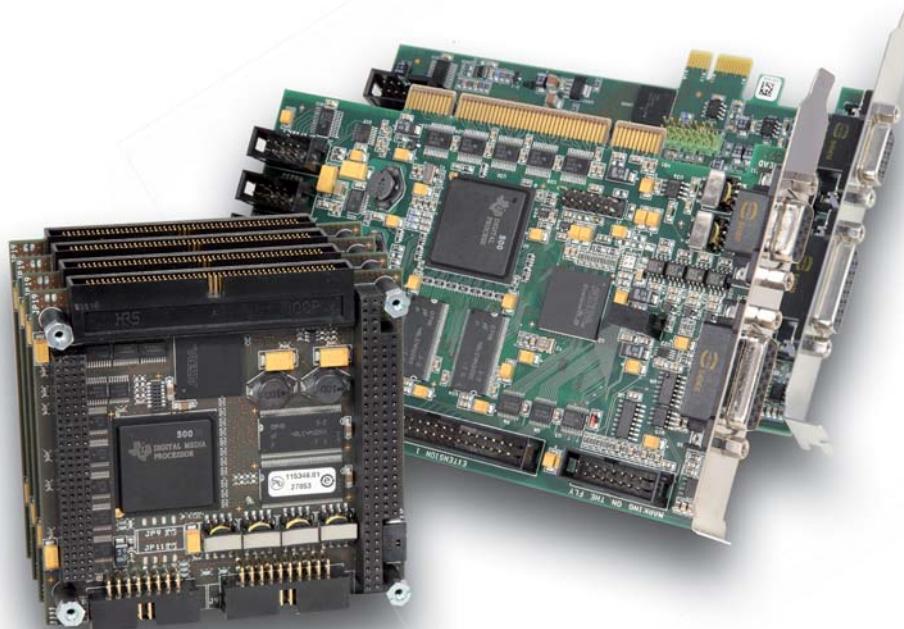




Installation and Operation

**RTC5 PCI Board,
RTC5 PCI Express Board,
RTC5 PC/104-Plus Board and
RTC5 PCIe/104 Board**

for Real Time Control of Scan Heads and Lasers



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 2017
(Doc. Rev. 1.13.0 e - Dezember 15, 2017)

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

No part of this manual 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.

RTC is a registered trademark of SCANLAB GmbH.

PC/104-Plus and PCIe/104 are a registered trademarks of the "PC/104 Embedded Consortium".

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



Contents

1	Introduction	21
1.1	Manufacturer	21
1.2	Labeling	21
1.3	Unpacking Instructions and Typical Package Contents	22
1.3.1	Delivered Software	22
2	Product Overview	24
2.1	Intended Use	24
2.2	System Requirements	26
2.2.1	Hardware	26
2.2.2	Software	26
2.3	Optional Functionality	27
2.4	Jumper Settings and Type Identification	28
2.4.1	Jumper JP1: Selecting the Output Signal Level at the EXTENSION 1 Socket Connector	28
2.4.2	Jumper JP2–JP8: Configuring Pin15 and Pin17 at the EXTENSION 2 Socket Connector	28
2.4.3	Type Identification	29
2.5	Accessories	29
2.5.1	XY2-100 Converter	29
2.5.2	Data Cables	29
2.5.3	Laser Adapter	29
2.5.4	PCI Slot Covers	29
2.5.5	ADC Add-On Board	30
2.5.6	RTC5 varioSCAN FLEX Extension Board	30
2.6	Supplementary Software	30
2.7	Notes for RTC4 Users	31
2.7.1	Hardware Changes	31
Scan System Control	31	
Laser Control	31	
EXTENSION 1 Socket Connector	31	
EXTENSION 2 Socket Connector	32	
MARKING ON THE FLY Socket Connector	32	
Other Interfaces	32	
2.7.2	Porting RTC4 Programs to the RTC5	32
Changed Initialization	32	
Command Changes	32	
Increased Parameter Resolution	33	
Changed Timing Behavior	34	
2.7.3	New and Changed Functionality	35
Interface to the PC	35	
Scan System Control	35	
Laser Control	36	
Interfaces for Peripheral Equipment	36	
General Programming	37	
Laser Marking	37	
Special Functions	38	
3	Safety During Installation and Operation	39
3.1	Steps for Safe Operation	39
3.2	Laser Safety	39



4	Layout and Interfaces	40
4.1	Connectors and Jumper Positions	40
4.2	Interface to PC	41
4.2.1	Master/Slave Synchronization	41
4.3	Interfaces to Scan System	42
4.3.1	Scan Head Connectors and Transfer Protocol	42
Primary Scan Head Connector		42
Secondary Scan Head Connector (Optionally Activated)		43
4.3.2	XY2-100 Converter (Optional)	43
4.3.3	Data Cables (Accessories)	45
4.4	Interfaces for the Laser and Peripheral Equipment	47
4.4.1	Laser Connector	47
Laser Signals		47
External Control Signals		47
BUSY Status		47
Digital Input and Output		48
Analog Output Ports		48
I/O Circuits		48
Laser Adapter (Optional)		50
4.4.2	EXTENSION 1 Socket Connector	51
Configuring the Output Signal Level		51
16-Bit Digital Input and Output		51
Synchronization of Data Transmission		51
BUSY Status		51
4.4.3	EXTENSION 2 Socket Connector	52
Jumper Setting		52
Laser Signals		52
8-Bit Digital Output Port		52
4.4.4	MARKING ON THE FLY Socket Connector	53
Encoder Inputs		53
External Control Signals		53
Analog Output Port		53
BUSY Status		53
Slot Cover (Optional)		53
4.4.5	RS232 Socket Connector	54
4.4.6	SPI / I2C Socket Connector	54
Use as McBSP Interface		54
Use as I ² C Interface		57
Use as SPI Interface		57
4.4.7	STEPPER MOTOR Socket Connector (Stepper Motor Control)	58
4.4.8	Analog Inputs (Optional Accessory)	59
5	Installation and Start-Up	60
5.1	Jumper Settings	60
5.2	Installing the RTC5 Board	60
5.3	Installing the Drivers	61
5.3.1	Software Package Upgrades	62
5.3.2	Exchange of RTC Boards and Update of RTC Board Drivers	62
5.4	Installing the RTC5 Software	62
5.5	Safe Start-up and Shutdown Sequences	63
5.6	Functionality Test	63
5.7	User Programs and Demo Software	64
5.8	Exchanging RTC5 Boards	64



6	Developing User Programs	65
6.1	RTC5 Software Basics	65
6.1.1	Controlling Scan Systems and Lasers – An Introductory Example	65
6.1.2	List Commands and Control Commands	65
6.2	Initialization and Program Start-Up	66
6.2.1	DLL Call	66
6.2.2	Importing Commands	66
Pascal	66	
C	67	
C++	67	
C#	67	
6.2.3	Initializing the DLL and Board Management	68
6.2.4	Start of Operation	69
Initialization of the Board	69	
Configuration of the Board	69	
Initialization of the Scan System Control	69	
Initialization of the Laser Control	70	
Loading and Executing Lists	70	
6.2.5	Example Code	71
6.3	List Memory	73
6.3.1	Lists and the Protected Buffer Area	73
Lists	73	
Protected Buffer Area	73	
6.3.2	Configuring the List Memory	74
6.4	List Handling	75
6.4.1	Loading Lists	75
“Unconditional” Loading	75	
Loading with Protection	75	
Terminating Lists	76	
6.4.2	List Status	76
6.4.3	List Execution Status	77
6.4.4	Starting and Stopping Lists	78
6.4.5	Interrupting Lists for Synchronization of Processing	79
6.4.6	Automatic List Changing	79
One-Time List Change	79	
Alternating List Changes	80	
6.5	Structured Programming	81
6.5.1	Subroutines	81
Non-Indexed Subroutines	81	
Indexed Subroutines	82	
Calling Subroutines	83	
Subsequent Protection and Conversion of Non-Indexed Subroutines	84	
Deprotecting Subroutines	85	
Index Management and Defragmentation	85	
6.5.2	Character Sets and Text Strings	87
Defining Indexed Character Sets	87	
Calling Indexed Characters	88	
Defining Indexed Text Strings for Times, Dates and Serial Numbers	88	
Calling Indexed Text Strings	89	
Management of Indexed Characters and Text Strings	89	
6.5.3	Jumps	89
6.5.4	Circular Queue Mode	90



6.5.5	Loops	91
6.6	Using Multiple RTC5 Boards in One Computer	92
6.6.1	Multi-Board Programming	92
6.6.2	Sequential Programming	93
6.6.3	Master/Slave Operation	93
	Initialization	93
	Clock Phase Synchronization	94
	Matching of Short-Command Counts (as of Version DLL 523, OUT 524)	94
	Synchronous Starts and Stops	94
	Example Code	95
6.7	Usage by Multiple Applications	96
6.7.1	Board Acquisition by a User Program	96
6.8	Error Handling	98
6.8.1	Download Verification	99
6.8.2	Checking for Overruns	99
6.8.3	Example Code	100
6.9	Miscellaneous	102
6.9.1	Free Variables	102
7	Basic Functions for Scan Head and Laser Control	103
7.1	Marking Points, Lines and Arcs	103
7.1.1	Marking with Vector and Arc Commands	103
	Jump Commands	104
	Mark Commands	104
	Arc Commands	104
	Ellipse Commands	105
	Para Commands	106
7.1.2	Microsteps	107
7.1.3	Marking Points	108
7.1.4	Example Code	109
7.2	Delay Settings for Synchronizing Scan Head and Laser Control	111
7.2.1	Laser Delays	111
	LaserOn Delay	111
	LaserOff Delay	112
7.2.2	Scanner Delays	113
	Jump Delay	113
	Variable Jump Delays	114
	Mark Delay	115
	Polygon Delay	117
	Variable Polygon Delay	118
	Customizing the Variable Polygon Delay	120
7.2.3	Notes on Optimizing the Delays	121
	Recommended Sequence	121
	Automatic Delay Adjustments	121
	Optimizing the Delays	124
7.2.4	Sky Writing	127
	Mode 1	127
	Mode 2	128
	Mode 3	128
	Synchronization	129
7.3	Scan Head Control	134
7.3.1	Reference System	134
7.3.2	Image Field Size and Calibration	134



