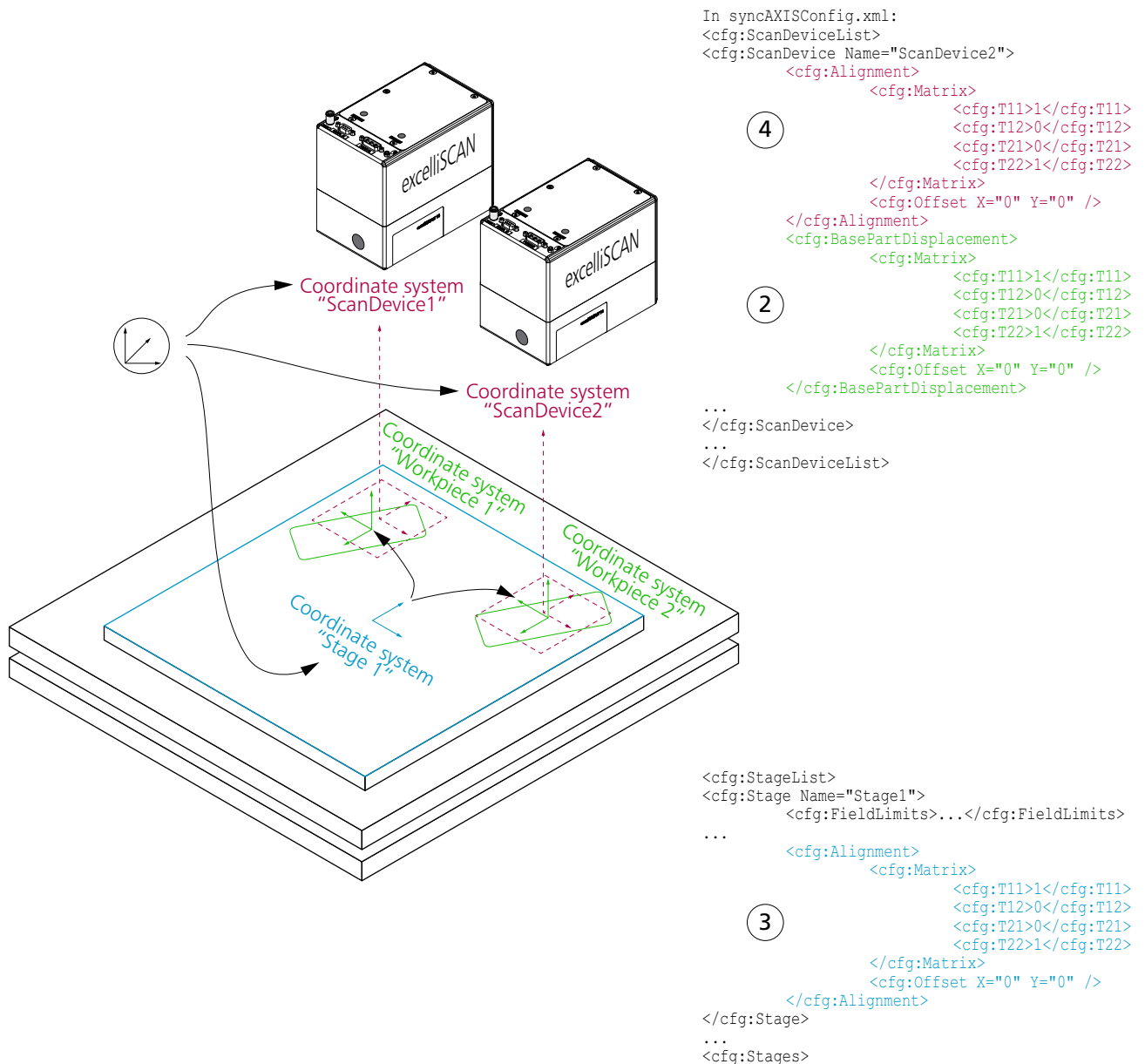
The effects are *not* visible in the simulation result.
- Transformation to change the control values for the scan device (as of V1.2.4)
 - See [Figure 51, page 333](#), (4).
 - See [Figure 52, page 334](#), (4).
 - In [syncAXISConfig.xml](#), Tag for

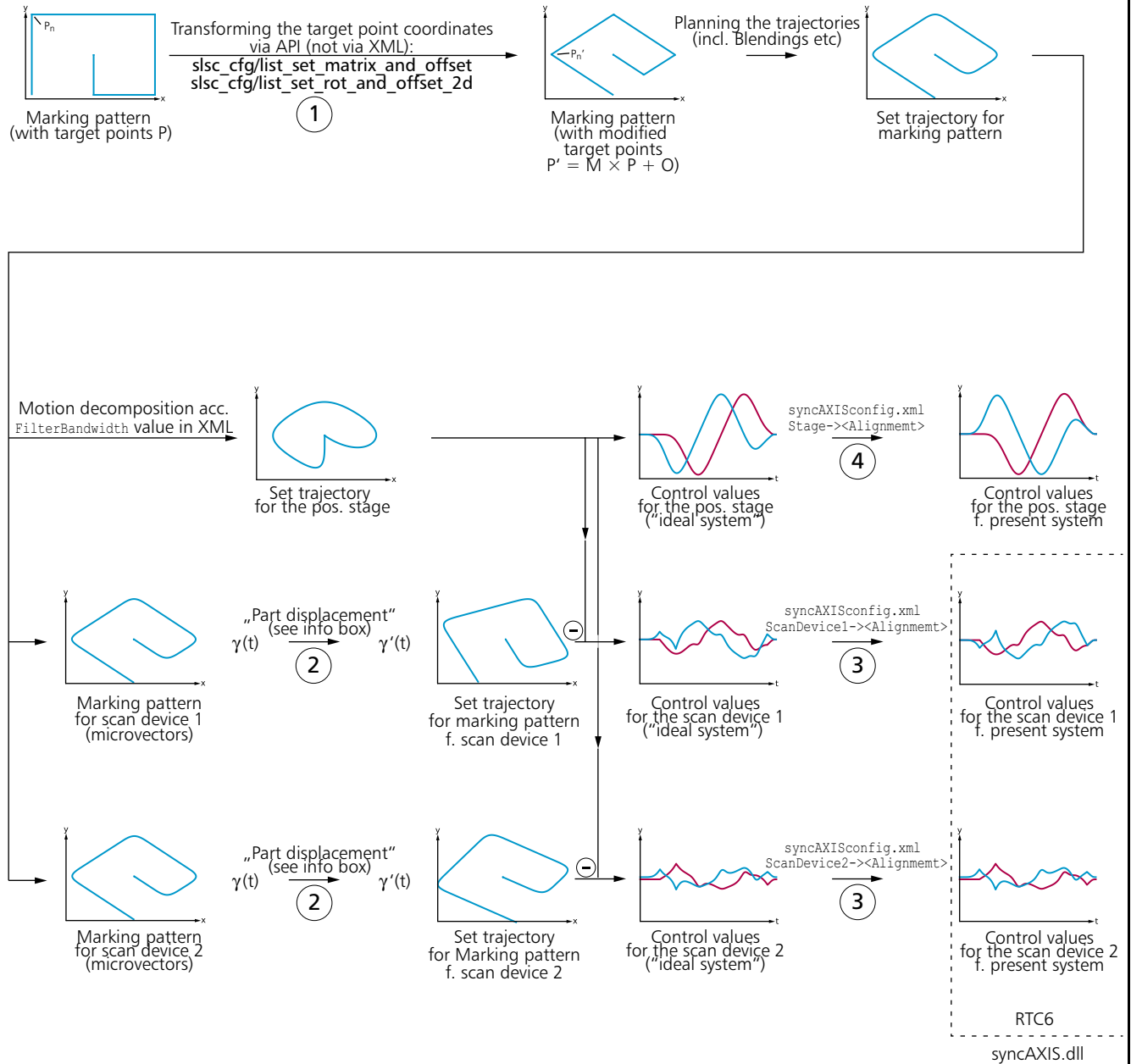

```
ScanDevice1 -> <Alignment>
```

With this, it is possible:

 - to align the scan device coordinate system with that of the positioning stage (to compensate for mounting errors)
 - to invert the axes

The effects are *not* visible in the simulation result.





Info box „Part displacement“

$$\gamma'(t) = \begin{matrix} \text{Matrix} \\ \text{Matrix}_{\text{XML}} \times \text{Matrix}_{\text{API}} \end{matrix} \times \gamma + \begin{matrix} \text{Offset} \\ \text{Offset}_{\text{XML}} + \text{Offset}_{\text{API}} \end{matrix}$$

where $\text{Matrix}_{\text{XML}}$ and $\text{Offset}_{\text{XML}}$ originates from the tag `<BasePartDisplacement>` in `syncAXISConfig.xml` and $\text{Matrix}_{\text{API}}$ and $\text{Offset}_{\text{API}}$ from `slsc_cfg_set_part_displacement`.
Example: if there is no tag `<BasePartDisplacement>`, then $\gamma'(t) = \text{Matrix}_{\text{API}} \times \gamma(t) + \text{Offset}_{\text{API}}$.

9 Appendix B: Application Note – Handling Lists with syncAXIS control

Notice!

The code sections in this Appendix only show the minimum set of functions necessary for an syncAXIS control-based user program that would be actually executable. These code sections only serve to illustrate certain concepts for the implementation of syncAXIS control-based user programs. In each example, the **Job** execution starts either once the buffer is full or once the execution state is `"slsc_ExecState_ReadyForExecution"`. In contrast, with actual syncAXIS control-based user programs, the start of a **Job** execution must depend on multiple conditions and security regulations. Make sure to comply to all relevant safety regulations and program your code accordingly. The code sections in this document do not take error handling into account. Make sure to always monitor the return value of each syncAXIS control function provides and react accordingly. Also note that you would have to change certain parameters in the code sections in order to fit the requirements of the actual XL SCAN system and **Job**.

In XL SCAN setups, RTC6 boards are used to control scan devices and laser. RTC6 boards buffer information (for example, positions) in their list memories⁽¹⁾. Later they are processed in real time (= every 10 μ s). The RTC6 list memory can hold up to 8 Million RTC6 list commands.

When programming with the `RTC6DLL.dll` only (that is, without syncAXIS control software), users themselves need to take care of the list handling. For this two approaches are common:

- to monitor input pointer and execution pointer
- to make use of the "2-list concept" (which is to fill one **List** while the other one is executing; in constant alternation).

With many RTC6 user programs, it is even possible to completely skip list handling: with the much longer RTC6 vector commands, RTC6 list memory is sufficient for far more than 40 s of execution time.

syncAXIS-DLL utilizes an own set of functions for the synchronous control of scan devices, positioning stage and laser. These syncAXIS control functions are similar to the RTC6 micro vector commands (see [RTC6 Manual](#)), but blocks 2 RTC6 list memory positions for 10 μ s of execution.

If the RTC6 list memory (with its capacity for 8 million RTC6 list commands) is used once and completely without reloading, a maximum marking execution time of 40 s can be achieved, see also [Figure 11, page 41](#).

(1) "List buffer".

Most users also want to mark **Jobs** that last longer than 40 s. Therefore, syncAXIS control is able to continuously loading lists (“automatic list reloading”). Windows PCs are not able to work in real time. Nevertheless, to avoid buffer underruns, **Job** calculation and **Job** transfer of the syncAXIS control instance is performed in a way that functional blocks are loaded and planning is faster than actual **Job** execution.

Therefore, it is only possible to start the **Job** execution after the processing of the first functional block has been completed. As syncAXIS control can calculate several **Jobs** subsequently while simultaneously transfer **Jobs** to the RTC6 board, the “automatic list reloading” addresses the use of several small **Jobs** as well as single larger **Jobs**.

This appendix shows examples of implementations in C++ for:

- small **Jobs**, without using a list handling mode, see [Figure 53, page 336](#)
- bigger **Jobs**, using the 3 different list handling modes
 - “List Handling Mode “ReturnAtOnce””, [page 337](#)
 - “List Handling Mode “RepeatWhileBufferFull””, [page 339](#)
 - “List Handling Mode “RepeatWhilePredicate””, [page 343](#)

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

size_t SLHandle;
slsc_cfg_initialize_from_file(&SLHandle, "syncAXISConfig.xml");

size_t JobID;
slsc_list_begin(SLHandle, &JobID);
// Insert slsc_list_-functions here.
slsc_list_end(SLHandle);

if (!startSingleJob(SLHandle))
{
    // !Insert proper error handling here!
}

slsc_cfg_delete(SLHandle);
```

Simplified code structure to execute small **Jobs** (that do not exceed the RTC6 list memory).

9.1 List Handling Mode “ReturnAtOnce”

“ReturnAtOnce” is the list handling mode most similar to RTC6 programming with the `RTC6DLL.dll` (that is, without `syncAXIS-DLL`). The `syncAXIS-DLL`-internal buffer is limited to the size of the RTC6 list memory and cannot be exceeded.

As soon as the 4.000.001st RTC6 micro vector command is to be written to the RTC6 list memory, the respective `syncAXIS` control function returns a return value $\neq 0$. This indicates that the RTC6 list memory is full. Then it is up to the user to first free up space in the RTC6 list memory before he/she can load more **Job** functions. This can be achieved by starting a **Job** execution (or deleting the **syncAXIS control instance**).

With list handling mode “ReturnAtOnce” it is not possible to overwrite not yet executed RTC6 list commands in the RTC6 list memory.

This is the major difference to the RTC6 programming with the `RTC6DLL.dll` where the writing just continues at the list memory begin, once the input pointer exceeds the list memory (which would overwrite not yet executed RTC6 list commands).

The main advantage compared to list handling mode “RepeatWhileBufferFull” and “RepeatWhilePredicate” is that the program flow gets does not stuck at the `syncAXIS` control function that is overloading the RTC6 list memory. This allows users to handle situations with full RTC6 list memory at will.

A possible approach of using the list handling mode “ReturnAtOnce”, see [Figure 54, page 338](#), is to load the **Job** until a `syncAXIS` control function returns the respective error bit. Then, the **Job** execution is started to free some RTC6 list memory. Then, the remaining **Job** (including the `syncAXIS` control Job function that returned the return value $\neq 0$) is loaded piecewise as long as no buffer overruns are indicated. If so, the user can just wait for some list memory to be freed and continue loading.

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

size_t SLHandle;
slsc_cfg_initialize_from_file(&SLHandle, "syncAXISConfig.xml");
slsc_cfg_set_list_handling_mode(SLHandle, slsc_ListHandlingMode::slsc_ListHandlingMode_ReturnAtOnce, nullptr);
size_t JobID;

slsc_list_begin(SLHandle, &JobID);

// Customize timeout. If the Buffer is not freed after a certain timeout,
// probably no Job is running and you should start execution.
size_t Retry_TimeOut = 5;
size_t BufferDelay = 10;

// For each slsc_list_* function call (for example, by creating wrapper functions).
{
    std::array<double, 2> Target{X, Y};
    size_t Retry_Index = 0;
    while ((slsc_list_jump_abs(SLHandle, Target.data()) == 0x0010) && (Retry_TimeOut > Retry_Index))
    {
        std::this_thread::sleep_for(std::chrono::milliseconds(BufferDelay));
        Retry_Index++;
    }
    if (Retry_TimeOut > Retry_Index)
    {
        // The final buffer is full now. Execution should be started.
        slsc_ctrl_start_execution(SLHandle);
    }
}

slsc_list_end(SLHandle);

if (!startSingleJob(SLHandle))
{
    // !Insert proper error handling here!
}

slsc_cfg_delete(SLHandle);
```

Simplified code structure to execute bigger **Jobs** (that do exceed the RTC6 list memory) and list handling mode `slsc_ListHandlingMode_ReturnAtOnce`.

9.2 List Handling Mode “RepeatWhileBufferFull”

SCANLAB recommends users new to syncAXIS control to use list handling mode “RepeatWhileBufferFull” (for this reason it is used in the “Installation_Project”). It is the most convenient way to handle large Jobs with syncAXIS control.

In list handling mode “RepeatWhileBufferFull”, syncAXIS-DLL reads the Job functions being called and buffers their transfer. If the RTC6 list memory should be full at any time, then syncAXIS-DLL waits at the respective Job function until there is free space again (for example, after the Job execution has been started).

Note that with list handling mode “RepeatWhileBufferFull” the syncAXIS control instance does not change to an error state, once the buffer is full. Instead, it waits silently for it to be freed by the user.

To achieve continuous Job reloading, 2 parallel threads are the simplest way

- (1) one for filling the RTC6 list memory
- (2) one for Job execution start

slsc_ctrl_get_exec_state is used to determine whether the Job is ready to get started.

The following is an example of how to use the list handling mode “RepeatWhileBufferFull” with parallel threads:

- Asynchronous thread 1: for filling the RTC6 list memory, see Figure 55, page 340
- Asynchronous thread 2: to start the Job execution, see Figure 56, page 341
- Automatic Job Start (as Soon as Ready for Execution) by Utilizing Callbacks, see Figure 57, page 342

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

size_t SLHandle;
slsc_cfg_initialize_from_file(&SLHandle, "syncAXISConfig.xml");

slsc_cfg_set_list_handling_mode(SLHandle, slsc_ListHandlingMode::slsc_ListHandlingMode_RepeatWhileBufferFull,
nullptr);

auto ListFilling = std::async(std::launch::async, [SLHandle]()
{
    size_t JobID;
    slsc_list_begin(SLHandle, &JobID);
    // Insert slsc_list_-functions here.
    slsc_list_end(SLHandle);
    return 0;
});

if (!startSingleJob(SLHandle))
{
    // !Insert proper error handling here!
}
slsc_cfg_delete(SLHandle);
```

Simplified code structure to execute bigger **Jobs** (that do exceed the RTC6 list memory) and list handling mode **RepeatWhileBufferFull**. Asynchronous Thread 1: For filling the RTC6 list memory.

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

size_t SLHandle;
slsc_cfg_initialize_from_file(&SLHandle, "syncAXISConfig.xml");

slsc_cfg_set_list_handling_mode(SLHandle, slsc_ListHandlingMode::slsc_ListHandlingMode_RepeatWhileBufferFull,
nullptr);

slsc_ExecState State = slsc_ExecState_Idle;
auto ListFilling = std::async(std::launch::async, [&]()
{
    while (State != slsc_ExecState_ReadyForExecution)
    {
        slsc_ctrl_get_exec_state(SLHandle, &State);
        std::this_thread::sleep_for(std::chrono::milliseconds(1));
    }
    slsc_ctrl_start_execution(SLHandle);
    return 0;
});

size_t JobID;
slsc_list_begin(SLHandle, &JobID);

// Insert slsc_list_-functions here.

slsc_list_end(SLHandle);
ListFilling.wait();
while (State != slsc_ExecState_Idle)
{
    slsc_ctrl_get_exec_state(SLHandle, &State);
    std::this_thread::sleep_for(std::chrono::milliseconds(1));
}

slsc_cfg_delete(SLHandle);
```

Simplified code structure to execute bigger **Jobs** (that do exceed the RTC6 list memory) and list handling mode **RepeatWhileBufferFull**. Asynchronous Thread 2: to start the **Job** execution.

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

struct Contents
{
    size_t SLHandle;
    uint32_t* Context;
}

size_t SLHandle;
slsc_cfg_initialize_from_file(&SLHandle, "syncAXISConfig.xml");

slsc_cfg_set_list_handling_mode(SLHandle, slsc_ListHandlingMode::slsc_ListHandlingMode_RepeatWhileBufferFull,
nullptr);

Contents Content = {SLHandle, &Context };

slsc_JobCallback AutoStart = [](size_t JobID, void* Context)
{
    slsc_ExecState State = slsc_ExecState_Idle;

    Contents*Content = static_cast<Contents*>(Context);
    size_t SLHandle = Content->SLHandle;

    while (State != slsc_ExecState_ReadyForExecution)
    {
        std::this_thread::sleep_for(std::chrono::milliseconds(1));
        slsc_ctrl_get_exec_state(SLHandle, &State);
    }
    std::cout << "Start Execution" << std::endl;
    slsc_ctrl_start_execution(SLHandle);

    return;
};

slsc_ExecTimeCallback PrintExecutionFinished = [](size_t JobID, uint64_t Progress, double ExecTime, void* Context)
{
    std::cout << "Execution finished after " << ExecTime << " sec!" << std::endl;
    return;
};

slsc_cfg_register_callback_job_loaded_enough(SLHandle, AutoStart, &Content);
slsc_cfg_register_callback_job_finished_executing(SLHandle, PrintExecutionFinished, &Content);

size_t JobID;
slsc_list_begin(SLHandle, &JobID);
// Insert slsc_list_-functions here.
slsc_list_end(SLHandle);

// Wait for Job end.
slsc_cfg_delete(SLHandle);
```

Simplified code structure to execute bigger **Jobs** (that do exceed the RTC6 list memory) and list handling mode **RepeatWhileBufferFull**. Automatic **Job** start (as Soon as Ready for Execution) utilizing **Callback functions**.

9.3 List Handling Mode “RepeatWhilePredicate”

In list handling mode “RepeatWhilePredicate”, users have more freedom, but also more responsibility loading the RTC6 list memory. For this mode, a user can program his own predicate that defines the behavior with full RTC6 list memory.

In the following example, see [Figure 58, page 343](#), the predicate is programmed in a way to mimic the list handling mode “RepeatWhileBufferFull”. Using this implementation, a user can for example, also modify the waiting time between each reloading.

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

size_t SLHandle;
slsc_cfg_initialize_from_file(&SLHandle, "syncAXISConfig.xml");

slsc_cfg_set_list_handling_mode(SLHandle, slsc_ListHandlingMode::slsc_ListHandlingMode_RepeatWhilePredicate,
[] (uint32_t RetVal)
{
    bool Flag = (0x0010 & RetVal == 0x0010);
    size_t BufferDelay = 10;
    if (Flag)
    {
        // Customize sleep time between each reloading
        std::this_thread::sleep_for(std::chrono::milliseconds(BufferDelay));
    }
    return Flag;
});

auto ListFilling = std::async(std::launch::async, [SLHandle] ()
{
    size_t JobID;
    slsc_list_begin(SLHandle, &JobID);
    // Insert slsc_list_-functions here.
    slsc_list_end(SLHandle);
    return 0;
});

if (!startSingleJob(SLHandle))
{
    // !Insert proper error handling here!
}
slsc_cfg_delete(SLHandle);
```

58

Simplified code structure to execute bigger **Jobs** (that do exceed the RTC6 list memory) and list handling mode “RepeatWhilePredicate”.


```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

// As in Figure 28, page 68, a syncAXIS control instance is initialized to record Modules.
// The content of each Module is a glyph, that is, only a single letter/digit from a specific font
// and in a certain font size only.
// The procedure must be repeated for each glyph in each font and size that is supposed to be
// available later. In the future, you can mark any character string using these Modules.

size_t JobID = 0;
size_t SLHandle = 0;

std::vector<std::string> Letters = { "X", "L", "S", "C", "A", "N" };
std::array<double, 2> StartPosition{ 0, 0 };

slsc_cfg_initialize_from_file(&SLHandle, "syncAXISConfig.xml");

for (std::string Letter : Letters)
{
    std::string Path = ("ModuleAlphabet\\" + Letter + ".slm");
    ModuleRecordingFinished = false;

    slsc_list_begin_module(SLHandle, &JobID, StartPosition.data(), Path.data());

    // Write vectors for the respective glyph here.

    slsc_list_end(SLHandle);

    // Wait for each Module recording to finalize.
    if (!startSingleJob(SLHandle))
    {
        // !Insert proper error handling here!
    }
}

slsc_cfg_delete(SLHandle);
```

Simplified code structure for recording several [Modules](#). Here, the content of each [Module](#) is a single glyph (X, L, S, C, A, N).

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

// As in Figure 29, page 69, an syncAXIS control instance is initialized to play back Modules.
// In this example, the Modules containing the glyphs are used to mark the text "XLSCAN".
// An offset (fixed spacing) is inserted between each letter.

size_t SLHandle = 0;
size_t JobID = 0;

std::vector<std::string> Letters = { "X", "L", "S", "C", "A", "N" };
std::array<double, 2> Position{ 0, 0 };
double Spacing = 3.;

slsc_cfg_initialize_from_file(&SLHandle, "syncAXISConfig.xml");

slsc_list_begin(SLHandle, &JobID);

for (std::string Letter : Letters)
{
    std::string Path = ("ModuleAlphabet\\" + Letter + ".slm");

    slsc_list_set_rot_and_offset_2d(SLHandle, 0, Position.data());
    Position[0] += Spacing;

    slsc_list_playback_module(SLHandle, Path.data());
}

slsc_list_end(SLHandle);

if (!startSingleJob(SLHandle))
{
    // !Insert proper error handling here!
}

slsc_cfg_delete(SLHandle);
```

Simplified code structure for replaying several [Modules](#) in sequence to create the text "XLSCAN".

11 Appendix D: Application Note – Avoiding Buffer Underruns by Using Modules

On the following pages you will find commented code sections showing how to use **Modules** to avoid a **Buffer underrun**:

- Part 1 of 4 Detecting a successful **Job** execution in **Figure 62, page 347**
- Part 2 of 4 Detecting a **Buffer underrun** in **Figure 63, page 348**
- Part 3 of 4 Recording a **Module** in **Figure 64, page 349**
- Part 4 of 4 Replaying a **Module** in **Figure 65, page 349**

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// To trigger execution of a single Job as soon as it is allowed.
// Then detect whether the execution has been successful or not.
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
bool startSingleJob(size_t SLHandle)
{
    slsc_ExecState State = slsc_ExecState_Idle;
    while (State != slsc_ExecState_ReadyForExecution)
    {
        uint32_t RetVal = slsc_ctrl_get_exec_state(SLHandle, &State);
        if (RetVal != 0 || State == slsc_ExecState_NotInitOrError)
        {
            return false;
        }
    }
    uint32_t RetVal = slsc_ctrl_start_execution(SLHandle);
    if (RetVal != 0)
    {
        return false;
    }
    while (State != slsc_ExecState_Idle)
    {
        uint32_t RetVal = slsc_ctrl_get_exec_state(SLHandle, &State);
        if (RetVal != 0 || State == slsc_ExecState_NotInitOrError)
        {
            return false;
        }
    }
    return true;
}
```

Simplified code structure, part 1 of 4 – Detecting a successful **Job** execution.

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

////////////////////////////////////
// The following lines demonstrate the detection of a Buffer underrun.
// With short vectors at high speeds, a Buffer underrun may occur
// which interrupts the processing and thus makes the workpiece unusable.
// To avoid this, one possible approach is to test all Jobs with the laser
// turned off (for example, slsc_ctrl_disable_laser) prior to the actual execution.
////////////////////////////////////
size_t SLHandle = 0;
slsc_cfg_initialize_from_file(&SLHandle, "syncAXISConfig.xml");

auto ListFilling = std::async(std::launch::async, [&SLHandle]()
{
    size_t JobID = 0;

    slsc_list_begin(SLHandle, &JobID);
    // Insert the respective vector functions here.
    slsc_list_end(SLHandle);

    return JobID;
});

bool SuccessfulExecution = startSingleJob(SLHandle);
ListFilling.wait();
bool BufferUnderrunOccurred = false;
// If an error, e.g. buffer underrun did occur, the processing may be incomplete.
if (!SuccessfulExecution)
{
    size_t ErrorCount = 0;
    slsc_ctrl_get_error_count(SLHandle, &ErrorCount);

    for (int i = 0; i < ErrorCount; i++)
    {
        uint64_t ErrorCode = 0;
        constexpr static size_t ErrorTextSize = 1000;
        std::array<char, ErrorTextSize> ErrorText;
        slsc_ctrl_get_error(SLHandle, i, &ErrorCode, ErrorText.data(), ErrorText.size());
        BufferUnderrunOccurred |= (ErrorCode == 0x00000000300000001);
    }
}
```

Simplified code structure, part 2 of 4 – Detecting a **Buffer underrun**.

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

////////////////////////////////////
// If a Job is likely to create a Buffer underrun, it is recommended to use Modules to avoid this.
// By slsc_cfg_initialize_copy (≥ V1.3.0), it is possible to create a syncAXIS control instance
// in simulation mode with the same configuration as an already active one.
// In this example, a syncAXIS control instance is still active with the Handle SLHandle.
////////////////////////////////////
size_t SLHandle = 1; // from previous run
size_t JobID = 0;
size_t ModuleHandle = 0;
slsc_cfg_initialize_copy(&ModuleHandle, SLHandle);

std::array<double, 2> Position{ 0, 0 };

// Record initial Job
slsc_list_begin_module(ModuleHandle, &JobID, Position.data(), "Module.slm");
// Insert the respective vector functions here.
slsc_list_end(ModuleHandle);

bool ModuleWritingSuccessful = startSingleJob(ModuleHandle);

slsc_cfg_delete(ModuleHandle);
```

64

Simplified code structure, part 3 of 4 – Recording a Module.

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

////////////////////////////////////
// After a Module has been recorded, it can simply be executed by initializing a
// new syncAXIS control instance and replaying the Module.
////////////////////////////////////
size_t SLHandle = 0;
size_t JobID = 0;

slsc_cfg_initialize_from_file(&SLHandle, "syncAXISConfig.xml");

// The Trajectory configuration is saved within the Module
// Replaying the Module.
slsc_list_begin(SLHandle, &JobID);
slsc_list_playback_module(SLHandle, "Module.slm");
slsc_list_end(SLHandle);

bool SuccessfulExecution = startSingleJob(SLHandle);

if (!SuccessfulExecution)
{
    // !Insert proper error handling here!
}

slsc_cfg_delete(SLHandle);
```

65

Simplified code structure, part 4 of 4 – Replaying a Module.

12 Appendix E: Application Note – C#

This chapter explains the main differences of the syncAXIS-DLL functions under C# (compared to C).

In this Chapter:

- Differences in the syncAXIS-DLL function signatures, page 350
- Differences in the Use of Callback Functions, page 351
- Code Example 1 (C#), page 352
- Code Example 2 (C#), page 357

12.1 Differences in the syncAXIS-DLL function signatures

- Notation der Datentypen, page 350
- Pointer-Replacements for C#, page 350

12.1.1 Notation der Datentypen

C, C++	C#
char*	string
uint32_t	uint
uint64_t	ulong

12.1.2 Pointer-Replacements for C#

C, Example		C#, Example
pointer	↔	out
uint32_t slsc_cfg_get_mode(size_t Handle, slsc_OperationMode* Mode);		uint slsc_cfg_get_mode(uint Handle, out slsc_OperationMode Mode);
pointer to constant	↔	array
uint32_t slsc_list_jump_abs(size_ t Handle, const double* Target);		uint slsc_list_jump_abs(uint Handle, double[] Target);
pointer to constant	↔	property
uint32_t slsc_cfg_set_trajectory_ config(size_t Handle, const slsc_TrajectoryConfig* TrajConfig);		uint slsc_cfg_set_trajectory_ config(uint Handle, slsc_TrajectoryConfig TrajConfig);
pointer to pointer	↔	property of property
uint32_t slsc_cfg_get_trajectory_ config(size_t Handle, slsc_TrajectoryConfig** TrajConfig);		slsc_TrajectoryConfig trajectoryConfig = syncAXIS.slsc_cfg_get_tr ajjectory_config(Handle); slsc_BlendModes blendMode = trajectoryConfig.Geometr yConfig.BlendMode;

12.2 Differences in the Use of Callback Functions

Callback functions basically work the same in C and C#. However, in C# you have to observe specific programming principles:

- In C#, the callbacks are implemented by inheritance of the callback classes, see [Code Example 1 \(C#\), page 352](#)
- In general, a specified **Callback function** is implemented on a **Callback event** of the respective type by overriding the `Run()` method
- Objects that are to be accessed during the **Callback event** (that is, when the `Run()` method is called) must be created as member variables of the derived callback class. These can, for example, be initialized by a constructor.
- You could implement the required callback “return values” in the derived class by (alternatively):
 - User-defined methods
 - Properties



12.3 Code Example 1 (C#)

```
// C# code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.IO;
using System.Threading;

namespace MyDemo
{
    class DemoCode
    {
        static void Main(string[] args)
        {
            string xmlFilePath = Directory.GetCurrentDirectory() + "\\..\\..\\..\\..\\syncAXISConfig.xml";
            uint handle = 0;
            Console.WriteLine(xmlFilePath);

            uint retVal = syncAXIS.slsc_cfg_initialize_from_file(out handle, xmlFilePath);

            // Example - public static uint slsc_cfg_set_list_handling_mode_with_context(uint Handle,
            //                               slsc_ListHandlingMode mode, ListHandlingCallback internal_CALLBACK);
            slsc_ListHandlingMode mode = slsc_ListHandlingMode.slsc_ListHandlingMode_RepeatWhilePredicate;
            if (mode == slsc_ListHandlingMode.slsc_ListHandlingMode_RepeatWhilePredicate)
            {
                MyListHandlingPredicate listHandlingPredicate = new MyListHandlingPredicate(retVal);
                retVal |= syncAXIS.slsc_cfg_set_list_handling_mode_with_context(handle, mode, listHandlingPredicate);
            }
            else
            {
                // Default constructor
                ListHandlingCallback listHandlingCallback = new ListHandlingCallback();
                retVal |= syncAXIS.slsc_cfg_set_list_handling_mode_with_context(handle, mode, listHandlingCallback);
            }

            // Example - public static slsc_TrajectoryConfig slsc_cfg_get_trajectory_config(uint Handle);
            // No deletion needed.
            slsc_TrajectoryConfig trajectoryConfig = syncAXIS.slsc_cfg_get_trajectory_config(handle);
            Console.WriteLine("Original GeometryConfig Blend Mode = " + trajectoryConfig.GeometryConfig.BlendMode);

            // Example - public static uint slsc_cfg_set_trajectory_config(uint Handle,
            //                               slsc_TrajectoryConfig TrajConfig);
            trajectoryConfig.GeometryConfig.BlendMode = slsc_BlendModes.slsc_BlendModes_FixedBlending;
            syncAXIS.slsc_cfg_set_trajectory_config(handle, trajectoryConfig);
            Console.WriteLine("New GeometryConfig Blend Mode = " + trajectoryConfig.GeometryConfig.BlendMode);
        }
    }
}
```




```
// Example - public static uint slsc_cfg_register_callback_job_start_planned(uint Handle,
//                               JobCallback internal_CALLBACK);
Job_start_planned planningStarted = new Job_start_planned(handle);
retVal |= syncAXIS.slsc_cfg_register_callback_job_start_planned(handle, planningStarted);

// Example - public static uint slsc_cfg_register_callback_job_loaded_enough(uint Handle,
//                               JobCallback internal_CALLBACK);
AutoStart starter = new AutoStart(handle);
retVal |= syncAXIS.slsc_cfg_register_callback_job_loaded_enough(handle, starter);

// Example - public static uint slsc_cfg_register_callback_job_finished_executing(uint Handle,
//                               execTimeCallback internal_CALLBACK);
WaitForFinished waiter = new WaitForFinished();
retVal |= syncAXIS.slsc_cfg_register_callback_job_finished_executing(handle, waiter);

uint jobID = 1;

retVal |= syncAXIS.slsc_list_begin(handle, out jobID);

// Simple Job
double[] lowerMid = new double[] { 0.0, 0.0 };
double[] lowerRightCorner = new double[] { 10.0, 0.0 };
double[] upperRightCorner = new double[] { 10.0, 10.0 };
double[] upperLeftCorner = new double[] { -10.0, 10.0 };
double[] lowerLeftCorner = new double[] { -10.0, 0.0 };
retVal |= syncAXIS.slsc_list_jump_abs(handle, lowerMid);
retVal |= syncAXIS.slsc_list_mark_abs(handle, lowerRightCorner);
retVal |= syncAXIS.slsc_list_mark_abs(handle, upperRightCorner);
retVal |= syncAXIS.slsc_list_mark_abs(handle, upperLeftCorner);
retVal |= syncAXIS.slsc_list_mark_abs(handle, lowerLeftCorner);
retVal |= syncAXIS.slsc_list_mark_abs(handle, lowerMid);

retVal |= syncAXIS.slsc_list_end(handle);

while (waiter.GetFinishedState() == false)
{
    Thread.Sleep(100);
}

Console.WriteLine("Return Value = " + retVal);

WriteError(handle);

syncAXIS.slsc_cfg_delete(handle);

Console.ReadKey();
}
```



```
// Example - ListHandlingCallback internal_CALLBACK
public class MyListHandlingPredicate : ListHandlingCallback
{
    uint returnVal;
    bool flag;
    uint bufferFull = 0x0010;

    public MyListHandlingPredicate(uint returnValIn)
    {
        returnVal = returnValIn;
    }
    public override bool Run(uint returnValIn)
    {
        uint indicator = bufferFull & returnVal;

        flag = (indicator == bufferFull);

        if (flag)
        {
            Console.WriteLine("Buffer is full. List loading waits for free buffer.");
            return flag;
        }
        else
        {
            return flag;
        }
    }
}

// Example - JobCallback internal_CALLBACK callback_job_start_planned
public class Job_start_planned : JobCallback
{
    uint handle;

    public Job_start_planned(uint handleIn)
    {
        handle = handleIn;
    }

    public override void Run(uint jobID)
    {
        Console.WriteLine("Planning started.");
    }
}
```



```
// Example - JobCallback internal_CALLBACK callback_job_loaded_enough
public class AutoStart : JobCallback
{
    uint handle;

    public AutoStart(uint handleIn)
    {
        handle = handleIn;
    }

    public override void Run(uint jobID)
    {
        slsc_ExecState rtc6State;
        uint retVal;

        retVal = syncAXIS.slsc_ctrl_get_exec_state(handle, out rtc6State);
        if (retVal != 0)
        {
            Console.WriteLine("An Error occurred after slsc_ctrl_get_exec_state, return value = " + retVal);
        }
        while (rtc6State != slsc_ExecState.slsc_ExecState_ReadyForExecution)
        {
            retVal = syncAXIS.slsc_ctrl_get_exec_state(handle, out rtc6State);
            if (retVal != 0)
            {
                Console.WriteLine("An Error occurred after slsc_ctrl_get_exec_state, return value = " + retVal);
            }
            Thread.Sleep(10);
            Console.WriteLine("Waiting 10 ms for execution ready to run...");
        }

        retVal = syncAXIS.slsc_ctrl_start_execution(handle);
        if (retVal != 0)
        {
            Console.WriteLine("An Error occurred after slsc_ctrl_start_execution. Return value = " + retVal);
        }
        else
        {
            Console.WriteLine("Execution started.");
        }
    }
}
```



```
// Example - JobCallback internal_CALLBACK callback_job_finished_executing
public class WaitForFinished : ExecTimeCallback
{
    bool isFinished = false;

    public bool GetFinishedState()
    {
        return isFinished;
    }

    public override void Run(uint jobID, ulong progress, double execTime)
    {
        isFinished = true;
        Console.WriteLine("Execution finished. Execution time: " + execTime + " sec.");
    }
}

private static void WriteError(uint handle)
{
    uint count = 0;
    syncAXIS.slsc_ctrl_get_error_count(handle, out count);
    if (count > 0)
    {
        Console.WriteLine(count + " errors detected.");
        for (uint i = 0; i < count; i++)
        {
            ulong ErrorNr = 0;
            string ErrorMessage = "";
            syncAXIS.slsc_ctrl_get_error(handle, i, out ErrorNr, out ErrorMessage);
            Console.WriteLine("ErrorMessage: " + ErrorMessage);
        }
    }
    else
    {
        Console.WriteLine("No error occurred!");
    }
    return;
}
}
```



12.4 Code Example 2 (C#)

```
// C++ code section for educational purposes only.
// Do not execute this code on actual XL SCAN systems without prior modification and simulation!
// Observe the safety notices and disclaimer on page 16.
// Example MultiPara by slsc_list_multi_para_arc_abs
// = multi-part (more complex) Ramp
// No blending curves are activated.
// Pseudo code (not complete)

uint jobID = 0;
syncAXIS.slsc_list_begin(handle, out jobID);
double[] TargetPosition_0 = new double[] { 0.0, 0.0 };
double[] TargetPosition_1 = new double[] { 1.0, 1.0 };
double[] TargetPosition_2 = new double[] { 1.025, 1.025 };
double[] TargetPosition_3 = new double[] { 2.0, 1.0 };
double[] TargetPosition_4 = new double[] { 3.0, 0.0 };

syncAXIS.slsc_list_jump_abs(handle, TargetPosition_0);
syncAXIS.slsc_list_mark_abs(handle, TargetPosition_1);
double[] TargetParaDefault = new double[] { 1.0 };
syncAXIS.slsc_list_para_enable(handle, TargetParaDefault);