Typical Image Field	135
RTC4 Compatibility Mode	135
7.3.3 Virtual Image Field	135
7.3.4 Image Field Correction and Correction Tables	136
Field Distortion	136
Field Correction Algorithm	137
Activating Image Field Correction	137
2D and 3D Correction Files	137
Loading Correction Tables	137
Assigning Loaded Correction Tables	138
1to1 Correction Tables	139
Inverse Tables	139
Correction File Header	140
Converting Correction Files	141
7.3.5 Output Values to the Scan System	141
Calculations	141
Value Ranges	142
Precalculation and Diagnosis	142
Clock Overruns	142
7.3.6 Status Monitoring and Diagnostics	143
Status Information Returned from the Scan System	143
7.4 Laser Control	144
7.4.1 Enabling, Activating and Switching Laser Control Signals	144
“Laser Active” Signals	145
“Laser Standby” Signals	145
Laser-Signal Auto-Suppression Triggered by Scan-System-Errors	146
Laser-Signal Auto-Suppression Triggered by Galvanometer Scanner Position Exceedances	
146	146
7.4.2 Configuring the Laser Connector (LASER)	146
7.4.3 CO ₂ Mode	147
7.4.4 YAG Modes	149
Q-Switch Signal	149
FirstPulseKiller Signal	150
Differences Between the YAG Modes	150
Lamp Current (Laser Power)	150
7.4.5 Softstart Mode	152
7.4.6 Laser Mode 4	154
7.4.7 Laser Mode 6	155
7.4.8 Pulse Picking Laser Mode	157
7.4.9 Automatic Laser Control	159
Position-Dependent Laser Control	161
Speed-Dependent Laser Control	165
Vector-Defined Laser Control	166
Encoder-Speed-Dependent Laser Control	168
7.4.10 Output Synchronization	169
7.5 Marking Dates, Times and Serial Numbers	170
7.5.1 Marking the Date and Time	170
7.5.2 Marking Serial Numbers	170
8 Advanced Functions for Scan Head and Laser Control	172
8.1 iDRIVE Functions	172
8.1.1 Transfer Protocol	172
8.1.2 Configuring Status Return Behavior	173



Selecting Data Signals	173
Querying Data	173
8.1.3 Position Monitoring	174
8.1.4 Configuring Dynamics Settings (Tuning)	176
8.1.5 Jump Mode	176
Functional Principle	176
Requirements and Activation	177
Jump-Length-Dependent Jump Delays	178
Experimental Determination of Jump Delay Values	178
Notes on Loading Determined Jump Delay Values	179
Automatic Determination of the Jump Delay Table	180
8.1.6 Configuring the PosAcknowledge Threshold Value	181
8.1.7 Configuring the Effective Calibration	181
8.1.8 Configuring the Start Behavior	182
8.1.9 Fault Diagnosis and Functional Test	182
8.2 Coordinate Transformations	183
8.3 Online Positioning	187
Configuring Online Positioning	187
8.4 Wobble Mode	189
8.4.1 Wobble Shapes – Important Notes on Choosing Appropriate Parameter Values	190
“Classic” Wobble Shapes	190
“Freely Definable Wobble Shapes”	191
8.5 Using Several Different Correction Tables	192
8.5.1 Configuration with Two 2D Scan Systems	192
8.5.2 Using Several Correction Tables with a Single Scan System	192
8.6 Controlling a 3-Axis Scan System (Optional)	193
8.6.1 Intended Use	193
8.6.2 Connection and Initialization	193
8.6.3 3D Marking Commands	194
8.6.4 Adjustment	196
Checking the Z-Axis Calibration	196
Test for 3-Axis Scan Systems with F-Theta Objective	197
8.6.5 Enhanced 3D Correction	198
8.7 Processing-on-the-fly (Optional)	199
8.7.1 Intended Use and Initialization	199
Overview	200
8.7.2 Compensation of Linear Movements	201
Correction by Encoder Counter(s)	201
Correction by McBSP/SPI Interface	202
Correction by McBSP/SPI Interface with Additional McBSP/SPI Input	203
Correction by McBSP/SPI Interface with Enhanced McBSP/SPI Input	203
8.7.3 Compensation of Rotary Movements	204
Correction by Encoder Counter	204
Correction by McBSP/SPI Interface	205
Correction by McBSP/SPI Interface with Additional McBSP/SPI Input	205
8.7.4 Compensation of 2D Motions	206
2D Encoder Compensation for XY Stages	206
Coordinate Transformations in the Virtual Image Field	208
8.7.5 Deactivating Processing-on-the-fly Corrections	209
8.7.6 Virtual Image Field	210
8.7.7 Synchronization of Processing-on-the-fly Applications	211
8.7.8 Encoder Resets	213



8.7.9	Monitoring Processing-on-the-fly Corrections	213
	Customer-Defined Monitoring Area and Conditional Command Execution (as of Version DLL 525, OUT 527)	214
8.7.10	Tracking Error Compensation of Encoder Values for Processing-on-the-fly Applications	215
8.7.11	Processing-on-the-fly Correction for the Z-Axis — “FlyZ Correction” (as of Version DLL 530, OUT 531)	215
8.8	Pixel Output Mode – Scanning Raster Images (Bitmaps)	217
8.8.1	Principle Of Operation	217
8.8.2	Software Commands	217
8.8.3	Laser Control	218
8.8.4	Timing	219
8.9	Microvector Commands	221
8.10	Timed Vector and Arc Commands	223
8.11	Automatic Self-Calibration	224
8.11.1	Using for Drift Compensation	224
8.11.2	How it Works	224
8.11.3	Determining Reference Values	226
8.11.4	Calibration During the Application	226
	Automatic Self-Calibration	226
	Customer-Specific Calibration	227
	Supplemental Information about Calibration	227
8.12	Camming	229
8.13	Time Measurements	231
9	Programming Peripheral Interfaces	232
9.1	Signal Output	232
9.1.1	16-Bit Digital Output Port	232
9.1.2	8-Bit Digital Output Port	233
9.1.3	2 Bit Digital Output Port	233
9.1.4	12-Bit Analog Output Ports	233
9.1.5	Stepper Motor Control	234
	Output Signals	234
	Reference Movements and Position Initialization	235
	Set-Position Movements	235
	Querying Signals and Status Values	236
	Terminating Infinite Movements	236
9.1.6	RS232 Interface	237
9.1.7	McBSP/SPI Interface	237
9.2	Signal Input	238
9.2.1	16-Bit Digital Input	238
9.2.2	2-Bit Digital Input	238
9.2.3	RS232 Interface	238
9.2.4	McBSP/SPI Interface	238
9.3	Control by External Signals	239
9.3.1	Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization	239
	External List Stop	239
	External List Start	240
	External List Start with Track Delay	241
	Regular (Periodic) External List Starts	242
9.3.2	Conditional Command Execution	244
	Programming Examples	245



9.3.3 Synchronization by Encoder Signals	246
Use	246
Inputs for External Encoder Signals	247
Encoder Simulation	247
9.3.4 Synchronization and Online Positioning by McBSP/SPI Signals	248
9.4 Periodical I/O Signals	249
10 Commands and Functions	250
10.1 Overview	250
10.1.1 Nomenclature	250
Multi-Board Commands	250
Normal, Short and Variable List Commands and List Multi-Commands	250
Undelayed and Delayed Short List Commands (as of Version OUT 515)	251
List Multi-Commands	252
10.1.2 Compatibility	252
10.1.3 Version Information	253
10.1.4 Optional Functions	253
10.1.5 Control Commands	254
10.1.6 List Commands	257
10.1.7 Data Types	260
10.2 RTC5 Command Set	261
acquire_RTC	261
activate_fly_2d	262
activate_fly_xy	263
apply_mcbsp	264
apply_mcbsp_list	265
arc_abs	266
arc_abs_3d	267
arc_rel	268
arc_rel_3d	269
auto_cal	270
auto_change	273
auto_change_pos	274
bounce_supp	275
camming	276
clear_fly_overflow	278
clear_io_cond_list	279
config_laser_signals	280
config_laser_signals_list	280
config_list	281
control_command	283
copy_dst_src	293
disable_laser	294
enable_laser	294
execute_at_pointer	295
execute_list	295
execute_list_1	295
execute_list_2	295
execute_list_pos	296
fly_return	297
fly_return_z	298
free_RTC_dll	299
get_auto_cal	299