// Here is created: ArrayList of type slsc_ParaChannel.
var Channell = new slsc_ParaChannel();
Channell.Add(new slsc_ParaSection { ds = 0.25, ParaTarget = 0.5 });
Channell.Add(new slsc_ParaSection { ds = 0.6186678697087737, ParaTarget = 0.5 });
Channell.Add(new slsc_ParaSection { ds = 0.25, ParaTarget = 1.0 });

// Here is created: ArrayList of type slsc_MultiParaChannels (ActiveChannel == 1).
var Paras = new slsc_MultiParaChannels();
Paras.Add(Channell);
syncAXIS.slsc_list_multi_para_arc_abs(handle, TargetPosition_2, TargetPosition_3, Paras);

syncAXIS.slsc_list_mark_abs(handle, TargetPosition_4);
syncAXIS.slsc_list_jump_abs(handle, TargetPosition_0);

syncAXIS.slsc_list_end(handle);
```

13 Appendix F: Reference of syncAXISConfig.xml Tags

13.1 xml-Structure Overview

- Legend
 - (*) This tag is optional
 - [] This tag is allowed to occur more than once.
 - {} 2 mutually exclusive meanings: this tag allows exactly 1 child tag ("choice") or any number of child tags ("sequence").
 - Container, STANDARD, STANDARD***, NON-ST'D, NON-ST'D*** is the **Behavior on Module replay**, page 66 of this tag

Configuration	Container	ScanDeviceConfig	Container
GeneralConfig	Container	DynamicLimits	Container
ACSController(*)	STANDARD	Velocity	NON-ST'D***
InitialOperationMode	STANDARD***	Acceleration	NON-ST'D***
InitialListHandlingMode	STANDARD	Jerk	NON-ST'D***
DynamicViolationReaction	STANDARD	CalculationDynamics	Container
LogConfig	Container	MarkDynamics	Container
LogfilePath	STANDARD	Acceleration	NON-ST'D
LogLevel	STANDARD	Jerk	NON-ST'D
EnableConsoleLogging	STANDARD	JumpDynamics	Container
EnableFileLogging	STANDARD	Acceleration	NON-ST'D
MaxLogfileSize(*)	STANDARD	Jerk	NON-ST'D
MaxBackupFileCount(*)	STANDARD	FieldLimits	Container
BaseDirectoryPath(*)	STANDARD	XDirection	STANDARD
SimulationConfig	Container	YDirection	STANDARD
SimulationMode	STANDARD	ZDirection(*)	STANDARD
SimOutputFileDirectory(*)	STANDARD	MonitoringLevel	STANDARD
BinaryOutput(*)	STANDARD	FocalLength(*)	STANDARD
DisableFileOutput(*)	STANDARD	Delay(*)	STANDARD***
RTCCConfig	Container	ScanDeviceList	Container
BoardIdentificationMethod	STANDARD	ScanDevice[]	STANDARD
ProgramFileDirectory	STANDARD	CorrectionFileList	Container
Boards	Container	CorrectionFilePath[]	STANDARD
RTC6[]	Container	Alignment	Container
SerialNumber(*)	STANDARD	Matrix	Container
HeadA	STANDARD	T11	STANDARD
HeadB	STANDARD	T12	STANDARD
EthSearch{ }(*)	Container	T21	STANDARD
Broadcast	Container	T22	STANDARD
IP	STANDARD	Offset	STANDARD
NetMask	STANDARD	BasePartDisplacement	Container
IPScan	Container	Matrix	Container
StartIp	STANDARD	T11	STANDARD
EndIp	STANDARD	T12	STANDARD
IPList	Container	T21	STANDARD
IPAddress[]	STANDARD	T22	STANDARD
EthMaxTimeout(*)	STANDARD	Offset	STANDARD
		DefaultCorrectionFile	STANDARD

LaserConfig	Container	TrajectoryConfig	Container
LaserMode ^(*)	STANDARD	MarkConfig	Container
LaserPortCfg ^(*)	Container	JumpSpeed	NON-ST'D
LaserOn	STANDARD	MarkSpeed	NON-ST'D
Laser1	STANDARD	MinimalMarkSpeed	NON-ST'D
Laser2	STANDARD	LaserSwitchConfig ^(*)	Container
LaserOutput ^(*)	STANDARD	LaserPreTriggerTime	STANDARD***
LaserStandby ^(*)	STANDARD	LaserSwitchOffsetTime	STANDARD
QSwitchDelay ^(*)	STANDARD	LaserMinOffTime ^(*)	NON-ST'D
FPulseKillerLength ^(*)	STANDARD	GeometryConfig	Container
LaserControlFlags ^(*)	Container	MaxBlendRadius	NON-ST'D
LaserDisable ^(*)	STANDARD	ApproxBlendLimit	NON-ST'D
PulseSwitchSetting ^(*)	STANDARD	BlendMode	NON-ST'D
LaserSignalPhaseShift ^(*)	STANDARD	AutoCyclicGeometry	NON-ST'D
LaserOnSignalActiveLow ^(*)	STANDARD	SplineConversionLengthLimit	NON-ST'D
Laser1Laser2SignalActiveLow ^(*)	STANDARD	SplineMode	NON-ST'D
LaserPulsesAtRisingEdge ^(*)	STANDARD	VectorResolution	STANDARD
OutputSynchronizationOn ^(*)	STANDARD	StageConfig	Container
AutomaticLaserControl	Container	DelayShift ^(*)	STANDARD***
ActiveChannel	Container	CTIME ^(*)	STANDARD***
Channel[] ^(*)	STANDARD***	MonitoringLevel	STANDARD
AnalogOut1 ^(*)	STANDARD	StageList ^(*)	Container
Shift	STANDARD	Stage[]	STANDARD
RadiusFactor	STANDARD	FieldLimits	Container
DataPoint[] ^(*)	STANDARD	XDirection	STANDARD
VelocityFactor	STANDARD	YDirection	STANDARD
DataPoint[] ^(*)	STANDARD	ZDirection ^(*)	STANDARD
AnalogOut2 ^(*)	STANDARD	DynamicLimits	Container
Shift	STANDARD	Velocity	STANDARD
RadiusFactor	STANDARD	Acceleration	STANDARD
DataPoint[] ^(*)	STANDARD	Jerk	STANDARD
VelocityFactor	STANDARD	CalculationDynamics	Container
DataPoint[] ^(*)	STANDARD	Velocity	NON-ST'D***
PulseLength ^(*)	STANDARD	Acceleration	NON-ST'D***
Shift	STANDARD	Jerk	NON-ST'D***
RadiusFactor	STANDARD	Alignment	Container
DataPoint[] ^(*)	STANDARD	Matrix	Container
VelocityFactor	STANDARD	T11	STANDARD
DataPoint[] ^(*)	STANDARD	T12	STANDARD
HalfPeriod ^(*)	STANDARD	T21	STANDARD
Shift	STANDARD	T22	STANDARD
RadiusFactor	STANDARD	Offset	STANDARD
DataPoint[] ^(*)	STANDARD	StageAxisX ^(*)	STANDARD
VelocityFactor	STANDARD	StageAxisY ^(*)	STANDARD
DataPoint[] ^(*)	STANDARD	SlecEtherCATNodeID ^(*)	STANDARD
SpotDistance ^(*)	STANDARD		
Shift	STANDARD		
RadiusFactor	STANDARD		
DataPoint[] ^(*)	STANDARD		
VelocityFactor	STANDARD		
DataPoint[] ^(*)	STANDARD		

IOConfig ^(*)	Container
DefaultOutputs ^(*)	Container
LaserPinOut ^(*)	STANDARD
AnalogOut1 ^(*)	STANDARD
AnalogOut2 ^(*)	STANDARD
LaserInitSequence{ } ^(*)	Container
Delay	STANDARD
SetLaserPinOut	STANDARD
SetAnalogOut1	STANDARD
SetAnalogOut2	STANDARD
SetExt1DigitalOut	STANDARD
LaserShutdownSequence{ } ^(*)	Container
Delay	STANDARD
SetLaserPinOut	STANDARD
SetAnalogOut1	STANDARD
SetAnalogOut2	STANDARD
SetExt1DigitalOut	STANDARD
MotionDecompositionConfig	Container
FilterBandwidth	STANDARD
HeuristicConfig ^(*)	Container
DynamicReductionFunctions ^(*)	Container
DynamicReductionFunction[] ^(*)	Container
DataPoint[]	NON-ST'D
HeuristicForJumpsOnly ^(*)	NON-ST'D
SystemConfig* INTERN - NICHT VERWENDEN	

13.2 xml Tags

XML tag	Configuration
XML signature (incl. defaults)	Configuration ''=optional; no ''=mandatory.
XML path(s)	<cfg:Configuration>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre><cfg:Configuration xmlns:cfg="syncAXIS" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="cfg syncAXIS_1_8.xsd" Version="1.7"> <!-- allowed/possible child tags Configuration in the XML structure overview --> </cfg:Configuration></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> Configuration is the topmost container tag in syncAXISConfig.xml.
Version info	syncAXIS_1_8.xsd

XML tag	GeneralConfig
XML signature (incl. defaults)	GeneralConfig ''=optional; no ''=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre><cfg:GeneralConfig> <!-- allowed/possible child tags GeneralConfig in the XML structure overview --> </cfg:GeneralConfig></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> GeneralConfig is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	ACSController
XML signature (incl. defaults)	ACSController* = 127.0.0.1
	<p>'*' = optional; no '*' = mandatory.</p> <p>ACSController value: string (format 'N.N.N.N').</p>
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:ACSController>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:ACSController>127.0.0.1</cfg:ACSController>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Sets the IP address of the ACS Motion Controller in the EtherCAT network. Prior the first startup, make sure that the ACSController value is already set. Even for simulation mode, the ACSController value must be a valid entry. See also <code>0x 00 00 00 06 00 00 00 01 INIT_ACS_TCPIP</code>.
Version info	syncAXIS_1_8.xsd

XML tag	InitialOperationMode
XML signature (incl. defaults)	InitialOperationMode = ScannerAndStage
	'*' = optional; no '*' = mandatory. InitialOperationMode value: string.
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:InitialOperationMode>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:InitialOperationMode> ScannerAndStage </cfg:InitialOperationMode>
Settable via API?	slsc_cfg_set_mode
Behavior on Module replay	The parameter value of the replaying syncAXIS control instance is applied. Exceptions: See Table page 66 . See also Section "Behavior on Module replay" , page 66.
Comment(s)	<ul style="list-style-type: none"> Sets the initial Operation mode of the syncAXIS control instance, see also enum slsc_OperationMode. Allowed entries: <ul style="list-style-type: none"> ScannerOnly StageOnly ScannerAndStage The Operation mode is a fundamental parameter for the Trajectory planning and defines the type of motion.
Version info	syncAXIS_1_8.xsd

XML tag	InitialListHandlingMode
XML signature (incl. defaults)	InitialListHandlingMode = ReturnAtOnce
	<p>'*' = optional; no '*' = mandatory.</p> <p>InitialListHandlingMode value: string.</p>
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:InitialListHandlingMode>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:InitialListHandlingMode> RepeatWhileBufferFull </cfg:InitialListHandlingMode>
Settable via API?	slsc_cfg_set_list_handling_mode slsc_cfg_set_list_handling_mode_with_context
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> • Sets the initial list handling mode of the syncAXIS control instance, see also enum slsc_ListHandlingMode. This setting specifies how buffering of the Job functions (slsc_list_*) is handled. • Allowed entries: <ul style="list-style-type: none"> – ReturnAtOnce – RepeatWhileBufferFull • Notice: "RepeatWhilePredicate" is only settable via the API! • In list handling mode "RepeatWhileBufferFull" or "RepeatWhilePredicate", the syncAXIS control instance is continuously trying to load positions to the RTC6 board, "patiently" waiting for the buffer to be freed once it is full (which can be achieved by starting the Job execution). <i>Therefore, make sure that you fill the List asynchronously to the Job start thread!</i> • If the list handling mode is set to "ReturnAtOnce", code 0x10 with the function that is overloading the buffer is returned.
Version info	syncAXIS_1_8.xsd

XML tag	DynamicViolationReaction
XML signature (incl. defaults)	DynamicViolationReaction = WarningOnly
	<p>'*' = optional; no '*' = mandatory.</p> <p>DynamicViolationReaction value: string.</p>
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:DynamicViolationReaction>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:DynamicViolationReaction>AbortImmediately</cfg:DynamicViolationReaction>
Settable via API?	slsc_cfg_set_dynamic_violation_reaction
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Defines which reaction occurs automatically, as soon as a certain monitoring criterion (dynamic limits, working field limits) is violated. Applies to hardware mode as well as simulation mode. Note: The monitoring criteria associated with DynamicViolationReaction are set separately for scan devices and positioning stages under: <ul style="list-style-type: none"> <cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:MonitoringLevel> <cfg:Configuration> → <cfg:StageConfig> → <cfg:MonitoringLevel> Allowed entries: <ul style="list-style-type: none"> WarningOnly AbortImmediately StopAndReport syncAXIS control uses to monitor working field and dynamics as: <ul style="list-style-type: none"> scan device dynamic limits the DynamicLimits values, page 388 scan device working field limits the FieldLimits values, page 395 scan device monitoring criterion the MonitoringLevel value, page 397 positioning stage dynamic limits the FieldLimits values, page 454 positioning stage working field limits the DynamicLimits values, page 456 positioning stage monitoring criterion the MonitoringLevel values, page 452 reaction on exceedances the DynamicViolationReaction values, page 365
Version info	syncAXIS_1_8.xsd

XML tag	LogConfig
XML signature (incl. defaults)	LogConfig ''=optional; no ''=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:LogConfig>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:LogConfig> <!-- allowed/possible child tags LogConfig in the XML structure overview --> </cfg:LogConfig></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> GeneralConfig is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	LogfilePath
XML signature (incl. defaults)	LogfilePath = Error.log
	<p>'*' = optional; no '*' = mandatory.</p> <p>LogfilePath value: string.</p>
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:LogConfig> → <cfg:LogfilePath>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:LogfilePath>[BaseDirectoryPath]/Log</cfg:LogfilePath>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Path of the log file that is created by the syncAXIS control instance, if file logging is switched on (see EnableFilelogging). For [BaseDirectoryPath], see BaseDirectoryPath. If this file already exists, logging information is appended.
Version info	syncAXIS_1_8.xsd

XML tag	LogLevel
XML signature (incl. defaults)	LogLevel = Warn '*'=optional; no '*'=mandatory. LogLevel value: string.
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:LogConfig> → <cfg:LogLevel>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:LogLevel>Warn</cfg:LogLevel>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> The LogLevel value sets the logging level, see Chapter 2.8 "About the Logging in syncAXIS control", page 47. Allowed entries: <ul style="list-style-type: none"> Error Generated are: [ERROR] log file lines Warn Generated are: [ERROR] log file lines [WARN] log file lines Info Generated are: [ERROR] log file lines [WARN] log file lines [INFO] log file lines [ERROR] log file lines indicate that the system is in error state. Example log file lines: <pre>19-10-04 18:16:10:669 [ERROR] 18:16:10.669762 ErrorCode: 0x0000000500000001 ErrorMessage: "Init failed!" 19-09-19 16:14:21:930 [WARN] Scanner dynamic breached or violated position limits: 0. Regarding Job 1 in Segment 0 19-10-08 18:16:30:508 [INFO] RTC6-SLEC establish handshake, activating stage</pre>
Version info	syncAXIS_1_8.xsd

XML tag	EnableConsoleLogging
XML signature (incl. defaults)	EnableConsoleLogging = false
	'*' = optional; no '*' = mandatory. EnableConsoleLogging value: boolean.
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:LogConfig> → <cfg:EnableConsoleLogging>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:EnableConsoleLogging>true</cfg:EnableConsoleLogging>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Specifies whether console logging is to be switched on or off. See Chapter 5 "Error Codes with slsc_ctrl_get_error, Log File and Console", page 282.
Version info	syncAXIS_1_8.xsd

XML tag	EnableFilelogging
XML signature (incl. defaults)	EnableFilelogging = true
	'*' = optional; no '*' = mandatory. EnableFilelogging value: boolean.
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:LogConfig> → <cfg:EnableFilelogging>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:EnableFilelogging>true</cfg:EnableFilelogging>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Specifies whether file logging is to be switched on or off. See Chapter 5 "Error Codes with slsc_ctrl_get_error, Log File and Console", page 282.
Version info	syncAXIS_1_8.xsd

XML tag	MaxLogfileSize
XML signature (incl. defaults)	MaxLogfileSize* = 5242880
	'*' = optional; no '*' = mandatory. MaxLogfileSize value: positive integer.
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:LogConfig> → <cfg:MaxLogfileSize>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:MaxLogfileSize>26214400</cfg:MaxLogfileSize>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Specifies the maximum size of the individual log files.
Version info	syncAXIS_1_8.xsd

XML tag	MaxBackupFileCount
XML signature (incl. defaults)	MaxBackupFileCount* = 10
	'*' = optional; no '*' = mandatory. MaxBackupFileCount value: non-negative integer.
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:LogConfig> → <cfg:MaxBackupFileCount>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:MaxBackupFileCount>0</cfg:MaxBackupFileCount>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Specifies how many files are to be backed up (by adding a counting integer to the file name) before the current file is going to be continuously overwritten.
Version info	syncAXIS_1_8.xsd

XML tag	BaseDirectoryPath
XML signature (incl. defaults)	BaseDirectoryPath* = "" ''=optional; no ''=mandatory. "" means empty string. BaseDirectoryPath value: string.
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:BaseDirectoryPath>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:BaseDirectoryPath>\${BASE_PATH}</cfg:BaseDirectoryPath>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> The BaseDirectoryPath value functions as macro which can be inserted as '[BaseDirectoryPath]' with certain tags (SimOutputFileDirectory, LogfilePath, ProgramFileDirectory). Its purpose is to make folderpath specifications shorter and easier to change.
Version info	syncAXIS_1_8.xsd

XML tag	SimulationConfig
XML signature (incl. defaults)	SimulationConfig
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:SimulationConfig>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:SimulationConfig> <!-- allowed/possible child tags SimulationConfig in the XML structure overview --> </cfg:SimulationConfig> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> GeneralConfig is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	SimulationMode
XML signature (incl. defaults)	SimulationMode = true
	<p>'*' = optional; no '*' = mandatory.</p> <p>SimulationMode value: boolean.</p>
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:SimulationConfig> → <cfg:SimulationMode>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:SimulationMode>true</cfg:SimulationMode>
Settable via API?	slsc_cfg_set_simulation_setting
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> The SimulationMode value specifies whether the simulation mode or hardware mode is to be switched on. See also Chapter 2.4 "About Initializing syncAXIS control-based User Programs", page 26 and Chapter 2.5 "About the syncAXIS control Simulation Mode", page 31. If true: the syncAXIS control instance is not going to communicate with any hardware – except the Dongle. The Trajectory planning results are written to simulation files, if the simulation file generation is switched on by DisableFileOutput.
Version info	syncAXIS_1_8.xsd

XML tag	SimOutputFileDirectory
XML signature (incl. defaults)	<p>SimOutputFileDirectory* = ""</p> <p>'*' = optional; no '*' = mandatory.</p> <p>"" means empty string.</p> <p>SimOutputFileDirectory value: string.</p>
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:SimulationConfig> → <cfg:SimOutputFileDirectory>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358 .
XML section example	<cfg:SimOutputFileDirectory>[BaseDirectoryPath]/Simulate/</cfg:SimOutputFileDirectory>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> For [BaseDirectoryPath], see BaseDirectoryPath. The SimOutputFileDirectory value defines the location where the simulation file (see DisableFileOutput) is to be saved. For the simulation file naming, see page 31.
Version info	syncAXIS_1_8.xsd

XML tag	BinaryOutput
XML signature (incl. defaults)	BinaryOutput* = false
	'*' = optional; no '*' = mandatory. BinaryOutput value: boolean.
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:SimulationConfig> → <cfg:BinaryOutput>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358 .
XML section example	<pre><cfg:BinaryOutput>true</cfg:BinaryOutput></pre> <p>=> The simulation files (see DisableFileOutput) are generated in binary format.</p> <pre><cfg:BinaryOutput>>false</cfg:BinaryOutput></pre> <p>=> The simulation files (see DisableFileOutput) are generated in ASCII format.</p>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Defines whether simulation files are generated in ASCII or binary format. ASCII format is well suited for small (sample) Jobs. The resulting simulation files can still be analyzed well by a human with the help of a text editor. Binary format is more suitable for large Jobs. Compared to ASCII format, the resulting simulation files are written faster, are smaller, syncAXIS Viewer can load them faster and are also not subject to any size limitation.
Version info	syncAXIS_1_8.xsd

XML tag	DisableFileOutput
XML signature (incl. defaults)	DisableFileOutput* = false
	<p>''=optional; no ''=mandatory.</p> <p>DisableFileOutput value: boolean.</p>
XML path(s)	<cfg:Configuration> → <cfg:GeneralConfig> → <cfg:SimulationConfig> → <cfg:DisableFileOutput>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<p><cfg:DisableFileOutput>false</cfg:DisableFileOutput></p> <p>=> A simulation file (see DisableFileOutput) is generated.</p> <p>For the simulation file naming, see page 31.</p> <p><cfg:DisableFileOutput>true</cfg:DisableFileOutput></p> <p>=> A simulation file (see DisableFileOutput) is <i>not</i> generated.</p>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Defines whether a simulation file is generated or not. The simulation process is much faster with <code>true</code> than with <code>false</code>. The setting <code>true</code> is recommended when (not the entire trajectory, but only) the Job characteristics ("Key", see enum slsc_JobCharacteristic) are of interest: the max. position and dynamic values can be queried by slsc_ctrl_get_job_characteristic.
Version info	syncAXIS_1_8.xsd

XML tag	RTCCConfig
XML signature (incl. defaults)	RTCCConfig
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:RTCCConfig>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:RTCCConfig> <!-- allowed/possible child tags RTCCConfig in the XML structure overview --> </cfg:RTCCConfig> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> RTCCConfig is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	BoardIdentificationMethod
XML signature (incl. defaults)	BoardIdentificationMethod = UseFirstFound
	<p>'*' = optional; no '*' = mandatory.</p> <p>BoardIdentificationMethod value: string.</p>
XML path(s)	<cfg:Configuration> → <cfg:RTCConfig> → <cfg:BoardIdentificationMethod>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:BoardIdentificationMethod>BySerialNumber</cfg:BoardIdentificationMethod>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Allowed entries: <ul style="list-style-type: none"> UseFirstFound BySerialNumber Determines which RTC6 is to be used for this syncAXIS control instance. If there is only one, with "UseFirstFound" the first RTC6 found by the RTC6DLL.dll is used. Using "BySerialNumber" a specific RTC6 can be selected by its serial number, see SerialNumber.
Version info	syncAXIS_1_8.xsd

XML tag	ProgramFileDirectory
XML signature (incl. defaults)	ProgramFileDirectory* = "" '*'=optional; no '*'=mandatory. "" means empty string. ProgramFileDirectory value: string.
XML path(s)	<cfg:Configuration> → <cfg:RTCConfig> → <cfg:ProgramFileDirectory>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358 .
XML section example	<cfg:ProgramFileDirectory>[BaseDirectoryPath]/../RTC6</cfg:ProgramFileDirectory>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> • Example: <cfg:ProgramFileDirectory>[BaseDirectoryPath]/../RTC6</cfg:ProgramFileDirectory> • For [BaseDirectoryPath], see BaseDirectoryPath. • Sets the folder path to the RTC6 files RTC6RBF.rbf, RTC6DAT.dat and RTC6OUT.out. • These are loaded onto the RTC6: <ul style="list-style-type: none"> – when initializing the syncAXIS control instance after every RTC6 power cycle (that is, with RTC6 PCI Express Boards the PC startup) – when a version mismatch (between the RTC6DLL.dll in use and the RTC6 files running on the RTC6) occurred, for example, if iSCANcfg.exe has been started previously with different RTC6 files. – Upon MasterSlaveSynchronizer.exe start (from the syncAXISConfig.xml which have been specified as calling parameter) • Note that loading RTC6 files: <ul style="list-style-type: none"> – resets the RTC6 board – restarts the clock – destroys a synchronization (between all RTC6 boards and also the ACS EtherCAT network) previously achieved by MasterSlaveSynchronizer.exe
Version info	syncAXIS_1_8.xsd

XML tag	Boards
XML signature (incl. defaults)	Boards
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:RTCConfig> → <cfg:Boards>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:Boards> <!-- allowed/possible child tags Boards in the XML structure overview --> </cfg:Boards></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> Boards is <i>only</i> a container tag. No value(s), no attribute(s). With the child tags of Boards, all to-be-used RTC6 boards are configured.
Version info	syncAXIS_1_8.xsd

XML tag	RTC6
XML signature (incl. defaults)	RTC6[]
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:RTCConfig> → <cfg:Boards> → <cfg:RTC6>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:RTC6> <!-- allowed/possible child tags RTC6 in the XML structure overview --> </cfg:RTC6></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> RTC6 is <i>only</i> a container tag. No value(s), no attribute(s). syncAXIS_1_8.xsd allows up to 4 RTC6 tags.
Version info	syncAXIS_1_8.xsd

XML tag	SerialNumber
XML signature (incl. defaults)	SerialNumber* = 0
	'*' = optional; no '*' = mandatory. SerialNumber value: non-negative integer.
XML path(s)	<cfg:Configuration> → <cfg:RTCConfig> → <cfg:Boards> → <cfg:RTC6> → <cfg:SerialNumber>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:SerialNumber>123457</cfg:SerialNumber>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Defines the serial number of the RTC6 board to which the hardware specified under HeadA and HeadB is connected. The SerialNumber value is only evaluated, if "BySerialNumber" is specified at BoardIdentificationMethod.
Version info	syncAXIS_1_8.xsd

XML tag	HeadA
XML signature (incl. defaults)	HeadA
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:RTCConfig> → <cfg:Boards> → <cfg:RTC6> → <cfg:HeadA>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:HeadA> ScanDevice1 </cfg:HeadA>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Scan device or positioning stage connected to the first scan head connector (SCANHEAD). Allowed entries: see enum slsc_ScanDevice and enum slsc_Stage.
Version info	syncAXIS_1_8.xsd

XML tag	HeadB
XML signature (incl. defaults)	HeadB
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:RTCConfig> → <cfg:Boards> → <cfg:RTC6> → <cfg:HeadB>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:HeadB>Stage1</cfg:HeadB> or, <cfg:HeadB>None</cfg:HeadB>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Scan device or positioning stage connected to the second scan head connector (2. SCANHEAD). Allowed entries: see enum slsc_ScanDevice and enum slsc_Stage.
Version info	syncAXIS_1_8.xsd

XML tag	EthSearch
XML signature (incl. defaults)	EthSearch{ }*
	'*' = optional; no '*' = mandatory. '{ }' = here: only <i>one</i> of the child tags is allowed ("choice").
XML path(s)	<cfg:Configuration> → <cfg:RTCConfig> → <cfg:EthSearch>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:EthSearch> <!-- allowed/possible child tags EthSearch in the XML structure overview --> </cfg:EthSearch></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> EthSearch is <i>only</i> a container tag. No value(s), no attribute(s). With the child tag ('{ }' = here: only <i>one</i> of the child tags is allowed ("choice")) of EthSearch the RTC6 Ethernet board(s) is (are) configured.
Version info	syncAXIS_1_8.xsd

XML tag	Broadcast
XML signature (incl. defaults)	Broadcast
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:RTCConfig> → <cfg:EthSearch> → <cfg:Broadcast>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:EthSearch> <cfg:Broadcast> <cfg:IP>169.254.1.0</cfg:IP> <cfg:NetMask>255.255.0.0</cfg:NetMask> </cfg:Broadcast> </cfg:EthSearch> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> Broadcast is <i>only</i> a container tag. No value(s), no attribute(s). The child tags of Broadcast define the subnet in which a broadcast search is to be carried out. For more information on RTC6 Ethernet boards, refer to the RTC6 Manual. <p>IP</p> <ul style="list-style-type: none"> XML signature (incl. defaults): IP = 169.254.1.0 Specifies the network address of the subnet for the broadcast search. Format: IP Address in dotted decimal notation. Settable via API?: Not possible. Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>NetMask</p> <ul style="list-style-type: none"> XML signature (incl. defaults): NetMask = 255.255.0.0 Specifies the network mask of the subnet for the broadcast search. Format: IP Address in dotted decimal notation. Settable via API?: Not possible. Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Version info	syncAXIS_1_8.xsd

XML tag	IPScan
XML signature (incl. defaults)	IPScan
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:RTCConfig> → <cfg:EthSearch> → <cfg:IPScan>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:EthSearch> <cfg:IPScan> <cfg:StartIp>169.254.1.0</cfg:StartIp> <cfg:EndIp>169.254.1.100</cfg:EndIp> </cfg:IPScan> </cfg:EthSearch> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> IPScan is <i>only</i> a container tag. No value(s), no attribute(s). The child tags of IPScan define the IP addresses for which an IP scan is to be carried out. Network packets are sent to these IP addresses. For more information on RTC6 Ethernet boards, refer to the RTC6 Manual. <p>StartIp</p> <ul style="list-style-type: none"> XML signature (incl. defaults): StartIp = 169.254.1.0 Specifies the lower IP range boundary for the IP scan. Format: IP Address in dotted decimal notation. Settable via API?: Not possible. Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>EndIp</p> <ul style="list-style-type: none"> XML signature (incl. defaults): EndIp = 169.254.1.100 Specifies the upper IP range boundary for the IP scan. syncAXIS_1_8.xsd allows up to 4 IPAddress tags. Format: IP Address in dotted decimal notation. Settable via API?: Not possible. Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Version info	syncAXIS_1_8.xsd

XML tag	IPList
XML signature (incl. defaults)	IPList
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:RTCConfig> → <cfg:EthSearch> → <cfg:IPList>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:EthSearch> <cfg:IPList> <cfg:IPAddress>192.168.0.1</cfg:IPAddress> <cfg:IPAddress>192.168.0.2</cfg:IPAddress> <cfg:IPAddress>192.168.0.3</cfg:IPAddress> </cfg:IPList> </cfg:EthSearch> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> IPList is <i>only</i> a container tag. No value(s), no attribute(s). The child tags of IPList define the IP addresses for which an IP scan is to be carried out. Network packets are sent to these IP addresses. For more information on RTC6 Ethernet boards, refer to the RTC6 Manual. <p>IPAddress</p> <ul style="list-style-type: none"> XML signature (incl. defaults): IPAddress[] = 169.254.1.0 Specifies the IP address of an RTC6 Ethernet board. syncAXIS_1_8.xsd allows up to 255 IPAddress tags. Format: IP Adress in dotted decimal notation. Settable via API?: Not possible. Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Version info	syncAXIS_1_8.xsd


XML tag	EthMaxTimeout
XML signature (incl. defaults)	EthMaxTimeout
	<p>'*' = optional; no '*' = mandatory.</p> <p>Unit attribute value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:RTCConfig> → <cfg:EthSearch> → <cfg:EthMaxTimeout>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:EthSearch> <cfg:EthMaxTimeout>2.0</cfg:EthMaxTimeout> </cfg:EthSearch></pre>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Defines the maximal Ethernet timeout. The default value 2.0 means 2 s. A change of the EthMaxTimeout value occurs as a call of the RTC6 command eth_set_com_timeouts_auto(MaxTimeout) with MaxTimeout = (EthMaxTimeout in ms). For more information on RTC6 Ethernet boards, refer to the RTC6 Manual.
Version info	syncAXIS_1_8.xsd

XML tag	ScanDeviceConfig
XML signature (incl. defaults)	ScanDeviceConfig
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:ScanDeviceConfig> <!-- allowed/possible child tags ScanDeviceConfig in the XML structure overview --> </cfg:ScanDeviceConfig> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> ScanDeviceConfig is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd


XML tag	DynamicLimits
XML signature (incl. defaults)	DynamicLimits '**'=optional; no '*'=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:DynamicLimits>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:DynamicLimits> <cfg:Velocity Unit="rad/s">90</cfg:Velocity> <cfg:Acceleration Unit="rad/s^2">1.1314e5</cfg:Acceleration> <cfg:Jerk Unit="rad/s^3">4e9</cfg:Jerk> </cfg:DynamicLimits> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> DynamicLimits is <i>only</i> a container tag. No value(s), no attribute(s). The DynamicLimits child tags Velocity, Acceleration, Jerk serve to specify the maximum dynamic capabilities of the intended scan device type (for example, excelliSCAN 14 with special tuning, excelliSCAN 20 with standard tuning). Special case: syncAXIS Viewer uses the values at Velocity, Acceleration and Jerk to visualize the positioning stage working field and to indicate dynamic limit value exceedances. syncAXIS control uses to monitor working field and dynamics as: <ul style="list-style-type: none"> scan device dynamic limits the DynamicLimits values, page 388 scan device working field limits the FieldLimits values, page 395 scan device monitoring criterion the MonitoringLevel value, page 397 positioning stage dynamic limits the FieldLimits values, page 454 positioning stage working field limits the DynamicLimits values, page 456 positioning stage monitoring criterion the MonitoringLevel values, page 452 reaction on exceedances the DynamicViolationReaction values, page 365 <p>Velocity</p> <ul style="list-style-type: none"> XML signature (incl. defaults): Velocity = 90.0 (Unit*) Settable via API?: slsc_cfg_set_dynamic_limits_scan_device The maximum speed the scan device type is capable of The default value 90 [rad/s] is that of an excelliSCAN 14 with standard tuning. The value for an excelliSCAN 20 with standard tuning is 35 [rad/s] Behavior on Module replay: The Module is rejected, if the replaying syncAXIS control instance is in Operation mode ScannerOnly or ScannerAndStage and the Velocity value is smaller than MarkSpeed value or JumpSpeed value in the Module. In this case, you need to record a new Module with correspondingly different parameter value. See also Section "Behavior on Module replay", page 66.

<p>Comment(s) (cont'd)</p>	<p>Acceleration</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): <code>Acceleration = 1.1314e5 (Unit*)</code> • Settable via API?: <code>slsc_cfg_set_dynamic_limits_scan_device</code> • The maximum acceleration the scan device type is capable of • The default value $1.1314e5 \text{ [rad/s}^2\text{]}$ is that of an excelliSCAN 14 with standard tuning. The value for an excelliSCAN 20 with standard tuning is $56000 \text{ [rad/s}^2\text{]}$ • Behavior on Module replay: The Module is rejected, if the replaying syncAXIS control instance is in Operation mode <code>ScannerOnly</code> or <code>ScannerAndStage</code> and the Acceleration value is smaller than in the Module. In this case, you need to record a new Module with correspondingly different parameter value. See also Section "Behavior on Module replay", page 66. <p>Jerk</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): <code>Jerk = 4e9 (Unit*)</code> • Settable via API?: <code>slsc_cfg_set_dynamic_limits_scan_device</code> • The maximum jerk the scan device type is capable of • The default value $4e9 \text{ [rad/s}^3\text{]}$ is that of an excelliSCAN 14 with standard tuning. The value for an excelliSCAN 20 with standard tuning is $10e9 \text{ [rad/s}^3\text{]}$ • Behavior on Module replay: The Module is rejected, if the replaying syncAXIS control instance is in Operation mode <code>ScannerOnly</code> or <code>ScannerAndStage</code> and the Jerk value is smaller than in the Module. In this case, you need to record a new Module with correspondingly different parameter value. See also Section "Behavior on Module replay", page 66.
<p>Version info</p>	<p>syncAXIS_1_8.xsd</p>

XML tag	CalculationDynamics
XML signature (incl. defaults)	CalculationDynamics
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:CalculationDynamics>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:CalculationDynamics> <!-- allowed/possible child tags CalculationDynamics in the XML structure overview --> </cfg:CalculationDynamics> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> CalculationDynamics is <i>only</i> a container tag. No value(s), no attribute(s). The child tags of CalculationDynamics are used to configure which acceleration & jerk maximum values are used in Trajectory planning calculations for the scan devices. Therefore, these are "planning upper limits" but not planned accelerations or planned jerks. The values are specified in separate child tags: <ul style="list-style-type: none"> MarkDynamics for markings JumpDynamics for jumps <p>Note that the corresponding speed maximum values are configured under:</p> <ul style="list-style-type: none"> <cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:MarkConfig> → <cfg:JumpSpeed ...> <cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:MarkConfig> → <cfg:MarkSpeed ...>
Version info	syncAXIS_1_8.xsd

XML tag	MarkDynamics
XML signature (incl. defaults)	MarkDynamics ''=optional; no ''=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:CalculationDynamics> → <cfg:MarkDynamics>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:MarkDynamics> <cfg:Acceleration>1.1314e5</cfg:Acceleration> <cfg:Jerk>4e9</cfg:Jerk> </cfg:MarkDynamics></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> MarkDynamics is <i>only</i> a container tag. No value(s), no attribute(s). The child tags of MarkDynamics refer to values for markings, see page 390. For the values for jumps there is JumpDynamics. <p>Acceleration</p> <ul style="list-style-type: none"> XML signature (incl. defaults): Acceleration = 1.1314e5 (Unit*) Specifies the maximum acceleration value used in Trajectory planning calculations for the scan devices. Allowed entries: <ul style="list-style-type: none"> ≥ 0.00001 Unit: rad/s². Format: double.  Caution! syncAXIS control uses the Acceleration value = MarkAngularAcc = MarkAngularAcc to plan trajectories for the Operation modes "ScannerOnly" and "ScannerAndStage". Make sure that the entered values are correct. Settable via API?: <pre>slsc_cfg_set_calculation_dynamics_mark_scan_device(MarkAngularAcc) slsc_list_set_calculation_dynamics_mark_scan_device(MarkAngularAcc)</pre> Behavior on Module replay: Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.

Comment(s) (cont'd)	<p>Jerk</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): <code>Jerk = 4e9 (Unit*)</code> • Specifies the maximum jerk value used in Trajectory planning calculations for the scan devices. • Allowed entries: <ul style="list-style-type: none"> – ≥ 0.00001 Unit: rad/s^3. Format: <code>double</code>. • ⚠ Caution! syncAXIS control uses the <code>Jerk</code> value = <code>MarkAngularJerk</code> = <code>MarkAngularJerk</code> to plan trajectories for the Operation modes "<code>ScannerOnly</code>" and "<code>ScannerAndStage</code>". Make sure that the entered values are correct. • Settable via API?: <pre>slsc_cfg_set_calculation_dynamics_mark_scan_device(MarkAngularJerk) slsc_list_set_calculation_dynamics_mark_scan_device(MarkAngularJerk)</pre> • Behavior on Module replay: Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.
Version info	syncAXIS_1_8.xsd

XML tag	JumpDynamics
XML signature (incl. defaults)	JumpDynamics ''=optional; no ''=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:CalculationDynamics> → <cfg:JumpDynamics>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:JumpDynamics> <cfg:Acceleration>1.1314e5</cfg:Acceleration> <cfg:Jerk>4e9</cfg:Jerk> </cfg:JumpDynamics></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> JumpDynamics is <i>only</i> a container tag. No value(s), no attribute(s). The child tags of JumpDynamics refer to values for jumps, see page 390. For the values for markers there is MarkDynamics. <p>Acceleration</p> <ul style="list-style-type: none"> XML signature (incl. defaults): Acceleration = 1.1314e5 (Unit*) Specifies the maximum acceleration value used in Trajectory planning calculations for the scan devices. Allowed entries: <ul style="list-style-type: none"> ≥ 0.00001 Unit: rad/s². Format: double.  Caution! syncAXIS control uses the Acceleration value = JumpAngularAcc = JumpAngularAcc to plan trajectories for the Operation modes "ScannerOnly" and "ScannerAndStage". Make sure that the entered values are correct. Settable via API?: <pre>slsc_cfg_set_calculation_dynamics_jump_scan_device(JumpAngularAcc) slsc_list_set_calculation_dynamics_jump_scan_device(JumpAngularAcc)</pre> Behavior on Module replay: Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.

Comment(s) (cont'd)	<p>Jerk</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): <code>Jerk = 4e9 (Unit*)</code> • Specifies the maximum jerk value used in Trajectory planning calculations for the scan devices. • Allowed entries: <ul style="list-style-type: none"> – ≥ 0.00001 Unit: rad/s^3. Format: <code>double</code>. • ⚠ Caution! syncAXIS control uses the <code>Jerk</code> value = <code>JumpAngularJerk</code> = <code>JumpAngularJerk</code> to plan trajectories for the Operation modes "<code>ScannerOnly</code>" and "<code>ScannerAndStage</code>". Make sure that the entered values are correct. • Settable via API?: <ul style="list-style-type: none"> <code>slsc_cfg_set_calculation_dynamics_jump_scan_device(JumpAngularJerk)</code> <code>slsc_list_set_calculation_dynamics_jump_scan_device(JumpAngularJerk)</code> • Behavior on Module replay: Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.
Version info	syncAXIS_1_8.xsd

XML tag	FieldLimits
XML signature (incl. defaults)	FieldLimits ''=optional; no ''=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:FieldLimits>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:FieldLimits> <cfg:XDirection Unit="mm" Max="27" Min="-27" /> <cfg:YDirection Unit="mm" Max="27" Min="-27" /> </cfg:FieldLimits></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> FieldLimits is <i>only</i> a container tag. No value(s), no attribute(s). The FieldLimits child tags XDirection and YDirection serve to specify the working field limits of the intended scan device type. syncAXIS control uses to monitor working field and dynamics as: <ul style="list-style-type: none"> scan device dynamic limits the DynamicLimits values, page 388 scan device working field limits the FieldLimits values, page 395 scan device monitoring criterion the MonitoringLevel value, page 397 positioning stage dynamic limits the FieldLimits values, page 454 positioning stage working field limits the DynamicLimits values, page 456 positioning stage monitoring criterion the MonitoringLevel values, page 452 reaction on exceedances the DynamicViolationReaction values, page 365 syncAXIS control does <i>not</i> use the values at XDirection and YDirection to plan trajectories! syncAXIS Viewer uses the values for values at XDirection and YDirection to visualize the scan device working field and to indicate limit value exceedances. The entered XDirection and YDirection do not need to correspond the usable working field which is defined by the correction file (*.ct5). Lower field boundaries can also be set, if a smaller part of the working field is to be monitored. Higher field boundaries should not be set.

<p>Comment(s) (cont'd)</p>	<p>XDirection</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): XDirection (Unit*, Max, Min) • Max, Min: double. • Settable via API?: slsc_cfg_set_field_limits_scan_device • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>YDirection</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): YDirection (Unit*, Max, Min) • Max, Min: double. • Settable via API?: slsc_cfg_set_field_limits_scan_device • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>ZDirection</p> <ul style="list-style-type: none"> • <i>This parameter is currently reserved!</i> • XML signature (incl. defaults): ZDirection* (Unit*, Max, Min) • Max, Min: double. • Settable via API?: slsc_cfg_set_field_limits_scan_device • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
<p>Version info</p>	<p>syncAXIS_1_8.xsd</p>

XML tag	MonitoringLevel
XML signature (incl. defaults)	MonitoringLevel = Position
	<p>'**'=optional; no '**'=mandatory.</p> <p>MonitoringLevel value: string.</p>
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:MonitoringLevel>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:MonitoringLevel> Acceleration </cfg:MonitoringLevel>