get_char_pointer	300
get_config_list	300
get_counts	301
get_dll_version	301
get_encoder	301
get_error	302
get_fly_2d_offset	304
get_free_variable	304
get_galvo_controls	305
get_head_para	306
get_head_status	307
get_hex_version	308
get_hi_data	309
get_hi_pos	309
get_input_pointer	310
get_io_status	310
get_jump_table	311
get_lap_time	311
get_laser_pin_in	311
get_last_error	312
get_list_pointer	312
get_list_serial	313
get_list_space	313
get_marking_info	314
get_master_slave	316
get_mcbsp	317
get_mcbsp_list	317
get_out_pointer	318
get_overrun	318
get_RTC_mode	318
get_RTC_version	319
get_serial	319
get_serial_number	320
get_standby	320
get_startstop_info	321
get_status	323
get_stepper_status	324
get_sub_pointer	325
get_sync_status	326
get_table_para	327
get_text_table_pointer	328
get_time	328
get_transform	329
get_value	332
get_values	334
get_wait_status	335
get_waveform	336
get_z_distance	337
goto_xy	338
goto_xyz	339
home_position	340
home_position_xyz	341

if_cond	342
if_fly_x_overflow	342
if_fly_y_overflow	343
if_fly_z_overflow	343
if_not_activated	344
if_not_cond	344
if_not_fly_x_overflow	345
if_not_fly_y_overflow	345
if_not_fly_z_overflow	346
if_not_pin_cond	346
if_pin_cond	347
init_fly_2d	347
init_rtc5_dll	348
jump_abs	350
jump_abs_3d	351
jump_rel	352
jump_rel_3d	353
laser_on_list	354
laser_on_pulses_list	355
laser_signal_off	356
laser_signal_off_list	356
laser_signal_on	357
laser_signal_on_list	357
list_call	358
list_call_abs	359
list_call_abs_cond	360
list_call_abs_repeat	360
list_call_cond	361
list_call_repeat	362
list_continue	363
list_jump_cond	364
list_jump_pos	365
list_jump_pos_cond	366
list_jump_rel	367
list_jump_rel_cond	368
list_next	369
list_nop	369
list_repeat	370
list_return	371
list_until	372
load_auto_laser_control	373
load_char	374
load_correction_file	375
load_disk	379
load_fly_2d_table	381
load_jump_table	382
load_jump_table_offset	383
load_list	385
load_position_control	387
load_program_file	389
load_stretch_table	392
load_sub	393



load_text_table	394
load_varpolydelay	395
load_z_table	396
long_delay	397
mark_abs	398
mark_abs_3d	399
mark_char	400
mark_char_abs	401
mark_date	402
mark_date_abs	404
mark_ellipse_abs	405
mark_ellipse_rel	406
mark_rel	407
mark_rel_3d	408
mark_serial	409
mark_serial_abs	411
mark_text	412
mark_text_abs	413
mark_time	413
mark_time_abs	415
mcbsp_init	415
mcbsp_init_spi	416
measurement_status	417
micro_vector_abs	418
micro_vector_abs_3d	419
micro_vector_rel	420
micro_vector_rel_3d	421
move_to	422
para_jump_abs	422
para_jump_abs_3d	423
para_jump_rel	424
para_jump_rel_3d	425
para_laser_on_pulses_list	426
para_mark_abs	427
para_mark_abs_3d	428
para_mark_rel	429
para_mark_rel_3d	430
park_position	431
park_return	432
pause_list	434
periodic_toggle	434
periodic_toggle_list	435
quit_loop	436
range_checking	436
read_abc_from_file	438
read_analog_in	439
read_encoder	440
read_io_port	440
read_io_port_buffer	441
read_io_port_list	442
read_mcbsp	443
read_multi_mcbsp	444



read_status	445
read_user_data	447
release_RTC	448
release_wait	449
reset_error	449
restart_list	450
rs232_config	450
rs232_read_data	451
rs232_write_data	452
rs232_write_text	452
rs232_write_text_list	453
rtc5_count_cards	453
save_and_restart_timer	454
save_disk	455
select_char_set	456
select_cor_table	457
select_cor_table_list	459
select_RTC	460
select_serial_set	461
select_serial_set_list	461
send_user_data	461
set_angle	462
set_angle_list	463
set_auto_laser_control	464
set_auto_laser_params	467
set_auto_laser_params_list	467
set_char_pointer	468
set_char_table	469
set_control_mode	470
set_control_mode_list	472
set_default_pixel	472
set_default_pixel_list	472
set_defocus	473
set_defocus_list	474
set_delay_mode	475
set_delay_mode_list	476
set_dsp_mode	476
set_ellipse	477
set_encoder_speed	478
set_encoder_speed_ctrl	479
set_end_of_list	480
set_extstartpos	481
set_extstartpos_list	481
set_ext_start_delay	482
set_ext_start_delay_list	483
set_firstpulse_killer	483
set_firstpulse_killer_list	483
set_fly_2d	484
set_fly_limits	485
set_fly_limits_z	486
set_fly_rot	487
set_fly_rot_pos	488



set_fly_tracking_error	489
set_fly_x	490
set_fly_x_pos	491
set_fly_y	492
set_fly_y_pos	493
set_fly_z	494
set_free_variable	495
set_free_variable_list	495
set_hi	496
set_input_pointer	497
set_io_cond_list	497
set_jump_mode	498
set_jump_mode_list	501
set_jump_speed	502
set_jump_speed_ctrl	502
set_jump_table	503
set_laser_control	504
set_laser_delays	507
set_laser_mode	508
set_laser_off_default	508
set_laser_pin_out	509
set_laser_pin_out_list	509
set_laser_pulses	510
set_laser_pulses_ctrl	511
set_laser_timing	512
set_list_jump	513
set_mark_speed	514
set_mark_speed_ctrl	514
set_matrix	515
set_matrix_list	516
set_max_counts	517
set_mcbsp_freq	517
set_mcbsp_in	518
set_mcbsp_in_list	519
set_mcbsp_matrix	520
set_mcbsp_matrix_list	521
set_mcbsp_out	521
set_mcbsp_out_ptr	522
set_mcbsp_rot	523
set_mcbsp_rot_list	523
set_mcbsp_x	524
set_mcbsp_x_list	524
set_mcbsp_y	525
set_mcbsp_y_list	525
set_multi_mcbsp_in	526
set_multi_mcbsp_in_list	528
set_n_pixel	529
set_offset	530
set_offset_list	530
set_offset_xyz	531
set_offset_xyz_list	532
set_pause_list_cond	533



set_pixel	534
set_pixel_line	535
set_pixel_line_3d	536
set_port_default	537
set_pulse_picking	538
set_pulse_picking_length	538
set_pulse_picking_list	539
set_qswitch_delay	539
set_qswitch_delay_list	539
set_rot_center	540
set_rot_center_list	540
set_RTC4_mode	541
set_RTC5_mode	541
set_scale	542
set_scale_list	543
set_scanner_delays	544
set_serial	544
set_serial_step	545
set_serial_step_list	546
set_sky_writing	546
set_sky_writing_limit	547
set_sky_writing_limit_list	547
set_sky_writing_list	547
set_sky_writing_mode	548
set_sky_writing_mode_list	549
set_sky_writing_para	550
set_sky_writing_para_list	552
set_softstart_level	553
set_softstart_level_list	554
set_softstart_mode	555
set_softstart_mode_list	556
set_standby	557
set_standby_list	558
set_start_list	558
set_start_list_1	558
set_start_list_2	558
set_start_list_pos	559
set_sub_pointer	560
set_text_table_pointer	561
set_trigger	562
set_trigger4	568
set_vector_control	569
set_verify	571
set_wait	572
set_wobble	573
set_wobble_control	575
set_wobble_direction	576
set_wobble_mode	577
set_wobble_offset	579
set_wobble_vector	580
simulate_encoder	582
simulate_ext_start	583



simulate_ext_start_ctrl	584
simulate_ext_stop	584
start_loop	585
stepper_abs	586
stepper_abs_list	587
stepper_abs_no	587
stepper_abs_no_list	588
stepper_control	588
stepper_control_list	589
stepper_disable_switch	589
stepper_enable	590
stepper_enable_list	590
stepper_init	591
stepper_rel	593
stepper_rel_list	593
stepper_rel_no	594
stepper_rel_no_list	594
stepper_wait	595
stop_execution	596
stop_list	596
stop_trigger	597
store_encoder	597
sub_call	598
sub_call_abs	599
sub_call_abs_cond	599
sub_call_abs_repeat	600
sub_call_cond	600
sub_call_repeat	601
switch_iport	602
sync_slaves	603
time_fix	604
time_fix_f	604
time_fix_f_off	605
time_update	606
timed_arc_abs	607
timed_arc_rel	608
timed_jump_abs	609
timed_jump_abs_3d	610
timed_jump_rel	611
timed_jump_rel_3d	612
timed_mark_abs	613
timed_mark_abs_3d	614
timed_mark_rel	615
timed_mark_rel_3d	616
timed_para_jump_abs	617
timed_para_jump_abs_3d	619
timed_para_jump_rel	620
timed_para_jump_rel_3d	621
timed_para_mark_abs	622
timed_para_mark_abs_3d	624
timed_para_mark_rel	625
timed_para_mark_rel_3d	626



transform	627
upload_transform	630
verify_checksum	631
wait_for_encoder	631
wait_for_encoder_in_range	632
wait_for_encoder_mode	633
wait_for_mcbsp	634
write_8bit_port	634
write_8bit_port_list	635
write_abc_to_file	635
write_da_1	636
write_da_1_list	636
write_da_2	636
write_da_2_list	637
write_da_x	637
write_da_x_list	638
write_hi_pos	639
write_io_port	640
write_io_port_list	640
write_io_port_mask	641
write_io_port_mask_list	641
10.3 Unsupported RTC2/RTC3/RTC4 Commands	642
11 Demo Programs	647
12 Troubleshooting	648
13 Customer Service	650
13.1 Servicing and Repairs	650
13.2 Warranty	650
13.3 Contacting SCANLAB	650
13.4 Product Disposal	650
14 Technical Specifications of the RTC5 PCI Board	651
14.1 Compliance with EC Guidelines for Electromagnetic Compatibility (EMC)	654
14.2 Compliance with FCC Rules	654
15 Appendix A: The RTC5 PC/104-Plus Board	655
15.1 Product Overview	655
15.1.1 Intended Use – Comparison to the RTC5 PCI Board	655
15.1.2 System Requirements	655
Hardware	655
Software	656
15.1.3 Optional Functionality	656
15.1.4 Labeling	656
15.1.5 Type Identification	656
15.1.6 Unpacking Instructions and Typical Package Contents	657
Delivered Software	657
15.1.7 Optional Accessories	657
15.1.8 Supplementary Software	657
15.2 Layout and Interfaces	658
15.2.1 Connectors and Jumper Positions	658
15.2.2 Interface to the CPU board	660
15.2.3 Master/Slave Synchronization	660
15.2.4 Interfaces to Scan System	660