Settable via API?	slsc_cfg_set_scan_device_dynamic_monitoring_level
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Defines a criterion for which the scan devices are to be monitored. The criterion does <i>not</i> apply to the positioning stages – the dedicated tag MonitoringLevel, see page 452, exists for these. Exceedances automatically trigger the reaction defined in DynamicViolationReaction. Allowed entries: <ul style="list-style-type: none"> Deactivated Position Velocity Acceleration Jerk syncAXIS control uses to monitor working field and dynamics as: <ul style="list-style-type: none"> scan device dynamic limits the DynamicLimits values, page 388 scan device working field limits the FieldLimits values, page 395 scan device monitoring criterion the MonitoringLevel value, page 397 positioning stage dynamic limits the FieldLimits values, page 454 positioning stage working field limits the DynamicLimits values, page 456 positioning stage monitoring criterion the MonitoringLevel values, page 452 reaction on exceedances the DynamicViolationReaction values, page 365
Version info	syncAXIS_1_8.xsd

XML tag	FocalLength
XML signature (incl. defaults)	FocalLength* = 100 (Unit*)
	'*' = optional; no '*' = mandatory. Unit attribute value: double.
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:FocalLength ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:FocalLength Unit="mm">100</cfg:FocalLength>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> The focal length of the scan system. Depends on the optics used. ⚠ Caution! syncAXIS control uses these values to plan trajectories for the Operation modes "ScannerOnly" and "ScannerAndStage". Make sure that the entered values are correct.
Version info	syncAXIS_1_8.xsd

XML tag	Delay
XML signature (incl. defaults)	Delay* = 0.00125 (Unit*) '*'=optional; no '*'=mandatory. Unit attribute value: double.
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:Delay ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:Delay Unit="s">0.00125</cfg:Delay>
Settable via API?	Not possible.
Behavior on Module replay	The parameter value of the replaying syncAXIS control instance is applied. If the calculated (from <cfg:Configuration> → <cfg:StageConfig> → <cfg:CTIME ...> and <cfg:Configuration> → <cfg:StageConfig> → <cfg:DelayShift ...>) positioning stage delay differs from the one in the Module , then [WARN] log file lines are generated. "SIGNAL" functions included in the Module that have a <i>negative</i> delay are not executed, if the absolute value of the delay is greater than either largest scan device delay (<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:Delay ...>) or calculated positioning stage delay. See also Section "Behavior on Module replay" , page 66.
Comment(s)	<ul style="list-style-type: none"> • Delay of the scan system. • excelliSCAN with standard tuning used for syncAXIS control have a typical delay of 1.25 ms. • Notice! The entries in a simulation file are shifted (among others) by this delay. This is taken into account in the syncAXIS Viewer. For a "simpler" simulation the scan device delay value (here at <code>Delay</code>) and the positioning stage delay value (at <code>CTIME</code>) can be set each to 0.
Version info	syncAXIS_1_8.xsd

XML tag	ScanDeviceList
XML signature (incl. defaults)	ScanDeviceList
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:ScanDeviceList>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:ScanDeviceList> <!-- allowed/possible child tags ScanDeviceList in the XML structure overview --> </cfg:ScanDeviceList> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> ScanDeviceList is <i>only</i> a container tag. No value(s), no attribute(s). With the child tags of ScanDeviceList, all to-be-used scan devices are configured.
Version info	syncAXIS_1_8.xsd

XML tag	ScanDevice
XML signature (incl. defaults)	<p>ScanDevice[] (Name)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>Name attribute value: string.</p>
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:ScanDeviceList> → <cfg:ScanDevice ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:ScanDevice Name="ScanDevice1"> <!-- allowed/possible child tags see ScanDevice in the XML structure overview --> </cfg:ScanDevice></pre>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> ScanDevice is <i>also</i> a container tag. No value(s), 1 attribute: Name. With syncAXIS_1_8.xsd allowed ScanDevice tags: <ul style="list-style-type: none"> – up to 4 Allowed Name attribute value, see enum slsc_ScanDevice: <pre>"ScanDevice1", "ScanDevice2", "ScanDevice3", "ScanDevice4".</pre>
Version info	syncAXIS_1_8.xsd

XML tag	CorrectionFileList
XML signature (incl. defaults)	CorrectionFileList
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:ScanDeviceList> → <cfg:ScanDevice ...> → <cfg:CorrectionFileList>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:CorrectionFileList> <!-- allowed/possible child tags see CorrectionFileList in the XML structure overview --> </cfg:CorrectionFileList></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> CorrectionFileList is <i>only</i> a container tag. No value(s), no attribute(s). With the child tags of CorrectionFileList, all to-be-used correction files are configured.
Version info	syncAXIS_1_8.xsd

XML tag	CorrectionFilePath
XML signature (incl. defaults)	<p>CorrectionFilePath[] = "" (CalibrationFactor* = -1)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>CalibrationFactor value: double.</p> <p>-1: The calibration factor is read from the correction file.</p> <p>Folder path: string.</p>
XML path(s)	<p><cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:ScanDeviceList> → <cfg:ScanDevice ...> → <cfg:CorrectionFileList> → <cfg:CorrectionFilePath ...></p>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358 .
XML section example	<pre><cfg:CorrectionFilePath CalibrationFactor="-1">C:\folderpath </cfg:CorrectionFilePath></pre>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> • syncAXIS_1_8.xsd allows up to 4 CorrectionFilePath tags. • CorrectionFilePath creates an entry in the syncAXIS-DLL-internal list of correction files. The first two correction files are written onto the RTC6 memory and then, can be quickly selected by slsc_ctrl_select_correction_file. • If the CalibrationFactor value is empty or invalid, a 1to1 correction file is loaded. • For more information on correction files, refer to the RTC6 Manual.
Version info	syncAXIS_1_8.xsd

XML tag	Alignment
XML signature (incl. defaults)	Alignment ''=optional; no ''=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:ScanDeviceList> → <cfg:ScanDevice ...> → <cfg:Alignment>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:Alignment> <cfg:Matrix> <cfg:T11>1</cfg:T11> <cfg:T12>0</cfg:T12> <cfg:T21>0</cfg:T21> <cfg:T22>1</cfg:T22> </cfg:Matrix> <cfg:Offset X="0" Y="0" /> </cfg:Alignment> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> Alignment is <i>only</i> a container tag. No value(s), no attribute(s). The Alignment-child tags Matrix and Offset are available to compensate for assembly errors in Multi-Head systems. The positions sent to a respective scan device have been transformed in a way that scan device coordinate system and reference system (for example, the positioning stage coordinate system) do match. Alignment is <i>not</i> applied to the Trajectory planning. Alignment occurs only on the RTC6. A simulation file does not contain alignment transformation (simulation files show the marking not taking into account any miscalibration). The matrix must be invertible, but does not need to be normed. For further information, see Chapter 8.3 "About Transformations in syncAXIS control V1.2.4 and Higher", page 332. <p>Matrix</p> <ul style="list-style-type: none"> XML signature (incl. defaults): Matrix Is container tag (only). No value(s), no attribute(s). $\text{Matrix} = \begin{bmatrix} T11 & T12 \\ T21 & T22 \end{bmatrix}$ <ul style="list-style-type: none"> Settable via API?: Not possible. Behavior on Module replay: No behavior as it is a container tag

<p>Comment(s) (cont'd)</p>	<p>T11</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): T11 = 1 • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>T12</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): T12 = 0 • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>T21</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): T21 = 0 • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>T22</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): T22 = 1 • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>Offset</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): Offset (X = 0, Y = 0, Unit*) • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
<p>Version info</p>	<p>syncAXIS_1_8.xsd</p>

XML tag	BasePartDisplacement
XML signature (incl. defaults)	BasePartDisplacement ''=optional; no ''=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:ScanDeviceList> → <cfg:ScanDevice ...> → <cfg:BasePartDisplacement>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:BasePartDisplacement> <cfg:Matrix> <cfg:T11>1</cfg:T11> <cfg:T12>0</cfg:T12> <cfg:T21>0</cfg:T21> <cfg:T22>1</cfg:T22> </cfg:Matrix> <cfg:Offset X="0" Y="0" /> </cfg:BasePartDisplacement> </pre>
Settable via API?	slsc_cfg_set_part_displacement
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> BasePartDisplacement is <i>only</i> a container tag. No value(s), no attribute(s). For further information, see Chapter 8.3 "About Transformations in syncAXIS control V1.2.4 and Higher", page 332. It is possible to compensate a misalignment of every workpiece located under this particular ScanDevice. The positioning stage only executes a certain part (defined by the Motion decomposition) of the overall contour. For this reason, such compensation must be carried out completely by the scan device, which makes only minor corrections possible. The BasePartDisplacement must be a previously known misalignment of each workpiece with a particular scan system. slsc_cfg_set_part_displacement extends the base transformation (= all BasePartDisplacement child tags) by multiplying the matrices and adding the offsets of these two transformations. <p>Matrix</p> <ul style="list-style-type: none"> XML signature (incl. defaults): Matrix No value(s), no attribute(s). $\text{Matrix} = \begin{bmatrix} T11 & T12 \\ T21 & T22 \end{bmatrix}$ <ul style="list-style-type: none"> Settable via API?: Not possible. Behavior on Module replay: No behavior as it is a container tag

<p>Comment(s) (cont'd)</p>	<p>T11</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): T11 = 1 • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>T12</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): T12 = 0 • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>T21</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): T21 = 0 • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>T22</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): T22 = 1 • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>Offset</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): Offset (X = 0, Y = 0, Unit*) • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
<p>Version info</p>	<p>syncAXIS_1_8.xsd</p>

XML tag	DefaultCorrectionFile
XML signature (incl. defaults)	DefaultCorrectionFile* = 0
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:DefaultCorrectionFile>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:DefaultCorrectionFile>0</cfg:DefaultCorrectionFile>
Settable via API?	slsc_ctrl_select_correction_file
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Determines which correction file from the syncAXIS-DLL-internal list of correction files is going to be used within this syncAXIS control instance. Allowed entries: 0...3. Format: non-negative integer. The correction file in use can be switched by slsc_ctrl_select_correction_file. slsc_ctrl_refresh_correction_file reloads the correction file in use from the specified path without changing its correction file index.
Version info	syncAXIS_1_8.xsd

XML tag	LaserConfig
XML signature (incl. defaults)	LaserConfig
	'*' = optional; no '**' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:LaserConfig> <!-- allowed/possible child tags see LaserConfig in the XML structure overview --> </cfg:LaserConfig></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> LaserConfig is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	LaserMode
XML signature (incl. defaults)	LaserMode* = 0
	'*' = optional; no '**' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:LaserMode>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:LaserMode>5</cfg:LaserMode>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Defines the RTC6 laser mode of the syncAXIS control instance. Allowed entries: 0, 4...6. Format: non-negative integer. For more information on laser modes, refer to the RTC6 Manual.
Version info	syncAXIS_1_8.xsd

XML tag	LaserPortCfg
XML signature (incl. defaults)	LaserPortCfg*
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:LaserPortCfg>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:LaserPortCfg> <cfg:LaserOn>LaserOnSignal</cfg:LaserOn> <cfg:Laser1>Laser1Signal</cfg:Laser1> <cfg:Laser2>Laser2Signal</cfg:Laser2> </cfg:LaserPortCfg> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> LaserPortCfg is <i>only</i> a container tag. No value(s), no attribute(s). The child tags of LaserPortCfg define the output ports of the various laser control signals. By default, each laser control signal is mapped to the port of its name. In some cases, they might need to be mapped differently. For more information on the laser control signals and output ports, refer to the RTC6 Manual. LaserOn <ul style="list-style-type: none"> XML signature (incl. defaults): LaserOn = LaserOnSignal Specifies the laser signal for output channel LASERON. Allowed entries: <ul style="list-style-type: none"> LaserOnSignal Laser1Signal Laser2Signal FirstPulseKillerSignal Format: string. Settable via API?: Not possible. Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.

<p>Comment(s) (cont'd)</p>	<p>Laser1</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): Laser1 = Laser1Signal • Specifies the laser signal for output channel LASER1. • Allowed entries: <ul style="list-style-type: none"> – LaserOnSignal – Laser1Signal – Laser2Signal – FirstPulseKillerSignal • Format: string. • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>Laser2</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): Laser2 = Laser2Signal • Specifies the laser signal for output channel LASER2. • Allowed entries: <ul style="list-style-type: none"> – LaserOnSignal – Laser1Signal – Laser2Signal – FirstPulseKillerSignal • Format: string. • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
<p>Version info</p>	<p>syncAXIS_1_8.xsd</p>

XML tag	LaserOutput
XML signature (incl. defaults)	<p>LaserOutput* (Unit*, HalfPeriod, PulseLength)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>Unit attribute value: double.</p> <p>HalfPeriod attribute value: double.</p> <p>PulseLength attribute value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:LaserOutput ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:LaserOutput Unit="s" HalfPeriod="5e-6" PulseLength="1e-6" />
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> LaserOutput defines (dependent on the LaserMode) the pulse train of the laser control output during (the laser control signal) LASERON. The HalfPeriod attribute value is half the time between the rising edges of two following laser control signal pulses. The PulseLength attribute value is their length. The attribute values are ignored, if the corresponding channel (PulseLength, HalfPeriod, SpotDistance) is entered as an "ActiveChannel" for the "Automatic Laser Control" (see ActiveChannel). These values are outputted again in Mode "Manual Positioning" and after slsc_list_suppress_spotdistance_control. For more information on laser control, refer to the RTC6 Manual.
Version info	syncAXIS_1_8.xsd

XML tag	LaserStandby
XML signature (incl. defaults)	<p>LaserStandby* (Unit*, HalfPeriod, PulseLength)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>HalfPeriod attribute value: double.</p> <p>PulseLength attribute value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:LaserStandby ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358 .
XML section example	<cfg:LaserStandby Unit="s" HalfPeriod="0.00" PulseLength="0.0" />
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> LaserStandby defines (dependent on the LaserMode) the pulse train of the laser control output during the laser is off. The HalfPeriod attribute value is half the time between the rising edges of two following laser control signal pulses. The PulseLength attribute value is their length. For more information on laser control, refer to the RTC6 Manual.
Version info	syncAXIS_1_8.xsd

XML tag	QSwitchDelay
XML signature (incl. defaults)	<p>QSwitchDelay* = 0 (Unit*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>Unit attribute value: double.</p> <p>QSwitchDelay value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:QSwitchDelay ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:QSwitchDelay Unit="s">0.0</cfg:QSwitchDelay>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Defines the delay of the Q-Switch signal. For more information on the Q-Switch delay, refer to the RTC6 Manual.
Version info	syncAXIS_1_8.xsd

XML tag	FPulseKillerLength
XML signature (incl. defaults)	FPulseKillerLength* = 0 (Unit*) '*'=optional; no '*'=mandatory. Unit attribute value: double. FPulseKillerLength value: double.
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:FPulseKillerLength ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:FPulseKillerLength Unit="s">0.0</cfg:FPulseKillerLength>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Defines the length of the FirstPulseKiller signal. For more information on the FirstPulseKiller signal, refer to the RTC6 Manual.
Version info	syncAXIS_1_8.xsd

XML tag	LaserControlFlags
XML signature (incl. defaults)	LaserControlFlags* '*'=optional; no '*'=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:LaserControlFlags>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre><cfg:LaserControlFlags> <!-- allowed/possible child tags see LaserControlFlags in the XML structure overview --> </cfg:LaserControlFlags></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> LaserControlFlags is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	LaserDisable
XML signature (incl. defaults)	LaserDisable* = false
	<p>'*' = optional; no '*' = mandatory.</p> <p>LaserDisable value: boolean.</p>
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:LaserControlFlags> → <cfg:LaserDisable>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:LaserDisable>false</cfg:LaserDisable>
Settable via API?	<p>To set to false: slsc_ctrl_enable_laser.</p> <p>To set to true: slsc_ctrl_disable_laser.</p>
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Defines whether the laser control signals are active or deactivated. For more information on enabling and disabling the laser control signals, refer to the RTC6 Manual (set_laser_control, Bit #2).
Version info	syncAXIS_1_8.xsd

XML tag	PulseSwitchSetting
XML signature (incl. defaults)	PulseSwitchSetting* = false
	'**'=optional; no '**'=mandatory. PulseSwitchSetting value: boolean.
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:LaserControlFlags> → <cfg:PulseSwitchSetting>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:PulseSwitchSetting>false</cfg:PulseSwitchSetting>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Defines whether LASER1 signal and LASER2 signal are cut off after the LASERON phase (that is, at the falling edge of the laser control signal LASERON). For more information on the Pulse Switch Setting, refer to the RTC6 Manual (set_laser_control, Bit #0).
Version info	syncAXIS_1_8.xsd

XML tag	LaserSignalPhaseShift
XML signature (incl. defaults)	LaserSignalPhaseShift* = false
	<p>'*' = optional; no '*' = mandatory.</p> <p>LaserSignalPhaseShift value: boolean.</p>
XML path(s)	<code><cfg:Configuration></code> → <code><cfg:LaserConfig></code> → <code><cfg:LaserControlFlags></code> → <code><cfg:LaserSignalPhaseShift></code>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<code><cfg:LaserSignalPhaseShift>false</cfg:LaserSignalPhaseShift></code>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Defines whether the LASER1 and LASER2 signals are phase shifted (that is, both signals are switched in CO2 mode (LaserMode = 0) and the LASER1 signal is shifted backwards by 180° in YAG modes (LaserMode = 1,2,3 and 5)). For more information on the phase shift of the laser control signals, refer to the RTC6 Manual (set_laser_control, Bit #1).
Version info	syncAXIS_1_8.xsd

XML tag	LaserOnSignalActiveLow
XML signature (incl. defaults)	LaserOnSignalActiveLow* = false
	<p>'*' = optional; no '*' = mandatory.</p> <p>LaserOnSignalActiveLow value: boolean.</p>
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:LaserControlFlags> → <cfg:LaserOnSignalActiveLow>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:LaserOnSignalActiveLow>false</cfg:LaserOnSignalActiveLow>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> true: LASERON, LASER1 and LASER2 are low active. false: LASERON, LASER1 and LASER2 are high active. For more information on the signal level of the laser control signal LASERON, refer to the RTC6 Manual (set_laser_control, Bit #3).
Version info	syncAXIS_1_8.xsd

XML tag	Laser1Laser2SignalActiveLow
XML signature (incl. defaults)	Laser1Laser2SignalActiveLow* = false
	'**'=optional; no '**'=mandatory. Laser1Laser2SignalActiveLow value: boolean.
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:LaserControlFlags> → <cfg:Laser1Laser2SignalActiveLow>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:Laser1Laser2SignalActiveLow>false</cfg:Laser1Laser2SignalActiveLow>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> true: LASERON, LASER1 and LASER2 are low active. false: LASERON, LASER1 and LASER2 are high active. For more information on the LASER1/LASER2 signal level, refer to the RTC6 Manual (set_laser_control, Bit #4).
Version info	syncAXIS_1_8.xsd

XML tag	LaserPulsesAtRisingEdge
XML signature (incl. defaults)	LaserPulsesAtRisingEdge* = false
	'**'=optional; no '**'=mandatory. LaserPulsesAtRisingEdge value: boolean.
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:LaserControlFlags> → <cfg:LaserPulsesAtRisingEdge>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:LaserPulsesAtRisingEdge>false</cfg:LaserPulsesAtRisingEdge>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> LaserPulsesAtRisingEdge is currently reserved.
Version info	syncAXIS_1_8.xsd

XML tag	OutputSynchronizationOn
XML signature (incl. defaults)	OutputSynchronizationOn* = false
	'*' = optional; no '*' = mandatory. OutputSynchronizationOn value: boolean.
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:LaserControlFlags> → <cfg:OutputSynchronizationOn>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:OutputSynchronizationOn>false</cfg:OutputSynchronizationOn>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> OutputSynchronizationOn is currently reserved.
Version info	syncAXIS_1_8.xsd

XML tag	AutomaticLaserControl
XML signature (incl. defaults)	AutomaticLaserControl
	'*' = optional; kein '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:AutomaticLaserControl> <!-- allowed/possible child tags see AutomaticLaserControl in the XML structure overview --> </cfg:AutomaticLaserControl></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> AutomaticLaserControl is <i>only</i> a container tag. No value(s), no attribute(s). See also Chapter 2.9 "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48.
Version info	syncAXIS_1_8.xsd

XML tag	ActiveChannel
XML signature (incl. defaults)	ActiveChannel ''=optional; no ''=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:ActiveChannel>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:ActiveChannel> <cfg:Channel>AnalogOut1</cfg:Channel> <cfg:Channel>SpotDistance</cfg:Channel> </cfg:ActiveChannel> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> ActiveChannel is <i>only</i> a container tag. No value(s), no attribute(s). For the "Automatic Laser Control", Shift, VelocityFactor and RadiusFactor can be specified that can be set as disabled or enabled separately. Note that Ramps are only applied to "ActiveChannel". See also Chapter 2.9 "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48. <p>Channel</p> <ul style="list-style-type: none"> XML signature (incl. defaults): Channel[]* syncAXIS_1_8.xsd allows up to 2 Channel tags. Defines which control parameter is used for the "Automatic Laser Control". Allowed entries: <ul style="list-style-type: none"> AnalogOut1 AnalogOut2 PulseLength HalfPeriod SpotDistance Format: string. Settable via API?: Not possible. Behavior on Module replay: The parameter value of the replaying syncAXIS control instance is applied. If it differs from the one in the Module, then [WARN] log file lines are generated. See also Section "Behavior on Module replay", page 66.
Version info	syncAXIS_1_8.xsd

XML tag	AnalogOut1
XML signature (incl. defaults)	<p>AnalogOut1* (DefaultOutput, Format*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>DefaultOutput attribute value: double.</p> <p>Format attribute value: string.</p>
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:AnalogOut1 ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre><cfg:AnalogOut1 DefaultOutput="0.5" Format="Factor"> <!-- allowed/possible child tags see AnalogOut1 in the XML structure overview --> </cfg:AnalogOut1></pre>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> AnalogOut1 is <i>also</i> a container tag. No value(s), 2 attributes: DefaultOutput, Format. Note: If "AnalogOut1" is not set as (one of the both) "ActiveChannel" (see ActiveChannel), then AnalogOut1 (child tag of DefaultOutputs) is outputted. To define, how AnalogOut1 should be adjusted based on radius and velocity: for this purpose, with the child tags below AnalogOut1 characteristics for the RadiusFactor and VelocityFactor can be defined. DefaultOutput attribute: Default output which is multiplied by factors, see Figure 17, page 52. Format attribute: factor (fraction of the maximum analog voltage of 10 V). See Chapter 2.9 "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48.
Version info	syncAXIS_1_8.xsd

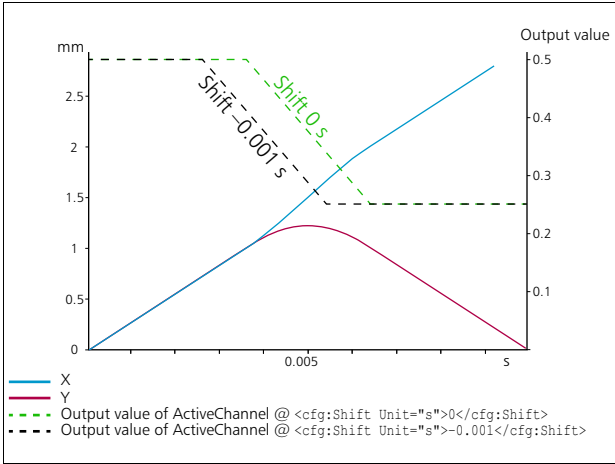
XML tag	AnalogOut2
XML signature (incl. defaults)	<p>AnalogOut2* (DefaultOutput, Format*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>DefaultOutput attribute value: double.</p> <p>Format attribute value: string.</p>
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:AnalogOut2 ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre><cfg:AnalogOut2 DefaultOutput="0.5" Format="Factor"> <!-- allowed/possible child tags see AnalogOut2 in the XML structure overview --> </cfg:AnalogOut2></pre>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> AnalogOut2 is <i>also</i> a container tag. No value(s), 2 attributes: DefaultOutput, Format. Note: If "AnalogOut2" is not set as (one of the both) "ActiveChannel" (see ActiveChannel), then AnalogOut2 (child tag of DefaultOutputs) is outputted. To define, how AnalogOut2 should be adjusted based on radius and velocity: for this purpose, with the child tags below AnalogOut2 characteristics for the RadiusFactor and VelocityFactor can be defined. DefaultOutput attribute: Default output which is multiplied by factors, see Figure 17, page 52. Format attribute: factor (fraction of the maximum analog voltage of 10 V). See Chapter 2.9 "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48.
Version info	syncAXIS_1_8.xsd

XML tag	PulseLength
XML signature (incl. defaults)	PulseLength* (DefaultOutput, Unit*) '*'=optional; no '*'=mandatory. DefaultOutput attribute value: double. Unit attribute value: double.
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:PulseLength ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre> <cfg:PulseLength DefaultOutput="1e-5" Unit="s"> <!-- allowed/possible child tags see PulseLength in the XML structure overview --> </cfg:PulseLength> </pre>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> PulseLength is also a container tag. No value(s), 2 attributes: DefaultOutput, Unit. Note: If "PulseLength" is not set as (one of the both) "ActiveChannel" (see ActiveChannel), then the PulseLength attribute value of LaserOutput is outputted. To define, how the pulse length should be adjusted based on radius and velocity: for this purpose, with the child tags below PulseLength characteristics for the RadiusFactor and VelocityFactor can be defined. DefaultOutput attribute: Default output which is multiplied by factors, see Figure 17, page 52. See Chapter 2.9 "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48.
Version info	syncAXIS_1_8.xsd

XML tag	HalfPeriod
XML signature (incl. defaults)	<p>HalfPeriod* (DefaultOutput, Unit*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>DefaultOutput attribute value: double.</p> <p>Unit attribute value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:HalfPeriod ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre><cfg:HalfPeriod DefaultOutput="1e-5" Unit="s"> <!-- allowed/possible child tags see HalfPeriod in the XML structure overview --> </cfg:HalfPeriod></pre>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> HalfPeriod is <i>also</i> a container tag. No value(s), 2 attributes: DefaultOutput, Unit. Note: If "HalfPeriod" is not set as (one of the both) "ActiveChannel" (see ActiveChannel), then the PulseLength attribute value of LaserOutput is outputted. To define, how the HalfPeriod should be adjusted based on radius and velocity: for this purpose, with the child tags below HalfPeriod characteristics for the RadiusFactor and VelocityFactor can be defined. Note : Scaling the HalfPeriod based on the marking speed can lead to gaps in the marking result, if this speed is close to 0. It is recommended to use SpotDistance instead of HalfPeriod! DefaultOutput attribute: Default output which is multiplied by factors, see Figure 17, page 52. See Chapter 2.9 "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48.
Version info	syncAXIS_1_8.xsd

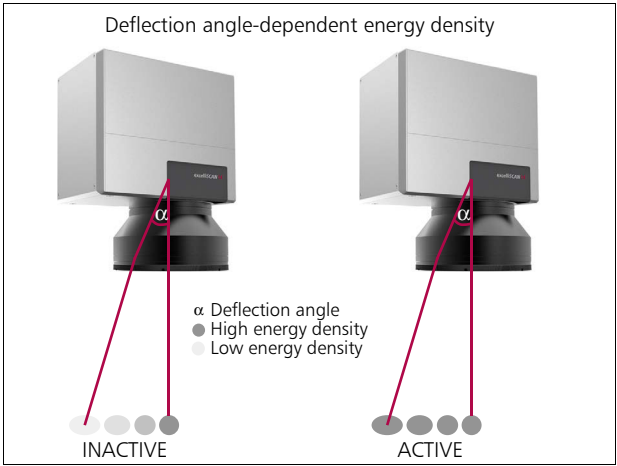
XML tag	SpotDistance
XML signature (incl. defaults)	<p>SpotDistance* (DefaultOutput, Unit*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>DefaultOutput attribute value: double.</p> <p>Unit attribute value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:SpotDistance ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre><cfg:SpotDistance DefaultOutput="0.005" Unit="mm"> <!-- allowed/possible child tags see SpotDistance in the XML structure overview --> </cfg:SpotDistance></pre>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> SpotDistance is <i>also</i> a container tag. No value(s), 2 attributes: DefaultOutput, Unit. With the child tags below SpotDistance, characteristics for the RadiusFactor and VelocityFactor are defined. The DefaultOutput attribute value is the laser spot spacing if no RadiusFactor or VelocityFactor is applied. The DefaultOutput value is rounded internally to a 1/40 RTC6 position bit, that is, $(1/40) / K_{xy}$ [mm]. This means that for long lines a small rounding error can accumulate during positioning. The max. error is $1/40 \times (\text{vector length} \times \text{working field size}) / (2^{20} \times \text{SpotDistance})$. With a working field edge length of 50 mm, a pulse distance of 50 μm and a line length of 10 mm, a rounding error of max. 0.24 μm would result. If the default output value is a multiple of 1/40 RTC6 position bit, there will be no rounding error. syncAXIS control makes use of the RTC6-"Spot Distance Control" feature. This allows a precise positioning of laser pulses with 64 MHz resolution depending on the scanner position (formerly, only HalfPeriod was possible on the RTC). RadiusFactor is not valid for the spot distance itself, but for the spot speed and thus for the spot distance, which is scaled by reversing the factor (1/Factor).

Comment(s) (cont'd)	<ul style="list-style-type: none"> SpotDistance is a special case, in which the velocity information is used although the VelocityFactor might be deactivated to calculate the trigger timing for constant spot distances. But VelocityFactor can still be used for fine tuning. For more information on the SpotDistance Channel, see Chapter 2.9.5 "About the "Contour-dependent speed calculation"", page 60. See Chapter 2.9 "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48.
Version info	syncAXIS_1_8.xsd

XML tag	Shift
XML signature (incl. defaults)	Shift* = 0.0 (Unit*) ''=optional; no ''=mandatory. Unit attribute value: double.
XML path(s)	(1) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:AnalogOut1> → <cfg:Shift> (2) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:AnalogOut2> → <cfg:Shift> (3) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:PulseLength> → <cfg:Shift> (4) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:HalfPeriod> → <cfg:Shift> (5) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:SpotDistance> → <cfg:Shift>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:Shift Unit="s">-1e-5</cfg:Shift>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> By Shift the output values of the "ActiveChannel" (see also Figure 17, page 52, Output value) can be shifted in time. Shift value with negative sign: output is earlier than with 0. The following figure is Figure 21, page 56 plus the additional "Shift -0.001 s" curve (dotted, black).  <p>mm</p> <p>Output value</p> <p>s</p> <p>— X</p> <p>— Y</p> <p>--- Shift 0 s</p> <p>... Shift -0.001 s</p> <p>Output value of ActiveChannel @ <cfg:Shift Unit="s">0</cfg:Shift></p> <p>Output value of ActiveChannel @ <cfg:Shift Unit="s">-0.001</cfg:Shift></p>



Comment(s) (cont'd)	<ul style="list-style-type: none">• <code>Shift</code> is provided to compensate for laser control delay times.• For each channel, an individual time shift can be specified, that is, <code>Shift</code> is an allowed child tag with <code>AnalogOut1</code>, <code>AnalogOut2</code>, <code>PulseLength</code>, <code>HalfPeriod</code>, <code>SpotDistance</code> (see XML path 1...5, above). This is made possible because, for example, analog signals need a longer settling time than the laser pulse trigger.
Version info	<code>syncAXIS_1_8.xsd</code>

XML tag	RadiusFactor
XML signature (incl. defaults)	<p>RadiusFactor (Enabled, RadiusUnit*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>RadiusUnit attribute value: string.</p> <p>Enabled attribute value: boolean.</p>
XML path(s)	<p>(1) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:AnalogOut1> → <cfg:RadiusFactor></p> <p>(2) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:AnalogOut2> → <cfg:RadiusFactor></p> <p>(3) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:PulseLength> → <cfg:RadiusFactor></p> <p>(4) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:HalfPeriod> → <cfg:RadiusFactor></p> <p>(5) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:SpotDistance> → <cfg:RadiusFactor></p>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre><cfg:RadiusFactor RadiusUnit="mm" Enabled="true"> <cfg:DataPoint Radius="0" Factor="0.0" /> <cfg:DataPoint Radius="15" Factor="1.0" /> <cfg:DataPoint Radius="27" Factor="2.0" /> </cfg:RadiusFactor></pre>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> RadiusFactor is <i>also</i> a container tag. No value(s), 2 attributes: RadiusUnit, Enabled. RadiusFactor is a scaling factor. It is multiplied by the output value of that specific Channel dependent on the excursion radius of the laser beam. It can be used to compensate for energy spread along the spot distortion due to the angle of impact on the workpiece. <div data-bbox="715 1547 1334 2011"> <p>Deflection angle-dependent energy density</p>  <p>α Deflection angle</p> <p>● High energy density</p> <p>● Low energy density</p> <p>INACTIVE</p> <p>ACTIVE</p> </div>

<p>Comment(s) (cont'd)</p>	<ul style="list-style-type: none"> • For this type of automatic laser control setting, a characteristic of individual interpolation points (=child tags <code>DataPoint</code>, see below) can be defined. The values are linearly interpolated inbetween the interpolation points and constantly extrapolated outside of them. • For the <code>RadiusFactor</code> of each Channel (see XML path(s) above), a characteristics can be defined separately even if they are <code>Enabled="false"</code>. • See Chapter 2.9 "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48. <p><code>DataPoint</code></p> <ul style="list-style-type: none"> • XML signature (incl. defaults): <code>DataPoint[]*</code> (Radius, Factor) • <code>syncAXIS_1_8.xsd</code> allows an unlimited number of <code>DataPoint</code> tags. • An interpolation point of a characteristic. • Format: double. • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
<p>Version info</p>	<p><code>syncAXIS_1_8.xsd</code></p>

XML tag	VelocityFactor
XML signature (incl. defaults)	VelocityFactor (Enabled, VelocityUnit*) '*' = optional; no '*' = mandatory. VelocityUnit attribute value: string. Enabled attribute value: boolean.
XML path(s)	(1) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:AnalogOut1> → <cfg:VelocityFactor> (2) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:AnalogOut2> → <cfg:VelocityFactor> (3) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:PulseLength> → <cfg:VelocityFactor> (4) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:HalfPeriod> → <cfg:VelocityFactor> (5) <cfg:Configuration> → <cfg:LaserConfig> → <cfg:AutomaticLaserControl> → <cfg:SpotDistance> → <cfg:VelocityFactor>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre><cfg:VelocityFactor VelocityUnit="mm" Enabled="true"> <cfg:DataPoint Velocity="0" Factor="0.0"/> <cfg:DataPoint Velocity="400" Factor="1.0"/> <cfg:DataPoint Velocity="4000" Factor="2.0"/> </cfg:VelocityFactor></pre>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> VelocityFactor is <i>also</i> a container tag. No value(s), 2 attributes: VelocityUnit, Enabled. VelocityFactor is a scaling factor. It is multiplied by the output value of that specific Channel dependent on the spot velocity of the laser beam. It can be used to compensate for larger energy deposition due to slower marking speed, for example, in corners (without Sky Writing-like motions). For this type of automatic laser control setting, a characteristic of individual interpolation points (=child tags DataPoint, see below) can be defined. The values are linearly interpolated inbetween the interpolation points and constantly extrapolated outside of them. For the VelocityFactor of each Channel (see XML path(s) above), a characteristic can be defined separately even if they are Enabled="false".

Comment(s) (cont'd)	<ul style="list-style-type: none"> SpotDistance as a channel is a special case: the inverse of the characteristic is going to be applied to the actual spot distance. The characteristic affects the velocity information that is used for the "tuning". Also note that using SpotDistance as an "ActiveChannel", velocity information is used independently from the VelocityFactor to calculate the trigger timing to achieve constant spot distances. See Chapter 2.9 "About Automatically Controlling the Laser by syncAXIS control ("Automatic Laser Control")", page 48. <p>DataPoint</p> <ul style="list-style-type: none"> XML signature (incl. defaults): DataPoint[]* (Velocity, Factor) An interpolation point of a characteristic. Format: double. Settable via API?: Not possible. Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Version info	syncAXIS_1_8.xsd

XML tag	TrajectoryConfig
XML signature (incl. defaults)	TrajectoryConfig '*'=optional; kein '*'=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre> <cfg:TrajectoryConfig> <!-- allowed/possible child tags see TrajectoryConfig in the XML structure overview --> </cfg:TrajectoryConfig> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> TrajectoryConfig is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	MarkConfig
XML signature (incl. defaults)	MarkConfig (VelocityUnit*) ''=optional; kein ''=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:MarkConfig>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:MarkConfig> <!-- allowed/possible child tags see MarkConfig in the XML structure overview --> </cfg:MarkConfig></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> MarkConfig is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	JumpSpeed
XML signature (incl. defaults)	<p>JumpSpeed = 400 (Unit*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>Unit attribute value: double.</p> <p>JumpSpeed value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:MarkConfig> → <cfg:JumpSpeed ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:JumpSpeed Unit="mm/s">400</cfg:JumpSpeed>
Settable via API?	slsc_cfg_set_jump_speed slsc_list_set_jump_speed slsc_cfg_set_trajectory_config
Behavior on Module replay	Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Meaning: see JumpSpeed.
Version info	syncAXIS_1_8.xsd

XML tag	MarkSpeed
XML signature (incl. defaults)	<p>MarkSpeed = 400 (Unit*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>Unit attribute value: double.</p> <p>MarkSpeed value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:MarkConfig> → <cfg:MarkSpeed ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:MarkSpeed Unit="mm/s">400</cfg:MarkSpeed>
Settable via API?	<p>slsc_cfg_set_mark_speed</p> <p>slsc_list_set_mark_speed</p> <p>slsc_cfg_set_trajectory_config</p>
Behavior on Module replay	Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Meaning: see MarkSpeed.
Version info	syncAXIS_1_8.xsd

XML tag	MinimalMarkSpeed
XML signature (incl. defaults)	MinimalMarkSpeed = 0 (Unit*)
	<p>'*' = optional; no '*' = mandatory.</p> <p>Unit attribute value: double.</p> <p>MinimalMarkSpeed value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:MarkConfig> → <cfg:MinimalMarkSpeed ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:MinimalMarkSpeed Unit="mm/s">200</cfg:MinimalMarkSpeed>
Settable via API?	slsc_cfg_set_trajectory_config
Behavior on Module replay	Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Meaning: see MinimalMarkSpeed.
Version info	syncAXIS_1_8.xsd

XML tag	LaserSwitchConfig
XML signature (incl. defaults)	<p>LaserSwitchConfig (Unit*)</p> <p>'*' = optional; kein '*' = mandatory.</p> <p>Unit attribute value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:MarkConfig> → <cfg:LaserSwitchConfig>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre><cfg:LaserSwitchConfig> <!-- allowed/possible child tags see LaserSwitchConfig in the XML structure overview --> </cfg:LaserSwitchConfig></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> LaserSwitchConfig is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	LaserPreTriggerTime
XML signature (incl. defaults)	<p>LaserPreTriggerTime = 0 (Unit*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>Unit attribute value: double.</p> <p>LaserPreTriggerTime value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:MarkConfig> → <cfg:LaserSwitchConfig> → <cfg:LaserPreTriggerTime ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:LaserPreTriggerTime Unit="s">1e-6</cfg:LaserPreTriggerTime>
Settable via API?	slsc_cfg_set_trajectory_config
Behavior on Module replay	The parameter value of the replaying syncAXIS control instance is applied. If the parameter value of the replaying syncAXIS control instance is greater than the shortest Jump Segment duration in the Modulee , then [WARN] log file lines are generated. The laser signal is not triggered in advance for the Mark Segments that follow such Jump Segments . See also Section "Behavior on Module replay" , page 66.
Comment(s)	<ul style="list-style-type: none"> Meaning: see LaserPreTriggerTime.
Version info	syncAXIS_1_8.xsd

XML tag	LaserSwitchOffsetTime
XML signature (incl. defaults)	<p>LaserSwitchOffsetTime = -20e-6 (Unit*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>Unit attribute value: double.</p> <p>LaserSwitchOffsetTime value: double.</p>
XML path(s)	<p><cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:MarkConfig> → <cfg:LaserSwitchConfig> → <cfg:LaserSwitchOffsetTime ...></p>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:LaserSwitchOffsetTime Unit="s">-20e-6</cfg:LaserSwitchOffsetTime>