15.2.5 Interfaces for the Laser and Peripheral Equipment	661
LASER connector	661
MULTI connector	661
15.3 Installation and Start-Up	664
15.3.1 Safety During Installation and Operation	664
15.3.2 Jumper Settings	665
Jumper 17+18: Setting the Binary Stack Address	665
JP19: +5 V Power Supply	666
JP10+12: 10 V Output Voltage for ANALOG OUT1 and ANALOG OUT2	666
Jumper JP1: Selecting the Output Signal Level at the MULTI connector	667
Jumper JP2-JP8: Configuring Pin B40 and Pin B42 at the MULTI connector	667
15.3.3 Installing the RTC5 PC/104-Plus Board	668
15.3.4 Installing the Drivers and the RTC5 Software	670
15.3.5 Functionality Test, User Programs and Demo Software	670
15.4 Technical Specifications of RTC5 PC/104-Plus Board	671
16 Appendix B: The RTC5 PCI Express Board	672
16.1 Product Overview	672
16.1.1 Intended Use – Comparison to the RTC5 PCI Board	672
16.1.2 System Requirements	672
Hardware	672
Software	672
16.1.3 Optional Functionality	672
16.1.4 Labeling	673
16.1.5 Type Identification	673
16.1.6 Unpacking Instructions and Typical Package Contents	673
Delivered Software	673
16.1.7 Optional Accessories	673
16.1.8 Supplementary Software	673
16.2 Layout and Interfaces	674
16.2.1 Connectors and Jumper Positions	674
16.2.2 Interface to PC	675
16.2.3 McBSP/SPI Interface and Analog Inputs	676
McBSP/SPI Interface	676
Analog Inputs	676
16.3 Installation and Start-Up	676
16.4 Technical Specifications of the RTC5 PCI Express Board	677
16.5 Compliance with EC Guidelines for Electromagnetic Compatibility (EMC)	677
16.6 Compliance with FCC Rules	677
17 Appendix C: The RTC5 PCIe/104 Board	678
17.1 Product Overview	678
17.1.1 Intended Use – Comparison to the RTC5 PCI Board	678
17.1.2 Comparison to the RTC5 PC/104-Plus Board	678
17.1.3 System Requirements	679
Hardware	679
Software	679
17.1.4 Optional Functionality	679
17.1.5 Optional Connector	679
17.1.6 Labeling	679
17.1.7 Type Identification	680
17.1.8 Unpacking Instructions and Typical Package Contents	680
Delivered Software	680



17.1.9 Optional Accessories	680
17.1.10 Supplementary Software	680
17.2 Layout and Interfaces	681
17.2.1 Connectors and Jumper Positions	681
17.2.2 Interface to the CPU Board	682
17.2.3 Master/Slave Synchronization	682
17.2.4 Interfaces to Scan System	683
17.2.5 Interfaces for the Laser and Peripheral Equipment	683
17.3 Installation and Start-Up	683
17.4 Technical Specifications of RTC5 PCIe/104 Board	684
18 Revision History	685



1 Introduction

This operating manual describes the available SCANLAB RTC5 Boards and their usage for synchronous control of scan systems, lasers and peripheral equipment.

The main chapters of this manual use the RTC5 PCI Board to exemplify. For a product overview see [page 24](#).

The Appendix of this manual describes additional SCANLAB RTC5 hardware variants:

- The RTC5 PC/104-Plus Board in Appendix A on [page 655](#),
- The RTC5 PCI Express Board in Appendix B on [page 672](#),
- The RTC5 PCIe/104 Board in Appendix C on [page 678](#).

The manual is a part of the product. Read these instructions carefully before you proceed with installing and operating the RTC5 Board. In particular, observe all safety guidelines in this manual. If there are any questions regarding the contents of this manual, contact SCANLAB (see [chapter 1.1 "Manufacturer"](#)).

Keep the manual available for servicing, repairs and product disposal. This manual should accompany the product if ownership changes hands.

SCANLAB reserves the right to update this operating manual at any time and without notification.

This manual refers to the following versions of the RTC5 software and firmware:

- DLL file (`RTC5DLL.dll`) version 514 or higher,
- DSP program file (`RTC5OUT.out`) version 513 or higher,
- RTC5 firmware file (`RTC5RBF.rbf`) version 511 or higher.

The version numbers of the supplied DLL and program files are indicated in the names of the corresponding zip files (see [page 23](#)).

After installation, the commands `get_dll_version`, `get_hex_version` and `get_RTC_version` can be used to query the current software and firmware version numbers.

1.1 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

1.2 Labeling

The serial number of the RTC5 PCI Board is printed on a white label attached to the board (format: "RTC SN...").

The ID number and configuration of the board are described in the packaging list (see [page 27](#) and [page 28](#)).



1.3 Unpacking Instructions and Typical Package Contents

- ▶ Carefully remove the RTC5 Board from the package.
- ▶ Keep the packaging, including the antistatic bag the RTC5 Board is delivered in, so that in case of repair the board can be properly repackaged and returned to SCANLAB.
- ▶ Also remove all other articles from the package. Check that all parts have been delivered. Refer to the corresponding packaging list.
An RTC5 package typically includes an RTC5 Board and a data CD (containing the corresponding software (see below) and this manual). Some product versions may also supply additional components such as data cables or converters, see [chapter 2.5 "Accessories", page 29](#).

1.3.1 Delivered Software

The delivered RTC5 software package contains drivers for the 32-bit and 64-bit versions of the operating systems Microsoft Windows 10, 8, 7, Vista, XP SP2, XP SP3.

The data CD contains all files as unzipped versions. For easy identification and archiving of different software versions (see also [section "Revision History", page 23](#)), the complete software package is also delivered zipped:

- RTC5_Software_<Date>.zip

All files of the software package are listed in the following.

Description of Content and Installation

- Readme.txt
[Description in English](#)
- Liesmich.txt
[Description in German](#)

Correction Files

- cor_1to1.CT5
correction file
- usually more correction files (D2_XXX.CT5, D3_YYY.CT5) and Readme files

CorrectionFileConverter Program

- The Win32-based program
CorrectionFileConverter.exe converts RTC4 correction files into RTC5 correction files and vice versa. The package also contains the corresponding operating manuals in German and English.

Demo Files

- Source codes in C of Demo1...Demo7
(Demo1.cpp...Demo7.cpp)
- Demo1 compiled to executable files for 32-bit as well as 64-bit
- Source code in C# of Demo3 (Class1.cs)
- Project generating file for CMake
(CMakeLists.txt) and the corresponding include file (RTC_Variables.cmake)⁽¹⁾

HPGL Converter Program

- Win32-based HPGL demo programs.
Needs program files and the 32-bit DLL in the same directory.

iSCANConfig Program

- The program iSCANcfg.exe is a Win32-based diagnosis and configuration program for iDRIVE scan systems (intelliSCAN, intelliSCAN_{de}, intelliDRILL, intellicube, intelliWELD, varioSCAN_{de}). Needs program files and the 32-bit DLL in the same directory. The corresponding manuals in German and English are contained as well.

(1) These are CMake (cross-platform make) files to easily generate executable demo programs. CMake (<https://cmake.org>) is a free and open-source cross-platform programming tool for the development and creating software. Using Cmake, make files and projects for many integrated development environments and compilers can be generated by script files (CMakeLists.txt).