Settable via API?	slsc_cfg_set_trajectory_config
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Meaning: see LaserPreTriggerTime.
Version info	syncAXIS_1_8.xsd

XML tag	LaserMinOffTime
XML signature (incl. defaults)	<p>LaserMinOffTime = 1e-6 (Unit*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>Unit attribute value: double.</p> <p>LaserMinOffTime value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:MarkConfig> → <cfg:LaserSwitchConfig> → <cfg:LaserMinOffTime ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:LaserMinOffTime Unit="s">1.5625e-8</cfg:LaserMinOffTime>
Settable via API?	slsc_cfg_set_trajectory_config
Behavior on Module replay	Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Meaning: see LaserMinOffTime. The smallest allowed value is: 1/64 μs
Version info	syncAXIS_1_8.xsd

XML tag	GeometryConfig
XML signature (incl. defaults)	<p>GeometryConfig</p> <p>'*' = optional; kein '*' = mandatory.</p>
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:GeometryConfig>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre><cfg:GeometryConfig> <!-- allowed/possible child tags see GeometryConfig in the XML structure overview --> </cfg:GeometryConfig></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> GeometryConfig is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	MaxBlendRadius
XML signature (incl. defaults)	MaxBlendRadius = 1.0 (Unit*) **=optional; no **=mandatory. Unit attribute value: double. MaxBlendRadius value: double.
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:GeometryConfig> → <cfg:MaxBlendRadius ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:MaxBlendRadius Unit="mm">1.0</cfg:MaxBlendRadius>
Settable via API?	slsc_cfg_set_trajectory_config
Behavior on Module replay	Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Meaning: see also MaxBlendRadius. Defines the distance from corners starting from which the blending curve might start. Depending on the blending settings, this distance might be smaller. See also Figure 39, page 292.
Version info	syncAXIS_1_8.xsd

XML tag	ApproxBlendLimit
XML signature (incl. defaults)	<p>ApproxBlendLimit = 0.5 (Unit*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>Unit attribute value: double.</p> <p>ApproxBlendLimit value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:GeometryConfig> → <cfg:ApproxBlendLimit ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358 .
XML section example	<cfg:ApproxBlendLimit Unit="mm">0.5</cfg:ApproxBlendLimit>
Settable via API?	slsc_cfg_set_trajectory_config
Behavior on Module replay	Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Meaning: see also ApproxBlendLimit. Defines the distance from corners the blending curves must reach at least. Depending on the blending settings, this distance might be smaller. See also Figure 39, page 292.
Version info	syncAXIS_1_8.xsd

XML tag	BlendMode
XML signature (incl. defaults)	BlendMode = MinimalBlending
	<p>'*' = optional; no '*' = mandatory.</p> <p>BlendMode value: string.</p>
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:GeometryConfig> → <cfg:BlendMode>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:BlendMode>VariableBlending</cfg:BlendMode>
Settable via API?	slsc_cfg_set_trajectory_config
Behavior on Module replay	Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Defines the blend mode of the syncAXIS control instance, see also enum slsc_BlendModes. Allowed entries: <ul style="list-style-type: none"> Deactivated No blending. A Sky Writing-like motion is going to be added. VariableBlending The blending curves are calculated to be as fast as possible while trying to stick to the MaxBlendRadius and ApproxBlendLimit constraint. MinimalBlending The blending curves are as close as possible to the defined contour while maintaining the MinimalMarkSpeed. FixedBlending Deprecated. If a blending cannot be performed sticking to the desired restrictions, a Sky Writing-like motion is going to be added (as with "Deactivated"). To detect its number, the enumeration constant slsc_JobCharacteristic_InsertedSkywritings is available. See also Figure 39, page 292.
Version info	syncAXIS_1_8.xsd

XML tag	AutoCyclicGeometry (deprecated)
XML signature (incl. defaults)	AutoCyclicGeometry = false
	'*' = optional; no '*' = mandatory. AutoCyclicGeometry value: boolean.
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:GeometryConfig> → <cfg:AutoCyclicGeometry>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:AutoCyclicGeometry>false</cfg:AutoCyclicGeometry>
Settable via API?	slsc_cfg_set_trajectory_config
Behavior on Module replay	Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Deprecated. Recommended setting: false. Meaning: see also ApproxBlendLimit. AutoCyclicGeometry was a setting for Splines. Closed contours could be marked with consistent blend settings, even for the starting/end corner, that would normally not have been affected (transition from jump-to-mark and mark-to-jump).
Version info	syncAXIS_1_8.xsd

XML tag	SplineConversionLengthLimit (deprecated)
XML signature (incl. defaults)	<p>SplineConversionLengthLimit = 0.5 (Unit*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>Unit attribute value: double.</p> <p>SplineConversionLengthLimit value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:GeometryConfig> → <cfg:SplineConversionLengthLimit ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358 .
XML section example	<cfg:SplineConversionLengthLimit Unit="mm">0.5</cfg:SplineConversionLengthLimit>
Settable via API?	slsc_cfg_set_trajectory_config
Behavior on Module replay	Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Deprecated. Recommended setting: leave default value as it is. Meaning: see also SplineConversionLengthLimit. SplineConversionLengthLimit has defined the maximum length for vectors to be turned into splines. Vectors longer than the SplineConversionLengthLimit-value had not been added to the spline and blending curves or Sky Writing-like motions have been carried out in between.
Version info	syncAXIS_1_8.xsd

XML tag	SplineMode (deprecated)
XML signature (incl. defaults)	SplineMode = Deactivated
	<p>''=optional; no ''=mandatory.</p> <p>SplineConversionLengthLimit value: string.</p>
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:GeometryConfig> → <cfg:SplineMode>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:SplineMode>Deactivated</cfg:SplineMode>
Settable via API?	slsc_cfg_set_trajectory_config
Behavior on Module replay	Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Deprecated. Recommended setting: Deactivated. Has defined the spline mode of the syncAXIS control instance, see also enum slsc_SplineModes. Allowed entry: <ul style="list-style-type: none"> Deactivated Recommended. No splines. Interpolating Polynomial paths have been interpolated between the marking points. Approximating Polynomial paths have been approximated between the marking points. "Approximating" has reduced the dynamic load whereas "Interpolating" was more accurate. See also Figure 44, page 319.
Version info	syncAXIS_1_8.xsd

XML tag	VectorResolution
XML signature (incl. defaults)	<p>VectorResolution = 0.02 (Unit*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>Unit attribute value: double.</p> <p>VectorResolution value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:TrajectoryConfig> → <cfg:VectorResolution ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:VectorResolution Unit="mm">0.02</cfg:VectorResolution>
Settable via API?	slsc_cfg_set_trajectory_config
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Meaning: see also VectorResolution. Defines the "system resolution" – vectors shorter than that resolution are combined with the previous vector, see Figure 40, page 292.
Version info	syncAXIS_1_8.xsd

XML tag	StageConfig
XML signature (incl. defaults)	StageConfig
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:StageConfig>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:StageConfig> <!-- allowed/possible child tags see StageConfig in the XML structure overview --> </cfg:StageConfig></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> StageConfig is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	DelayShift
XML signature (incl. defaults)	DelayShift* = 0.0 (Unit*) '*'=optional; no '*'=mandatory. Unit attribute value: double. DelayShift value: double.
XML path(s)	<cfg:Configuration> → <cfg:StageConfig> → <cfg:DelayShift ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:DelayShift Unit="s">0.0</cfg:DelayShift>
Settable via API?	Not possible.
Behavior on Module replay	Same as <cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:Delay ...>: The parameter value of the replaying syncAXIS control instance is applied. If the calculated (from <cfg:Configuration> → <cfg:StageConfig> → <cfg:CTIME ...> and <cfg:Configuration> → <cfg:StageConfig> → <cfg:DelayShift ...>) positioning stage delay differs from the one in the Module , then [WARN] log file lines are generated. "SIGNAL" functions included in the Module that have a <i>negative</i> delay are not executed, if the absolute value of the delay is greater than either largest scan device delay (<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:Delay ...>) or calculated positioning stage delay. See also Section "Behavior on Module replay", page 66.
Comment(s)	<ul style="list-style-type: none"> By the DelayShift value, the time synchronization of scan device and positioning stage can be optimized. The default value of 0 should be correct for all XL SCAN systems. Necessary data is automatically read out and applied, see CTIME. If it turns out that the time synchronization needs to be improved, then contact SCANLAB.
Version info	syncAXIS_1_8.xsd

XML tag	CTIME
XML signature (incl. defaults)	<p>CTIME* = -1 (Unit*)</p> <p>'*' = optional; no '*' = mandatory.</p> <p>Unit attribute value: double.</p> <p>CTIME value: double.</p>
XML path(s)	<cfg:Configuration> → <cfg:StageConfig> → <cfg:CTIME ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358 .
XML section example	<cfg:CTIME Unit="ms">-1.0</cfg:CTIME>
Settable via API?	Not possible.
Behavior on Module replay	<p>Same as <cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:Delay ...>: The parameter value of the replaying syncAXIS control instance is applied. If the calculated (from <cfg:Configuration> → <cfg:StageConfig> → <cfg:CTIME ...> and <cfg:Configuration> → <cfg:StageConfig> → <cfg:DelayShift ...>) positioning stage delay differs from the one in the Module, then [WARN] log file lines are generated. "SIGNAL" functions included in the Module that have a <i>negative</i> delay are not executed, if the absolute value of the delay is greater than either largest scan device delay (<cfg:Configuration> → <cfg:ScanDeviceConfig> → <cfg:Delay ...>) or calculated positioning stage delay. See also Section "Behavior on Module replay", page 66.</p>
Comment(s)	<ul style="list-style-type: none"> In syncAXIS control ≤ V1.2.6, the positioning stage delay value needed to be entered based on the cycle time of the ACS protocol (ACS-Cycle Time, "CTIME") under <cfg:Configuration> → <cfg:StageConfig> → <cfg:Delay ...>. In syncAXIS control ≥ V1.3.0, the <cfg:Delay ...> tag is no longer available. Instead, with CTIME = -1, the CTIME is automatically read out from the ACS Motion Controller and a corresponding positioning stage delay value is automatically set. A CTIME value directly entered here is used (no read out from the ACS Motion Controller). A corresponding positioning stage delay value is set automatically. If a positioning stage delay value = 0 is desired, set <cfg:CTIME Unit="ms">0</cfg:CTIME>.
Version info	syncAXIS_1_8.xsd

XML tag	MonitoringLevel
XML signature (incl. defaults)	MonitoringLevel = Jerk
	<p>'*' = optional; no '*' = mandatory.</p> <p>MonitoringLevel value: string.</p>
XML path(s)	<cfg:Configuration> → <cfg:StageConfig> → <cfg:MonitoringLevel>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<cfg:MonitoringLevel> Acceleration </cfg:MonitoringLevel>
Settable via API?	slsc_cfg_set_stage_dynamic_monitoring_level
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Defines a criterion for which the positioning stages are to be monitored. The criterion does <i>not</i> apply to the scan devices – the dedicated tag MonitoringLevel, see page 397, exists for these. The description there applies analogously. Exceedances automatically trigger the reaction defined in DynamicViolationReaction. Allowed entries: <ul style="list-style-type: none"> Deactivated Position Velocity Acceleration Jerk syncAXIS control uses to monitor working field and dynamics as: <ul style="list-style-type: none"> scan device dynamic limits the DynamicLimits values, page 388 scan device working field limits the FieldLimits values, page 395 scan device monitoring criterion the MonitoringLevel value, page 397 positioning stage dynamic limits the FieldLimits values, page 454 positioning stage working field limits the DynamicLimits values, page 456 positioning stage monitoring criterion the MonitoringLevel values, page 452 reaction on exceedances the DynamicViolationReaction values, page 365
Version info	syncAXIS_1_8.xsd

XML tag	StageList
XML signature (incl. defaults)	StageList*
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:StageList> <!-- allowed/possible child tags see StageList in the XML structure overview --> </cfg:StageList></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> StageList is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	Stage
XML signature (incl. defaults)	Stage[] (Name)
	'*' = optional; no '*' = mandatory. Name attribute value: string.
XML path(s)	<cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:Stage Name="Stage1"> <!-- allowed/possible child tags see Stage in the XML structure overview --> </cfg:Stage></pre>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Stage is <i>also</i> a container tag. No value(s), 1 attribute: Name. syncAXIS_1_8.xsd allows up to 4 Stage tags. Allowed entries with Name, see enum slsc_Stage: "Stage1", "Stage2", "Stage3", "Stage4".
Version info	syncAXIS_1_8.xsd

XML tag	FieldLimits
XML signature (incl. defaults)	FieldLimits
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage> → <cfg:FieldLimits>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:FieldLimits> <cfg:XDirection Unit="mm" Max="150" Min="-150" /> <cfg:YDirection Unit="mm" Max="150" Min="-150" /> </cfg:FieldLimits></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> FieldLimits is <i>only</i> a container tag. No value(s), no attribute(s). The FieldLimits child tags XDirection and YDirection serve to specify the working field limits of the intended positioning stage type. syncAXIS control uses to monitor working field and dynamics as: <ul style="list-style-type: none"> scan device dynamic limits the DynamicLimits values, page 388 scan device working field limits the FieldLimits values, page 395 scan device monitoring criterion the MonitoringLevel value, page 397 positioning stage dynamic limits the FieldLimits values, page 454 positioning stage working field limits the DynamicLimits values, page 456 positioning stage monitoring criterion the MonitoringLevel values, page 452 reaction on exceedances the DynamicViolationReaction values, page 365 syncAXIS control does <i>not</i> use the values at XDirection and YDirection to plan trajectories! syncAXIS Viewer uses the values for values at XDirection and YDirection to visualize the positioning stage working field and to indicate limit value exceedances. The values at XDirection and YDirection do not need to correspond the usable working field. Lower field boundaries can also be set, if a smaller part of the working field is to be monitored. Higher field boundaries should not be set.

<p>Comment(s) (cont'd)</p>	<p>XDirection</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): XDirection (Unit*, Max, Min) • Format: double. • Settable via API?: slsc_cfg_set_field_limits_stage • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>YDirection</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): YDirection (Unit*, Max, Min) • Format: double. • Settable via API?: slsc_cfg_set_field_limits_stage • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>ZDirection</p> <ul style="list-style-type: none"> • <i>This parameter is currently reserved!</i> • XML signature (incl. defaults): ZDirection* (Unit*, Max, Min) • Max, Min: double. • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
<p>Version info</p>	<p>syncAXIS_1_8.xsd</p>

XML tag	DynamicLimits
XML signature (incl. defaults)	DynamicLimits ''=optional; no ''=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage> → <cfg:DynamicLimits>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:DynamicLimits> <cfg:Velocity Unit="mm/s">1000</cfg:Velocity> <cfg:Acceleration Unit="mm/s^2">10000</cfg:Acceleration> <cfg:Jerk Unit="mm/s^3">100000</cfg:Jerk> </cfg:DynamicLimits></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> DynamicLimits is <i>only</i> a container tag. No value(s), no attribute(s). The DynamicLimits child tags Velocity, Acceleration and Jerk serve to specify the dynamic limits of the intended positioning stage type. syncAXIS control uses to monitor working field and dynamics as: <ul style="list-style-type: none"> scan device dynamic limits the DynamicLimits values, page 388 scan device working field limits the FieldLimits values, page 395 scan device monitoring criterion the MonitoringLevel value, page 397 positioning stage dynamic limits the FieldLimits values, page 454 positioning stage working field limits the DynamicLimits values, page 456 positioning stage monitoring criterion the MonitoringLevel values, page 452 reaction on exceedances the DynamicViolationReaction values, page 365 syncAXIS control does <i>not</i> use the values at Velocity, Acceleration and Jerk to plan trajectories! For this purpose, CalculationDynamics is used. syncAXIS Viewer uses the values at Velocity, Acceleration and Jerk to indicate dynamic limit value exceedances. The values at Velocity, Acceleration and Jerk to do not need to correspond the usable working field. Lower dynamic limits can also be set if the dynamic limits are to be monitored more strictly.

<p>Comment(s) (cont'd)</p>	<p>Velocity</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): <code>Velocity = 1000.0 (Unit*)</code> • Format: double. • Settable via API?: <code>slsc_cfg_set_dynamic_limits_stage</code> • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>Acceleration</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): <code>Acceleration = 10000.0 (Unit*)</code> • Format: double. • Settable via API?: <code>slsc_cfg_set_dynamic_limits_stage</code> • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>Jerk</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): <code>Jerk = 100000.0 (Unit*)</code> • Format: double. • Settable via API?: <code>slsc_cfg_set_dynamic_limits_stage</code> • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
<p>Version info</p>	<p>syncAXIS_1_8.xsd</p>

XML tag	CalculationDynamics
XML signature (incl. defaults)	CalculationDynamics ''=optional; no ''=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage> → <cfg:CalculationDynamics>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:CalculationDynamics> <cfg:Velocity Unit="mm/s">1000</cfg:Velocity> <cfg:Acceleration Unit="mm/s^2">10000</cfg:Acceleration> <cfg:Jerk Unit="mm/s^3">100000</cfg:Jerk> </cfg:CalculationDynamics></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> CalculationDynamics is <i>only</i> a container tag. No value(s), no attribute(s). The CalculationDynamics child tags Velocity, Acceleration and Jerk serve to specify the dynamic capabilities ("dynamic limits") of the intended positioning stage type, which are used to calculate the positioning stage motion ⚠ Caution! syncAXIS control uses the values at Velocity, Acceleration and Jerk to plan trajectories for the Operation mode "StageOnly" as well as for the end motion at Job ends. Make sure that the entered values are correct. In "ScannerAndStage" mode, these values: <ul style="list-style-type: none"> are <i>not</i> taken into account for marking motions (the Motion decomposition is performed based on the FilterBandwidth value) are taken into account (for the end motion) at Job ends The following applies to Operation mode "StageOnly": If a higher marking speed or jump speed is specified (via syncAXISConfig.xml or function call), then these are automatically reduced to the Velocity value. In this case, an [ERROR] log file line is generated, see [ERROR] log file lines.

<p>Comment(s) (cont'd)</p>	<p>Velocity</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): <code>Velocity = 1000.0 (Unit*)</code> • Format: double • Settable via API?: <code>slsc_cfg_set_calculation_dynamics_stage</code> • <code>Velocity</code> is the velocity of the positioning stage ("dynamic limit") • Behavior on Module replay: The Module is rejected, if the replaying syncAXIS control instance is in Operation mode <code>StageOnly</code> and the <code>Velocity</code> value is smaller than <code>MarkSpeed</code> value or <code>JumpSpeed</code> value in the Module. In this case, you need to record a new Module with correspondingly different parameter value. See also Section "Behavior on Module replay", page 66. <p>Acceleration</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): <code>Acceleration = 10000.0 (Unit*)</code> • Format: double • Settable via API?: <code>slsc_cfg_set_calculation_dynamics_stage</code> • <code>Acceleration</code> is the acceleration of the positioning stage ("dynamic limit") • Behavior on Module replay: The Module is rejected, if the replaying syncAXIS control instance is in Operation mode <code>StageOnly</code> and the <code>Acceleration</code> value is smaller than in the Module. In this case, you need to record a new Module with correspondingly different parameter value. See also Section "Behavior on Module replay", page 66. <p>Jerk</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): <code>Jerk = 100000.0 (Unit*)</code> • Format: double • Settable via API?: <code>slsc_cfg_set_calculation_dynamics_stage</code> • <code>Jerk</code> is the jerk of the positioning stage ("dynamic limit") • Behavior on Module replay: The Module is rejected, if the replaying syncAXIS control instance is in Operation mode <code>StageOnly</code> and the <code>Jerk</code> value is smaller than in the Module. In this case, you need to record a new Module with correspondingly different parameter value. See also Section "Behavior on Module replay", page 66.