Windows Drivers

- RTC5DRV.sys, RTC5DRVx64.sys, RTC5DRVx86.sys
RTC5 driver files
- RTC5DRV.inf
Installation file (setup information)
- RTC5DRV.cat, rtc5drv64.cat, rtc5drv86.cat
Security catalog files
- amd64/WdfCoInstaller01009.dll,
x86/WdfCoInstaller01009.dll
Installation assistant help files
- AfterInstallation/ScanlabClassChecker.cmd,
Security installation script with description in
ReadMe_ScanlabClassChecker.pdf

RTC5 Files

- **DLL Files:**
 - RTC5DLL.dll, RTC5DLLx64.dll
Win32- and Win64-based RTC5 dynamic link library
- **Program Files:**
 - RTC5OUT.out
DSP program file
 - RTC5RBF.rbf
Firmware file
 - RTC5DAT.dat
Binary support file
- **Utility Files for C, C++ and C#:**
 - RTC5DLL.lib, RTC5DLLx64.lib
Visual C++ import libraries for implicit linking of the DLL for Win32-based and Win64-based applications
 - RTC5expl.c
C functions for DLL handling for explicit linking
 - RTC5expl.h
C function prototypes of the RTC5 for explicit linking of the DLL
 - RTC5impl.h
C function prototypes of the RTC5 for implicit linking of the DLL
 - RTC5impl.hpp
C++ function prototypes of the RTC5 for implicit linking of the DLL
 - RTC5Wrap.cs
Import declarations of the wrapper class for implicit linking in C#
- **Utility File for Delphi:**
 - RTC5Import.pas
Import declarations for Delphi

For easy identifying and archiving of different software versions, the RTC5 files are also delivered zipped:

- RTC5DAT_<Version>.zip
- RTC5DLL_<Version>.zip
(includes DLL and utility files)
- RTC5OUT_<Version>.zip
- RTC5RBF_<Version>.zip

Differing versions of the program files and DLL cannot be arbitrarily combined with another. Each zip file includes a text file with corresponding version information.

Revision History

Description of changes in the RTC5 software

- RTC5_Software_RevisionHistory_<Date>.pdf
Description in English
- RTC5_Software_Aenderungshistorie_<Date>.pdf
Description in German

Notes

- Older RTC5 software packages additionally include MS Visual Studio run time libraries for C and C++. As of version DLL 527, these libraries are no longer necessary.



2 Product Overview

2.1 Intended Use

The SCANLAB RTC5 PCI Board and its associated software allow synchronous real-time control of scan systems, lasers and peripheral equipment by a Windows PC with a PCI bus interface. The included driver provides an extensive set of control commands that enable quick and flexible programming of laser scan processes.

The RTC5 features a fast digital signal processor (DSP) system whose control commands can also handle more complex signal processing and computational steps, such as simultaneous control of two scan systems or coordinate transformations. Moreover, you can store control commands on the RTC5 and start their execution at a later time. Command execution by the RTC5 can then take place independently of the host PC. This makes it possible to meet the stringent demands of real-time control for scan systems, lasers and peripheral equipment even if the PC must simultaneously respond to other tasks such as machine control and network communication.

The interface to the scan system, together with the associated software commands, allows bidirectional communication with the scan system, thereby providing both control and monitoring capabilities for the scan system.

To control lasers, the RTC5 supplies interfaces that output laser control signals and are software-configurable for each application's requirements. For example, users can select among seven different laser control modes and such parameters as the logical signal level (active-high or active-low) or the output frequency. With the RTC5, all commonly used laser types can be controlled.

For controlling peripheral equipment and incorporating external control signals, the RTC5 provides a range of interfaces (e.g. a 16-bit digital input port, a 16-bit digital output port, two 12-bit analog output ports and an RS232 interface) and associated software commands.

Any number of RTC5 Boards can be used simultaneously (master/slave synchronized, if needed) in one PC.

Moreover, the RTC5's DLL allows multi-threading as well as multi-processing; therefore any number of applications (user programs) can be used simultaneously. However, no board can be simultaneously used by multiple applications. Multiple threads of one user program can use the same board, but can not send commands to it at the same time.

The RTC5 PCI Board is available in various configurations, see [chapter 2.3 "Optional Functionality", page 27](#) and [chapter 2.4 "Jumper Settings and Type Identification", page 28](#).

The RTC5 Board interfaces are described on [page 40](#), installation and start-up on [page 60](#), and programming on [page 65](#). Individual command descriptions are listed beginning with [page 250](#).

The technical specifications of the RTC5 PCI Board are summarized on [page 651](#).



Danger!

- Do not operate the RTC5 Board outside of the PC.
- The RTC5 Board is intended only for industrial usage. It is designed to be incorporated in machines (normally laser systems). It does **not** meet all criteria of ready-to-use products. Do not operate the RTC5 Board unless it is incorporated in a machine which itself complies with the regulations of all applicable standards and directives (of the country concerned). It is **not** suitable to be used as toy, in household or under unfavourable environment conditions (e.g. in the open). Appropriate precautions to avoid such unforeseeable misapplications must be taken by users.
- Installation and operation must only be performed by trained specialists, among other things knowledgeable in the safe and proper use of electrical devices. Only carry out installation and maintenance work when supply voltages and lasers have been switched off.
- The RTC5 Board is a class A product. In a domestic environment this product may cause radio interferences in which case the user may be required to take adequate measures.

2.2 System Requirements

2.2.1 Hardware

The RTC5 PCI Board requires a Windows PC with a PCI bus interface and at least one free PCI slot.

The dimensions are shown in [figure 2](#).

RTC5 PCI Boards intended for synchronized master/slave operation must be installed in adjacent PCI slots.

2.2.2 Software

The RTC5 Board is delivered with a software package including drivers and DLL files for Microsoft Windows operating systems. The RTC5 drivers support the plug and play capability of the RTC5 and simultaneously drive any number of RTC5 Boards. RTC5 DLL files contain the functions which can be called by RTC5 commands.

- RTC5 drivers and RTC5 DLL files included in RTC5 software packages of version 2013_02_21 and later (with version \geq DLL 535) are designed for 32-bit as well as 64-bit versions of Microsoft Windows 10, 8, 7, Vista, XP SP2, XP SP3. RTC5 DLL files from these packages do not support Microsoft Windows 2000 and XP \leq SP1.
- RTC5 drivers and RTC5 DLL files included in RTC5 software packages of version 2011_09_29 through 2012_09_05 (with version DLL 528...DLL 535) are designed for Microsoft 32-bit as well as 64-bit versions of Windows 7, Vista, XP SP2, XP SP3. RTC5 DLL files from these packages do not support Microsoft Windows 2000 and XP \leq SP1.
- RTC5 drivers and RTC5 DLL files included in RTC5 software packages of version 2011_07_27 and earlier (with version \leq DLL 527; they have less functionality) are designed for 32-bit versions of Microsoft Windows 7, Vista, XP and 2000.



Caution!

- The RTC5 PCI Board does not support power-saving modes, that switch off power to the PCI bus. Accordingly, you must disable standby or sleep modes of the operating system (see also note below).

Notes

- RTC5 software package version 2013_02_21 and later contain (newer) WDF⁽¹⁾ drivers version 6.1.7600.16385. Earlier packages contain (older) WDM⁽²⁾ drivers of version 1.0.4.0 or 2.0.6.0.
- RTC5 DLL files version DLL \leq 533 are *not* compatible with the WDF drivers.
- RTC5 DLL files version DLL \geq 535 are compatible with the WDM drivers.
- In comparison to old WDM drivers, the new WDF driver offers the following new functionality: If `init_rtc5_dll` is called, then the driver prevents automatic activation of standby or sleep modes (continuously until the next system restart). This enables RTC5 Boards to continue processing lists autonomously even when the initiating program has already terminated.

However, standby or sleep modes cannot be prevented if triggered manually or by discharged batteries. Afterward, loaded lists and other settings of the RTC5 Board are lost. After a wake-up, the board might no longer be correctly addressable. Users themselves must prevent standby or sleep modes prior to the first call of `init_rtc5_dll`.

(1) Windows Driver Framework

(2) Windows Driver Model

2.3 Optional Functionality

The RTC5 can be configured for functionality not available in the standard version. This extended functionality must be enabled by SCANLAB or installed. The following options are available:

- **Processing-on-the-fly**

If the Processing-on-the-fly option is enabled, Processing-on-the-fly corrections can be activated (see [page 199](#)). Otherwise, the associated commands are partially or fully unusable.

- **Controlling a 3-Axis Scan System**

If the RTC5's 3D option is enabled, then an RTC5 Board can synchronously control a third axis (Z-axis, e.g. a varioSCAN as a dynamic focus unit) along with the scan system's X and Y axes (usually two galvanometer scanners) by its two scan head connectors (see [page 193](#)). Otherwise, the associated 3D commands are partially or fully unusable.

- **Synchronous Control of Two Scan Systems**

If the RTC5's "second scan head control" option is enabled, then an RTC5 Board can simultaneously control two XY scan systems by its two scan head connectors (see also [page 42](#) and [page 192](#)).

- **Optoelectronic Couplers**

RTC5 Boards with this option are equipped with additional DC/DC converters in the factory. Therefore, the laser signals LASERON, LASER1 and LASER2 at the LASER connector and EXTENSION 2 socket connector are galvanically decoupled from the PC ground (see [page 47](#) and [page 52](#)).

Notes

- Each RTC5's ID number indicates which options are enabled and/or installed. Enabled or installed options are stated in the packing list as follows:
 - "Fly" (Processing-on-the-fly option)
 - "3D" (3D option)
 - "SSHC" ("second scan head control" option)
 - "DCDC" (optoelectronic couplers)
- The `get_rtc_version` command can be used for querying whether the Processing-on-the-fly, the 3D and/or the "second scan head control" option are enabled.
- If you need one or more of the Processing-on-the-fly, "3D" and/or "second scan head control" options and they are not already activated on your board, then you can activate them by special upgrade software available from SCANLAB. When requesting the upgrade, you will need to supply the serial number of your RTC5 Board.
If, on the other hand, you want optoelectronic couplers installed on your RTC5, then you will need to send the board to SCANLAB.
- For optical data transfer between the RTC5 and the scan system (by a polymer optical fiber), an appropriate data cable is required. But special (optical) data interfaces are not required – neither for the RTC5 nor for the scan system (see [page 45](#)).

2.4 Jumper Settings and Type Identification

SCANLAB ships RTC5s in various jumper configurations. The jumpers **JP1-JP8** are soldered junctions. At a later date, users can reconfigure them by using a soldering iron.



Caution!

- When altering a jumper configuration, take care not to damage the board's electronics!

2.4.1 Jumper JP1: Selecting the Output Signal Level at the EXTENSION 1 Socket Connector

By Jumper JP1 on the back side of the RTC5, see [figure 3](#), the level of all output signals at the EXTENSION 1 socket connector can be configured for 5 V or 3.3 V.

See also "[Configuring the Output Signal Level](#)", Seite 51.

Jumper JP1	Signal Level
Position 1-2	
Position 2-3	
open	



Caution!

- Be sure that not all three jumper pins are closed. Otherwise, the electronics of the board is damaged.

2.4.2 Jumper JP2-JP8: Configuring Pin15 and Pin17 at the EXTENSION 2 Socket Connector

By the jumpers **JP2-JP8** on the back side of the RTC5, see [figure 3](#), pins (15) and (17) of the EXTENSION 2 socket connector can be configured. See also "["Jumper Setting"](#)", Seite 52).

The most significant bit (DATA7) of the 8-bit output value can be assigned to pin (15) or to pin (17). Alternatively, each of the two pins can be set permanently to +5 V (HIGH) or to GND (LOW level). Alternatively, pin (17) can be configured for the LATCH signal by closing the correspondingly labeled jumper.

[Figure 1](#) shows an example jumper configuration assigning DATA7 to pin (15) and the LATCH signal to pin (17).

Pin17	Pin15
(JP6) Data7	
(JP5) +5V	
(JP7) GND	
(JP8) Latch	

1

Example configuration for jumpers JP2-JP8:
Pin (15): DATA7, Pin (17): LATCH signal



Caution!

- Make sure that not more than **one** of the three jumpers for pin (15) is closed. Also make sure that not more than **one** of the four jumpers for pin (17) is closed.
Closing more than one jumper for pin (15) or (17) damages the electronics of the board.

2.4.3 Type Identification

The RTC5's ID number reveals the board's factory-equipped jumper configuration. The jumper configuration is additionally encoded in a three-digit type code scheme:

Digit 1	=0: Jumper JP1 open (no signal) =1: Jumper JP1 in position 1-2 (5 V) =2: Jumper JP1 in position 2-3 (3.3 V)
Digit 2 (relates to pin 15)	=0: Jumper JP2-4 open (no signal) =1: Jumper JP2 closed (+5 V) =2: Jumper JP3 closed (DATA7) =3: Jumper JP4 closed (GND)
Digit 3 (relates to pin 17)	=0: Jumper JP5-8 open (no signal) =1: Jumper JP5 closed (+5 V) =2: Jumper JP6 closed (DATA7) =3: Jumper JP7 closed (GND) =4: Jumper JP8 closed (LATCH)

Examples

- "Type 000": all jumpers open
- "Type 124": **JP1** in position 1-2 (5 V signal level), **JP3** closed (DATA7 at pin 15), **JP8** closed (LATCH at pin 17)

2.5 Accessories

In addition to the RTC5 PCI Board and its software package, the following accessories can be obtained from SCANLAB (only hardware extensions from SCANLAB should be used in combination with the RTC5):

2.5.1 XY2-100 Converter

SCANLAB's XY2-100 converter allows the RTC5 Board to control scan systems equipped with a conventional XY2-100 interface (see [page 43](#)).

2.5.2 Data Cables

To connect the RTC5 Board to scan systems, SCANLAB offers appropriate cables in a variety of lengths – either conventional cables for electrical data transfer or polymer optical fiber cables or optical data transfer (see [page 45](#)).

2.5.3 Laser Adapter

SCANLAB offers an laser adapter which can be plugged into the 15-pin LASER connector of the RTC5. This adapter's 9-pin female D-SUB connector provides the same signals and pin-out as the RTC4's 9-pin laser connector (see [page 50](#)).

2.5.4 PCI Slot Covers

For connecting a second scan head or a Z axis to the secondary scan head connector (10-pin 2. SCAN HEAD socket connector, see [page 43](#)), a suitable slot cover with a 9-pin D-SUB connector is available from SCANLAB (see [page 43](#)).

For using the inputs and signals of the MARKING ON THE FLY socket connector (16-pin socket connector, see [page 53](#)), a suitable slot cover with a 15-pin D-SUB connector is available from SCANLAB (see [page 53](#)).



2.5.5 ADC Add-On Board

SCANLAB offers an ADC add-on board (#121126) for the RTC5 PCI Board – but *not* for the RTC5 PCI Express Board, RTC5 PC/104-Plus Board and RTC5 PCIe/104 Board.

With installed add-on board, the RTC5 PCI Board provides two 10 V analog inputs (see [page 59](#)).

The RTC5 PCI Express Board provides two 10 V analog inputs even without any add-on board (see [page 676](#)).

2.5.6 RTC5 varioSCAN FLEX Extension Board

For RTC5 Boards, SCANLAB offers the extension board RTC5 varioSCAN 40 FLEX Extension (#128683)⁽¹⁾.

It has been specially developed to control analog and digital varioSCAN 40 FLEX devices by the SCANLAB laserDESK software. A position change of the varioSCAN 40 FLEX focusing optics is caused by the step motor which changes the working distance of the 3D scan system in the end.

For further information, refer to the pertaining manual “Installation and Operation RTC5 varioSCAN FLEX Extension for the RTC5 PC Interface Board”.

Notes

- The extension board RTC5 varioSCAN 40 FLEX Extension (#128683) does *not* support the command [**move_to**](#)⁽²⁾.

2.6 Supplementary Software

To facilitate customizing RTC correction files basing on your own test measurements, SCANLAB offers its correXion line of software and related documentation (see also [page 137](#)).

By using the SCANLAB laserDESK software, own laser marking and material-processing programs can be created and executed without software development. Almost any RTC5 functionality is accessible by the graphical user interface. SCANLAB laserDESK integrates SCANalign vision solution to enable use and management of camera-assisted position correction and quality control directly in laser jobs.

(1) In contrast to the RTC Step Motor Extension board (#112097) – which can also be used with RTC5 Boards – it has the advantage that the EXTENSION 1 socket connector remains unoccupied.

(2) The command [**move_to**](#) has been introduced for the RTC Step Motor Extension board (#112097).



2.7 Notes for RTC4 Users

This chapter provides an overview of key changes introduced by the RTC5 PCI Board in comparison to the RTC4 PCI Board.

For example, [chapter 2.7.2 "Porting RTC4 Programs to the RTC5", page 32](#) discusses a possible approach to porting RTC4 programs to run on the RTC5.

Other chapters of this manual also contain various sections with tips for RTC5 users who previously used the RTC4. The individual command descriptions in particular note relevant changes.

2.7.1 Hardware Changes

When migrating from the RTC4 to the RTC5, you need to consider the following hardware changes for correct cabling of the system components.

Scan System Control

- To control a scan system with the XY2-100 or XY2-100 Enhanced interface, you also need an XY2-100 converter (available from SCANLAB – see [page 43](#)).
- The RTC5 cannot control scan systems with XY2-100-O interfaces (for optical data transfer).
- To control a scan system with the SL2-100 interface, you need a different data cable (available from SCANLAB – see [page 45](#)).
- To use the secondary scan head connector, you need a different adapter cable (including slot cover available from SCANLAB – see [page 43](#)).
- For controlling a 3-axis scan system, both scan head connectors must be used (and a 3D correction file must be correctly assigned) (see [page 193](#)).
- Simultaneous control of two 3-axis scan systems requires two RTC5 Boards (see [page 193](#)).

Laser Control

- The female D-SUB laser connector at the RTC5 slot cover has 15 pins (the RTC4's laser connector, on the other hand, has 9 pins). The pinouts are not jumper-configurable.
 - The RTC5 does not require jumpers X6 and X7 of the RTC4 because all signals are already available at the RTC5 laser connector.
 - The voltage range of the analog outputs is always 0...10 V (0 V ... 2.50 V is no longer supported; the RTC4's jumper X3 does not exist on the RTC5).
- If you want to connect a laser to the RTC5 by the same (9-pin) cable that you previously used with the RTC4, then you need an adapter with a 9-pin female D-SUB connector (available from SCANLAB).
 - For use of the laser adapter from SCANLAB (see [page 50](#)), two jumpers are provided for configuring the pin-outs (JP1 corresponds to jumper X7 of the RTC4, JP2 corresponds to jumper X6 of the RTC4).
- The signal levels of the laser control signals are no longer determined by configuring jumpers. Instead, they can/must be software-configured (see [set_laser_control](#)).
 - Jumper X10 of the RTC4 does not exist on the RTC5.

EXTENSION 1 Socket Connector

- The RTC5 EXTENSION 1 socket connector is – except for the additionally provided signals at pins 33-35 – identical to the EXTENSION 1 socket connector of the RTC4 (see [page 51](#)).
- With the RTC5 (unlike the RTC4), the level of all output signals at the EXTENSION 1 socket connector can be configured for 5 V or 3.3 V by a jumper (see [page 51](#)).