<p>Version info</p>	<p><code>syncAXIS_1_8.xsd</code></p>

XML tag	Alignment
XML signature (incl. defaults)	Alignment
	'*' = optional; no '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage ...> → <cfg:Alignment>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:Alignment> <cfg:Matrix> <cfg:T11>1</cfg:T11> <cfg:T12>0</cfg:T12> <cfg:T21>0</cfg:T21> <cfg:T22>1</cfg:T22> </cfg:Matrix> <cfg:Offset X="0" Y="0" /> </cfg:Alignment> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> Alignment is <i>only</i> a container tag. No value(s), no attribute(s). The Alignment-child tags Matrix and Offset are available to compensate for assembly errors in Multi-Head systems. The positions sent to a respective positioning stage have been transformed in a way that positioning stage coordinate system and reference system (for example, the scan head coordinate system) do match. Alignment is applied to the Trajectory planning! (Alignment occurs not on the RTC6!) A simulation file does not contain alignment transformation (simulation files show the marking not taking into account any miscalibration). The matrix must be invertible, but does not need to be normed. For further information, see Chapter 8.3 "About Transformations in syncAXIS control V1.2.4 and Higher", page 332. <p>Matrix</p> <ul style="list-style-type: none"> XML signature (incl. defaults): Matrix No value(s), no attribute(s). $\text{Matrix} = \begin{bmatrix} T11 & T12 \\ T21 & T22 \end{bmatrix}$ <ul style="list-style-type: none"> Settable via API?: Not possible. Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.

<p>Comment(s) (cont'd)</p>	<p>T11</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): T11 = 1 • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>T12</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): T12 = 0 • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>T21</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): T21 = 0 • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>T22</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): T22 = 1 • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>Offset</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): Offset (X = 0, Y = 0, Unit*) • Format: double. • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
<p>Version info</p>	<p>syncAXIS_1_8.xsd</p>

XML tag	StageAxisX
XML signature (incl. defaults)	StageAxisX* = 0
	'*' = optional; no '*' = mandatory. StageAxisX value: non-negative integer.
XML path(s)	<cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage ...> → <cfg:StageAxisX>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:StageAxisX>0</cfg:StageAxisX>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> For positioning stage setups where ACS x axis index ≠ 0.
Version info	syncAXIS_1_8.xsd

XML tag	StageAxisY
XML signature (incl. defaults)	StageAxisY* = 1
	'*' = optional; no '*' = mandatory. StageAxisY value: non-negative integer.
XML path(s)	<cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage ...> → <cfg:StageAxisY>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:StageAxisY>1</cfg:StageAxisY>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> For positioning stage setups where ACS x axis index ≠ 1.
Version info	syncAXIS_1_8.xsd

XML tag	SlecEtherCATNodeID
XML signature (incl. defaults)	SlecEtherCATNodeID* = 0
	'*' = optional; no '*' = mandatory. SlecEtherCATNodeID value: non-negative integer.
XML path(s)	<cfg:Configuration> → <cfg:StageConfig> → <cfg:StageList> → <cfg:Stage ...> → <cfg:SlecEtherCATNodeID>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:SlecEtherCATNodeID>1</cfg:SlecEtherCATNodeID>
Settable via API?	Not possible.
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> Position of the SLEC in the EtherCAT network. For positioning stage setups where SLEC-ID ≠ 0.
Version info	syncAXIS_1_8.xsd

XML tag	IOConfig
XML signature (incl. defaults)	IOConfig*
	'*' = optional; kein '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:IOConfig>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:IOConfig> <!-- allowed/possible child tags see IOConfig in the XML structure overview --> </cfg:IOConfig></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> IOConfig is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	DefaultOutputs
XML signature (incl. defaults)	DefaultOutputs* ''=optional; no ''=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:IOConfig> → <cfg:DefaultOutputs>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:DefaultOutputs> <cfg:LaserPinOut Format="Bitpattern" Value="1" /> <cfg:AnalogOut1 Format="Factor" Value="0.5" /> <cfg:AnalogOut2 Format="Factor" Value="0.5" /> </cfg:DefaultOutputs></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> DefaultOutputs is <i>only</i> a container tag. No value(s), no attribute(s). After RTC6 startup, all analog ports are low (0 V) and the digital ports are in the (high impedance) tristate mode. Provided the respective port is not defined as an "ActiveChannel" (see Channel): the specified values are output as of the first Job execution start and continue until they are changed by an API function or, the LaserShutdownSequence is executed. <p>LaserPinOut</p> <ul style="list-style-type: none"> XML signature (incl. defaults): LaserPinOut* (Format*, Value) Format: string. Allowed entries: 0, 1, 2, 3. In case "Automatic Laser Control" is not used, this value is output at the respective port during Job execution. Settable via API?: slsc_list_write_analog_x, slsc_list_write_digital_out Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.

<p>Comment(s) (cont'd)</p>	<p>AnalogOut1</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): AnalogOut1* (Format*, Value) • Format: double. • Allowed entries: 0...1. • The factor for the analog output applies with respect to the maximum value of 10 V. In case "Automatic Laser Control" is not used, this value is output at the respective port during Job execution. • Settable via API?: slsc_list_write_analog_x, slsc_list_write_digital_out • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>AnalogOut2</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): AnalogOut2* (Format*, Value) • Format: double. • Allowed entries: 0...1. • The factor for the analog output applies with respect to the maximum value of 10 V. In case "Automatic Laser Control" is not used, this value is output at the respective port during Job execution. • Settable via API?: slsc_list_write_analog_x, slsc_list_write_digital_out • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
<p>Version info</p>	<p>syncAXIS_1_8.xsd</p>

XML tag	LaserInitSequence
XML signature (incl. defaults)	LaserInitSequence{ }* '*'=optional; no '*'=mandatory. '{}'=here: any number of child tags are allowed ("sequence").
XML path(s)	<cfg:Configuration> → <cfg:IOConfig> → <cfg:LaserInitSequence>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre> <cfg:LaserInitSequence> <cfg:Delay>0</cfg:Delay> <cfg:SetLaserPinOut Format="Bitpattern" Value="1" /> <cfg:SetAnalogOut1 Format="Factor" Value="0.5" /> <cfg:SetAnalogOut2 Format="Factor" Value="1.0" /> <cfg:SetExt1DigitalOut Value="1" /> </cfg:LaserInitSequence> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> LaserInitSequence is <i>only</i> a container tag. No value(s), no attribute(s). The individual child tags of LaserInitSequence are processed sequentially: <ul style="list-style-type: none"> During initialization of a syncAXIS control instance During positioning stage acquisition due to slsc_cfg_acquire_stage (deprecated) This can be used to initialize the laser (for example, to set the Global Enable of the laser) and peripherals. syncAXIS_1_8.xsd allows that LaserInitSequence child tags may occur several times. After RTC6 startup, all analog ports are low (0 V) and the digital ports are in the (high impedance) tristate mode. The most recently set values are output until: <ul style="list-style-type: none"> The first Job execution starts An API function call overwrites them <p>Delay</p> <ul style="list-style-type: none"> XML signature (incl. defaults): Delay (Unit*) Format: non-negative double. Settable via API?: Not possible. Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.

<p>Comment(s) (cont'd)</p>	<p>SetLaserPinOut</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): SetLaserPinOut (Format*, Value) • Format: string. • Allowed entries: 0, 1, 2, 3. • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>SetAnalogOut1</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): SetAnalogOut1 (Format*, Value) • Format: double. • Allowed entries: 0...1. • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>SetAnalogOut2</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): SetAnalogOut1 (Format*, Value) • Format: double. • Allowed entries: 0...1. • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>SetExt1DigitalOut</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): SetExt1DigitalOut (Format*, Value, Mask*) • Format: double. • Allowed entries: 0...65535 (decimal) or 0x0000...0xFFFF (hex). • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
<p>Version info</p>	<p>syncAXIS_1_8.xsd</p>

XML tag	LaserShutdownSequence
XML signature (incl. defaults)	LaserShutdownSequence{ }*
	<p>'*' = optional; no '*' = mandatory.</p> <p>'{}' = here: any number of child tags are allowed ("sequence").</p>
XML path(s)	<cfg:Configuration> → <cfg:IOConfig> → <cfg:LaserShutdownSequence>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre> <cfg:LaserShutdownSequence> <cfg:Delay>0</cfg:Delay> <cfg:SetLaserPinOut Format="Bitpattern" Value="0" /> <cfg:SetAnalogOut2 Format="Factor" Value="0.0" /> <cfg:SetAnalogOut1 Format="Factor" Value="0.0" /> <cfg:SetExt1DigitalOut Value="0" /> </cfg:LaserShutdownSequence> </pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> LaserShutdownSequence is <i>only</i> a container tag. No value(s), no attribute(s). The individual child tags are processed sequentially: <ul style="list-style-type: none"> During syncAXIS control instance destruction During positioning stage release due to slsc_cfg_release_stage (deprecated) This can be used to shut-down the laser (for example, to switch off the Global Enable of the laser) and peripherals. syncAXIS_1_8.xsd allows that LaserShutdownSequence child tags may occur several times. After RTC6 startup, all analog ports are low (0 V) and the digital ports are in the (high impedance) tristate mode. <p>Delay</p> <ul style="list-style-type: none"> XML signature (incl. defaults): Delay (Unit*) Format: non-negative double. Settable via API?: Not possible. Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.

<p>Comment(s) (cont'd)</p>	<p>SetLaserPinOut</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): SetLaserPinOut (Format*, Value) • Format: string. • Allowed entries: 0, 1, 2, 3. • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>SetAnalogOut1</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): SetAnalogOut1 (Format*, Value) • Format: double. • Allowed entries: 0...1. • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>SetAnalogOut2</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): SetAnalogOut1 (Format*, Value) • Format: double. • Allowed entries: 0...1. • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this. <p>SetExt1DigitalOut</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): SetExt1DigitalOut (Format*, Value, Mask*) • Format: double. • Allowed entries: 0...65535 (decimal) or 0x0000...0xFFFF (hex). • Settable via API?: Not possible. • Behavior on Module replay: Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
<p>Version info</p>	<p>syncAXIS_1_8.xsd</p>

XML tag	MotionDecompositionConfig
XML signature (incl. defaults)	MotionDecompositionConfig
	'*' = optional; kein '*' = mandatory.
XML path(s)	<cfg:Configuration> → <cfg:MotionDecompositionConfig>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre><cfg:MotionDecompositionConfig> <!-- allowed/possible child tags see MotionDecompositionConfig in the XML structure overview --> </cfg:MotionDecompositionConfig></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> MotionDecompositionConfig is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	FilterBandwidth
XML signature (incl. defaults)	FilterBandwidth = 2 (Unit*) ''=optional; no ''=mandatory. FilterBandwidth value: double.
XML path(s)	<cfg:Configuration> → <cfg:MotionDecompositionConfig> → <cfg:FilterBandwidth>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<cfg:FilterBandwidth>2</cfg:FilterBandwidth>
Settable via API?	slsc_cfg_set_bandwidth
Behavior on Module replay	Standard behavior: The parameter value of the replaying syncAXIS control instance is applied. If you want to vary the parameter value, then you do not need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> slsc_cfg_initialize_from_file fails, if the FilterBandwidth value is smaller than 0.23. Then, the return value indicates that Bit #14 is set (XmlLoadError). The FilterBandwidth value is the very parameter that determines the decomposition of scan device and positioning stage motion. The larger the FilterBandwidth value, the more dynamic load is assigned to the positioning stage. The lower, the more load to the scan device. Dependent on the process constraints, users might desire <ul style="list-style-type: none"> – minimum positioning stage load in order to reduce inaccuracies due to positioning stage vibrations, – minimal scanner working field to reduce inaccuracies due to scanner calibration inaccuracies, or – minimum process time by a maximum utilization of the scanners and positioning stage dynamic capabilities. The optimal FilterBandwidth value strongly depends on the contour. It can be determined by checking the dynamic load (the Trajectory planning has calculated) by using the simulation mode, see Chapter 2.6 "About Optimizing syncAXIS control-based User Programs", page 36. The dynamic and position capabilities of scanner and positioning stage are not taken into account for the Motion decomposition (motion assignment). This is another reason why you always must always simulate every Job in advance before executing it on hardware for the first time. This is the only way you can ensure that no limits are violated. See Chapter 2.2 "About the SAFE Use of syncAXIS control – General Approach", page 18.
Version info	syncAXIS_1_8.xsd

XML tag	HeuristicConfig
XML signature (incl. defaults)	HeuristicConfig* ''=optional; kein ''=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:MotionDecompositionConfig> → <cfg:HeuristicConfig>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre><cfg:HeuristicConfig> <!-- allowed/possible child tags see HeuristicConfig in the XML structure overview --> </cfg:HeuristicConfig></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> HeuristicConfig is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	DynamicReductionFunctions
XML signature (incl. defaults)	DynamicReductionFunctions* ''=optional; kein ''=mandatory.
XML path(s)	<cfg:Configuration> → <cfg:MotionDecompositionConfig> → <cfg:HeuristicConfig> → <cfg:DynamicReductionFunctions>
XML structure overview	See Chapter 13.1 "xml-Structure Overview", page 358.
XML section example	<pre><cfg:HeuristicConfig> <!-- allowed/possible child tags see DynamicReductionFunctions in the XML structure overview --> </cfg:HeuristicConfig></pre>
Settable via API?	Not possible.
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> DynamicReductionFunctions is <i>only</i> a container tag. No value(s), no attribute(s).
Version info	syncAXIS_1_8.xsd

XML tag	DynamicReductionFunction
XML signature (incl. defaults)	DynamicReductionFunction[]* (Units*) ''=optional; no ''=mandatory. DynamicReductionFunction attribute value: 'mm and mm/s'.
XML path(s)	<cfg:Configuration> → <cfg:MotionDecompositionConfig> → <cfg:HeuristicConfig> → <cfg:DynamicReductionFunctions> → <cfg:DynamicReductionFunction ...>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<pre> <cfg:DynamicReductionFunction Units="mm and mm/s"> <cfg:DataPoint Length="0.0" Velocity="2000" /> <cfg:DataPoint Length="27.0" Velocity="2000" /> <cfg:DataPoint Length="27.01" Velocity="700" /> <cfg:DataPoint Length="54.0" Velocity="700" /> </cfg:DynamicReductionFunction> </pre>
Settable via API?	slsc_cfg_select_heuristic
Behavior on Module replay	No behavior as it is a container tag
Comment(s)	<ul style="list-style-type: none"> • Purpose of the <code>DynamicReductionFunction</code> tag is to be able to define speed reduction characteristics, see list bullet "Use case" below. • The Heuristic cannot be applied, if no characteristic is defined at all (= there is no <code>DynamicReductionFunction</code>). • <code>syncAXIS_1_8.xsd</code> allows an unlimited number of <code>DynamicReductionFunction</code> tags. Thus, any number of characteristics can be defined. • The syncAXIS control instance is initialized with the values of the first (topmost) <code>DynamicReductionFunction</code> tag (corresponds to slsc_cfg_select_heuristic(<code>HeuristicIndex</code> = 0)). • Thus, the first (= topmost) characteristic is used as default. • See also Chapter 2.10 "About Heuristic and Characteristics for Speed Reductions", page 64. • Use case: With long vectors (exceeding the length of half the scanner field of view) executed with high velocities, the positioning stage might experience a higher dynamic load than it is capable. For those cases, to reduce the combined velocity, a characteristic can be created: for this purpose, the individual characteristic interpolation points (with length and velocity information) are defined by means of <code>DataPoint</code> tags. Furthermore, you can define dedicated characteristics (for example, for low-frequency marking patterns and small high-frequency marking patterns) each in its own <code>DynamicReductionFunction</code> tag. The desired characteristics is set by slsc_cfg_select_heuristic(<code>HeuristicIndex</code> = [0...(<code>DynamicReductionFunction</code> – 1)]). Note that in rare cases (for example, with vectors that are a little longer than half the scan head working field and therefore, do not exceed any limit values) the execution time can be longer with a characteristic than without a characteristic. • Non-use case: Many short vectors that follow each other.

<p>Comment(s) (cont'd)</p>	<ul style="list-style-type: none"> • <code>HeuristicForJumpsOnly = true</code> sets that the characteristic is applied only to Jump Segments and not to Mark Segments. <p>DataPoint</p> <ul style="list-style-type: none"> • XML signature (incl. defaults): DataPoint[] (Length, Velocity) • <code>syncAXIS_1_8.xsd</code> allows an unlimited number of DataPoint tags. • An interpolation point of a characteristic. • Format: double. • Settable via API?: Not possible. • Make sure that the DataPoint tags follow each other as shown in XML section example, page 473: <ul style="list-style-type: none"> – Ascending Length values – Descending Velocity values • Behavior on Module replay: Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.
<p>Version info</p>	<p><code>syncAXIS_1_8.xsd</code></p>

XML tag	HeuristicForJumpsOnly
XML signature (incl. defaults)	HeuristicForJumpsOnly* = false
	'**'=optional; no '*'=mandatory. HeuristicForJumpsOnly value: boolean.
XML path(s)	<code><cfg:Configuration> → <cfg:MotionDecompositionConfig> → <cfg:HeuristicConfig> → <cfg:HeuristicForJumpsOnly ...></code>
XML structure overview	See Chapter 13.1 "xml-Structure Overview" , page 358.
XML section example	<code><cfg:HeuristicForJumpsOnly>true</cfg:HeuristicForJumpsOnly></code>
Settable via API?	Not possible.
Behavior on Module replay	Non-Standard behavior: The parameter value of the replaying syncAXIS control instance is not applied. If you want to vary the parameter value, then you need to record a new Module each time for this.
Comment(s)	<ul style="list-style-type: none"> HeuristicForJumpsOnly refers to the characteristic created under DynamicReductionFunction (which determines the syncAXIS-DLL-behavior with long vectors): <ul style="list-style-type: none"> – true The characteristic is applied only to Jump Segments but not to Mark Segments. This setting avoids burn-ins and bad marking results when non-pulse-on-demand lasers are used. – false The characteristic is applied to Jump Segments as well as to Mark Segments. The dynamic and position capabilities of scan device and positioning stage are not taken into account for the Motion decomposition (motion assignment). This is another reason why you always must always simulate every Job in advance before executing it on hardware for the first time. This is the only way you can ensure that no limits are violated. See Chapter 2.2 "About the SAFE Use of syncAXIS control – General Approach", page 18.
Version info	syncAXIS_1_8.xsd

14 Change Index

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

Changes to document revision **1.9.19 en-US** from document revision 0.9.18 en-US

Where	What
Global	Document Revision <ul style="list-style-type: none"> 1.9.19 en-US applies to syncAXIS control-software package <ul style="list-style-type: none"> V1.7.0
slsc_cfg_get_jump_time , page 128	Software change. New function.
Chapter 12 "Appendix E: Application Note – C#", page 350	Editorial enhancement.
Change Index, page 476	



Changes to document revision **1.9.20 en-US** from document revision **1.9.19 en-US**

Where	What
Global	Document Revision <ul style="list-style-type: none"> 1.9.20 en-US applies to syncAXIS control-software package <ul style="list-style-type: none"> V1.8.0
Chapter 5 "Error Codes with slsc_ctrl_get_error, Log File and Console", page 282	Software change. Deleted error code: 0x 00 00 00 00 nn nn nn nn RTC6 ERROR.
Chapter 12.4 "Code Example 2 (C#)", page 357	Editorial enhancement.
EthMaxTimeout, page 386	Software change. New tag.
Change Index, page 476	



Notes