EXTENSION 2 Socket Connector

- The RTC4 EXTENSION 2 socket connector does not exist on the RTC5. Accordingly, no I/O extension board can be attached to the RTC5.
- The RTC5 EXTENSION 2 socket connector is – except for the optional LATCH signal at pin 17 – identical to the LASER EXTENSION socket connector of the RTC4 (see [page 52](#)).
- The RTC5 provides jumpers JP2-JP8 for configuring pin-outs (these jumpers are equivalent to jumpers X8 and X9 of the RTC4) (see also [page 52](#)).

MARKING ON THE FLY Socket Connector

- The RTC5 MARKING ON THE FLY socket connector is identical to the RTC4 MARKING ON THE FLY socket connector (see [page 53](#)).

Other Interfaces

- PCI bus requirements are identical to those of the RTC4.
- PCI-Express, PC/104-Plus and PCIe/104 versions are only available for the RTC5; a standalone and an Ethernet version is currently only available for the RTC4.
- The following interfaces only exist on the RTC5:
 - MASTER and SLAVE (see [page 41](#))
 - RS232 (see [page 54](#))
 - SPI / I²C / McBSP (see [page 54](#))
 - STEPPER MOTOR (see [page 58](#))

2.7.2 Porting RTC4 Programs to the RTC5

User programs written for the RTC4 can only run on the RTC5 after suitable code revision. This applies even when the actual program flow shall remain unchanged and none of the RTC5's new functionality is being accessed.

Changed Initialization

The program's initialization section should be revised at least as follows:

- At the beginning of the program, a `init_rtc5_dll` command must be inserted for initializing the DLL and board management (see [page 68](#)).
- The files for initializing the board by `load_program_file` must be supplied and called differently than with the RTC4 (see command description).
- Scan system initialization by `load_correction_file` and `select_cor_table` utilizes different correction files (with file extension *.ct5) (see [page 136](#)).
- For laser control initialization, the `set_laser_control` command must be additionally inserted (see [page 144](#)), even when only standby signals are to be outputted.

Command Changes

All unsupported RTC4 commands must be removed or replaced (see also [chapter 10.3 "Unsupported RTC2/RTC3/RTC4 Commands"](#), page 642).

Changed or enhanced RTC4 commands might need to be handled differently in the program (e.g. by modifying supplied parameter values or evaluating returned values differently). Relevant changes to supported commands are listed in the individual command descriptions (in [chapter 10.2](#)) under the heading "RTC4→ RTC5".

Below is a list of RTC4 commands that need to be replaced or checked:

<code>aut_change</code>	not supported
<code>auto_cal</code>	changed
<code>auto_change_pos</code>	changed
<code>control_command</code>	changed
<code>dsp_start</code>	not supported
<code>get_head_status</code>	changed
<code>get_hi_data</code>	changed
<code>get_list_space</code>	changed
<code>get_marking_info</code>	changed
<code>get_RTC_version</code>	changed
<code>get_startstop_info</code>	changed
<code>get_status</code>	changed
<code>get_value</code>	changed
<code>get_waveform</code>	changed
<code>get_xy_pos</code>	not supported
<code>get_xyz_pos</code>	not supported
<code>goto_xy</code>	changed
<code>goto_xyz</code>	changed
<code>list_jump_cond</code>	changed (depending on the memory configuration)
<code>list_nop</code>	changed
<code>load_cor</code>	not supported
<code>load_correction_file</code>	changed
<code>load_pro</code>	not supported
<code>load_program_file</code>	changed
<code>read_pixel_ad</code>	not supported
<code>read_status</code>	changed
<code>rtc3_count_cards / rtc4_count_cards</code>	not supported
<code>select_cor_table</code>	changed
<code>select_list</code>	not supported
<code>select_rtc</code>	changed
<code>set_control_mode</code>	enhanced
<code>set_control_mode_list</code>	enhanced
<code>set_laser_mode</code>	enhanced
<code>set_laser_timing</code>	Firmware changed
<code>set_list_mode</code>	not supported
<code>set_matrix</code>	changed
<code>set_matrix_list</code>	changed
<code>set_offset</code>	changed
<code>set_offset_list</code>	changed
<code>set_piso_control</code>	not supported
<code>set_pixel</code>	changed
<code>set_pixel_line</code>	changed
<code>set_softstart_mode</code>	changed
<code>set_trigger</code>	enhanced
<code>set_wobble</code>	changed
<code>set_wobble_xy</code>	not supported

- `timed_jump_abs` changed
- `timed_jump_rel` changed
- `timed_mark_abs` changed
- `timed_mark_rel` changed
- `z_out` not supported
- `z_out_list` not supported

Increased Parameter Resolution

When switching from the RTC4 to the RTC5 – even for some commands not mentioned above – take note that the resolution has been increased for several parameters. Examples:

- For commands such as `mark_abs` or `jump_rel`, the real-image-field x and y coordinate values are specified with 20-bit resolution for the RTC5 (whereas with 16-bit resolution for the RTC4). Therefore, 16 times larger parameter values must be specified for a certain position (in mm) – see also [chapter 7.3.2, page 134](#). Moreover, an extended, virtual value range – altogether a virtual 24-bit image field – is available with the RTC5 (particularly for Processing-on-the-fly applications) – see also [chapter 7.3.3, page 135](#).
- For commands such as `write_da_x`, analog output values are specified with 12-bit resolution for the RTC5 (whereas with 10-bit resolution for the RTC4). Therefore, a 4 times larger parameter value must be specified for a certain output value (in V).
- For the command `set_laser_timing`, output period and pulse length are specified with 1/64 µs resolution for the RTC5 (whereas with 1/8 µs or 1 µs resolution for the RTC4). Therefore, an 8 times or 64 times larger parameter value must be specified for a certain timing value (in µs). For the clock frequency refer to the command description.
- For the command `set_laser_delays`, the laser delays are specified with 0.5 µs resolution for the RTC5 (whereas with 1 µs resolution for the RTC4). Therefore, a 2 times larger parameter value must be specified for a certain delay value (in µs).

Here, though, it generally suffices to set the RTC5 DLL to RTC4 compatibility mode by `set_RTC4_mode`. Then the RTC5 DLL automatically converts parameter values so that many RTC4 command sequences (e.g. for defining marking patterns) can run unchanged on the RTC5.



RTC4 compatibility mode affects the following RTC4 commands (descriptions of the respective commands might include relevant information under the heading "RTC4→ RTC5"):

- `arc_abs`
- `arc_rel`
- `fly_return`
- `get_z_distance`
- `goto_xy` (changed)
- `goto_xyz` (changed)
- `home_position`
- `jump_abs`
- `jump_abs_3d`
- `jump_rel`
- `jump_rel_3d`
- `mark_abs`
- `mark_abs_3d`
- `mark_rel`
- `mark_rel_3d`
- `set_delay_mode`
- `set_ext_start_delay`
- `set_ext_start_delay_list`
- `set_firstpulse_killer`
- `set_firstpulse_killer_list`
- `set_fly_x`
- `set_fly_y`
- `set_jump_speed`
- `set_laser_delays`
- `set_mark_speed`
- `set_pixel` (changed)
- `set_pixel_line` (changed)
- `set_rot_center`
- `set_softstart_level`
- `set_standby`
- `set_standby_list`
- `simulate_ext_start`
- `timed_jump_abs` (changed)
- `timed_jump_rel` (changed)
- `timed_mark_abs` (changed)
- `timed_mark_rel` (changed)
- `write_da_1`
- `write_da_1_list`
- `write_da_2`
- `write_da_2_list`
- `write_da_x`
- `write_da_x_list`

Nevertheless, the previously mentioned revision of initialization and checking of unsupported or changed RTC4 commands needs to be carried out regardless of whether the program is to execute in RTC5 mode or RTC4 compatibility mode.

Changed Timing Behavior

The following RTC4 list commands are processed by the RTC5 as "short list commands". This can result in a changed timing behavior during execution (see [page 250](#)).

- `clear_io_cond_list`
- `list_call`
- `list_call_cond`
- `list_jump_cond` (changed, depending on the memory configuration)
- `list_return`
- `save_and_restart_timer`
- `set_extstartpos_list`
- `set_firstpulse_killer_list`
- `set_io_cond_list`
- `set_jump_speed`
- `set_laser_delays`
- `set_laser_timing` (Firmware changed)
- `set_list_jump`
- `set_mark_speed`
- `set_scanner_delays`
- `set_standby_list`
- `set_trigger` (enhanced)
- `set_wobbel` (changed)
- `write_8bit_port_list`
- `write_da_1_list`
- `write_da_2_list`
- `write_da_x_list`
- `write_io_port_list`

Likewise, automatic delay adjustments can produce a changed timing behavior (see [page 121](#)).

2.7.3 New and Changed Functionality

Interface to the PC

- The RTC5 drivers supports simultaneous control of any number of RTC5 Boards in one PC (see [page 92](#)).
- Connectors and commands for master/slave synchronization of multiple RTC5 Boards (see [page 41](#))

Scan System Control

- New interface to the scan system (see [page 42](#)):
 - 9-pin female D-SUB connector at the RTC5 slot cover and on-board 10-pin socket connector
 - SL2-100 transfer protocol
 - 2 data channels each for both scan head connectors
 - Galvanically isolated signals
 - 20-bit positioning resolution (see [page 134](#))
 - Enhanced status return from the scan system (see [page 143](#))
 - Control and status channels for enhanced data transfer with iDRIVE scan systems (intelliSCAN, intelliSCAN_{de}, intelliDRILL, intellicube, intelliWELD, varioSCAN_{de})
- An XY2-100 converter is provided for data transfer according to the XY2-100 protocol (see [page 43](#)). The RTC5 provides power for this converter.
 - 25-pin female D-SUB connector
 - 16-bit positioning resolution
 - Status return according XY2-100 or XY2-100 Enhanced protocol
 - Transfer synchronization is configurable for long data cables by solder jumpers in the XY2-100 converter
- For optical data transfer between the RTC5 and scan systems, no special variant of the RTC5 (with XY2-100-O interface) is required. Optical data transfer can be realized by a SCANLAB data cable with electrical-to-optical conversion in its D-SUB connector housing (see [page 45](#)).

- For controlling a 3-axis scan system (see [page 193](#)):
 - Both scan head connectors must be used and a 3D correction file must be correctly assigned.
 - Simultaneous control of two 3-axis scan systems requires two RTC5 Boards.
- Image field correction
 - New correction files are needed (see [page 136](#)):
 - > File name extension ".ct5"
 - > Correction with higher resolution
 - > Correction data information in the file header is queryable
 - > RTC2/3/4 correction files (.ctb) need to be newly calculated or converted by a program supplied with the RTC5.
 - Up to four correction files can be loaded to the RTC5 Board.
 - Enhanced 3D image field correction by stretch correction tables (see [page 198](#))
- Coordinate transformations (see [page 183](#)):
 - The correction file is no longer transformed (rotation, shift, extension) at download
 - Matrix transformations are only applied after microvectorization – this may cause the marking speed to change.
 - 24-bit vector coordinates: objects larger than the image field are possible (virtual processing field, see [page 135](#))
 - Collection of transformation definitions and execution together with a following command
 - Online positioning (see [page 187](#))
- Position monitoring of iDRIVE scan systems by backward transformation of actual position values (see [page 174](#))
- Automatic self-calibration (see [page 224](#)):
 - Optimization of previous functions
 - ASC hardware check
- Jump mode (see [page 176](#))
- Output synchronization (see [page 169](#))

Laser Control

- The signal levels of the laser control signals are no longer determined by configuring jumpers. Instead, they can/must be software-configured (see [set_laser_control](#)).
- 15-pin female D-SUB laser connector with all laser signals at the RTC5 slot cover (see [page 47](#)), 9-pin female D-SUB connector only by an optional laser adapter (see [page 50](#))
- 15-pin female D-SUB laser connector configurable by software command (see [page 146](#)).
- Laser control signals with 15 ns resolution and 20 mA output current
- Standby signals in YAG modes (see [page 149](#))
- YAG mode 5: Time between FirstPulseKiller signal and first laser pulse in YAG mode is freely programmable (see [page 149](#))
- Laser mode 6: LASERON signal synchronized with a continuously-running LASER1 signal (see [page 155](#))
- Pulse picking laser mode (see [page 157](#))
- Laser pulse period, pulse length or analog output are also programmable within a polyline⁽¹⁾ between two vectors (see "short list commands" [page 250](#)).
- Commands for position-dependent, speed-dependent, vector-defined and encoder-speed-dependent laser control (see [page 159](#))

Interfaces for Peripheral Equipment

- 16-bit digital output (see [page 51](#) and [page 232](#)):
 - Level of output signals selectable by a jumper (3.3 V or 5 V)
 - LATCH signal for synchronization of data transmission
- 8-bit digital output (see [page 52](#) and [page 233](#)):
 - Provided at the EXTENSION 2 socket connector (on the RTC4, this socket connector is named "LASER EXTENSION")
 - LATCH signal for synchronization of data transmission
 - Adjustable "stop output value"
- Analog outputs (see [page 48](#) and [page 233](#)):
 - 12 bit resolution
 - 0...10 V (0 V ... 2.50 V no longer available)
 - Adjustable "stop output value"
- 16-bit digital input (see [page 51](#) and [page 238](#)).
 - SYNC signal for synchronization of data transmission
- Programmable debouncing of external start signals (see [bounce_supp](#) and [page 240](#)).
- Regular (periodic) external list starts (see [242](#))
- New interfaces:
 - 2-bit digital input and 2-bit digital output at the D-SUB laser connector (see [page 48](#))
 - RS232 interface by an on-board 10-pin socket connector (see [page 54](#))
 - Stepper motor signals for 2 motors by an on-board 10-pin socket connector (see [page 58](#))
 - McBSP, I²C and SPI (serial peripheral interface) by an on-board 10-pin socket connector (see [page 54](#))
- For the RTC5 there is no IO extension board. Therefore, it does not have a socket connector for installing such a board (on the RTC4, this socket connector is named "EXTENSION 2"; the RTC5 EXTENSION 2 socket connector corresponds to the RTC4 "LASER EXTENSION" socket connector).

(1) Syn.: polygonal chain, polygonal line, polygonal traversal



General Programming

- Utility files for C, C++, C# and Delphi (see [page 66](#)). But no utility files for Basic.
- Commands for changing access rights to RTC5 Boards (see [page 96](#))
- Improved and extended list handling (see [page 75](#))
 - List memory with 1048576 storage positions
 - List memory free configurable (in two list buffers and a protected buffer area)
 - Defining protected subroutines
 - Loading lists with protection
 - Loops in lists and subroutines
 - A circular queue mode is not available (but can be emulated)
 - List status and list execution status
- “Short” list commands (e.g. to change speed, analog output, I/O port, etc.) can be executed without time losses (multiple “short” list commands can be executed within one 10 µs clock period, see [page 250](#)).
- Functions for error handling and download verification (see [page 98](#))

Laser Marking

- Vectors and arcs
- Commands for marking ellipses (see [page 105](#))
- Commands for marking spirals (see [page 194](#))
- Timed arc commands, timed 3D vector commands (see [page 223](#))
- Para-mark and para-jump commands for vector-defined laser control (see [page 166](#))
- Characters and texts
 - Defining character sets and text strings (see [page 87](#))
 - Commands for marking individual characters and for marking texts (with selectable character set, see [page 88](#))
 - Commands for marking dates, times and serial numbers (with selectable character set and selectable serial-number-set, see [page 170](#))
- Micro vector commands (direct execution of vectors without microvectorization)



Special Functions

- Synchronization of scan system and laser control
 - Automatic delay adjustments (see [page 121](#))
 - Sky Writing (see [page 127](#))
- Pixel output mode (scanning bitmaps, see [page 217](#))
 - Pixel frequencies up to 300 kHz, irrespective of the 10 µs clock
 - 15 ns resolution
 - 0–100% laser pulse length
 - Pixel-Mode 0 not supported
 - Reading of analog voltages is not supported
 - Pixel marking on sloped surfaces
- Commands for conditional execution of any list command (see [page 244](#))
- Possible wobble motion shapes include not only circles, but also ellipses, horizontal figure-of-8s, vertical figure-of-8s, and “freely definable wobble shapes”. Options for the orientation of the wobble shapes are: stationary in space, continuously and automatically adjusted to the current direction of motion, or any other freely assigned motion direction. Unlike “classic” wobble shapes, “freely definable wobble shapes” have an option to vary the laser power (see [page 189](#)).
- Camming functionality (see [camming](#))
- Enhanced signal recording (see [set_trigger](#) and [set_trigger4](#)).
- Processing-on-the-fly (see [page 199](#))
 - two encoder inputs (RS422) with 32-bit counter for Processing-on-the-fly correction with encoder signals on 2 axes (see [page 246](#)); alternatively: Processing-on-the-fly correction with McBSP signals (see [page 248](#))
 - 24-bit vector coordinates: objects larger than the image field are possible (virtual processing field, see [page 135](#) and [page 210](#))
 - up to 8 objects within the Processing-on-the-fly track delay (between trigger and marking position, see “[External List Start](#)”, [page 240](#))
 - Accurate “external list start”: If accordingly configured by [set_control_mode](#), the encoder counter can be reset by external start signals for synchronizing a Processing-on-the-fly process. The reset occurs fully simultaneously (without 10 µs jitter) with the external start signal.
 - Compensation of 2D motions (XY table)
 - Encoder-based Processing-on-the-fly correction for the Z-axis (“[FlyZ correction](#)”, see [page 215](#))

3 Safety During Installation and Operation

Read these operating instructions completely before you proceed with installing and operating the RTC5 PCI Board.

If there are any questions regarding the contents of this manual, please contact SCANLAB.

The following symbols are used in this manual:



Instructions that may affect a person's health are marked with a warning triangle next to the word "Danger".



Instructions that recommend appropriate use of this device or warn of damage that may occur to it are labeled by a circle with an "X" through it, next to the word "Caution".

3.1 Steps for Safe Operation



Caution!

- Carefully check your user program before running it. Programming errors can cause a break down of the system. In this case neither the laser nor the scan system can be controlled.
- Protect the board from humidity, dust, corrosive vapors and mechanical stress.
- For storage and operation, avoid electro-magnetic fields and static electricity. These can damage the electronics on the RTC5 Board. For storage, always use the antistatic bag the RTC5 is delivered in.
- The allowed operating temperature range is 15 °C to 60 °C.
- The storage temperature should be between –20 °C and +60 °C.

3.2 Laser Safety

The RTC5 is intended for controlling scan systems and lasers. Therefore all relevant laser safety directives must be known and applied before installation and operation. The customer is solely responsible for ensuring the laser safety of the entire system.



Danger!

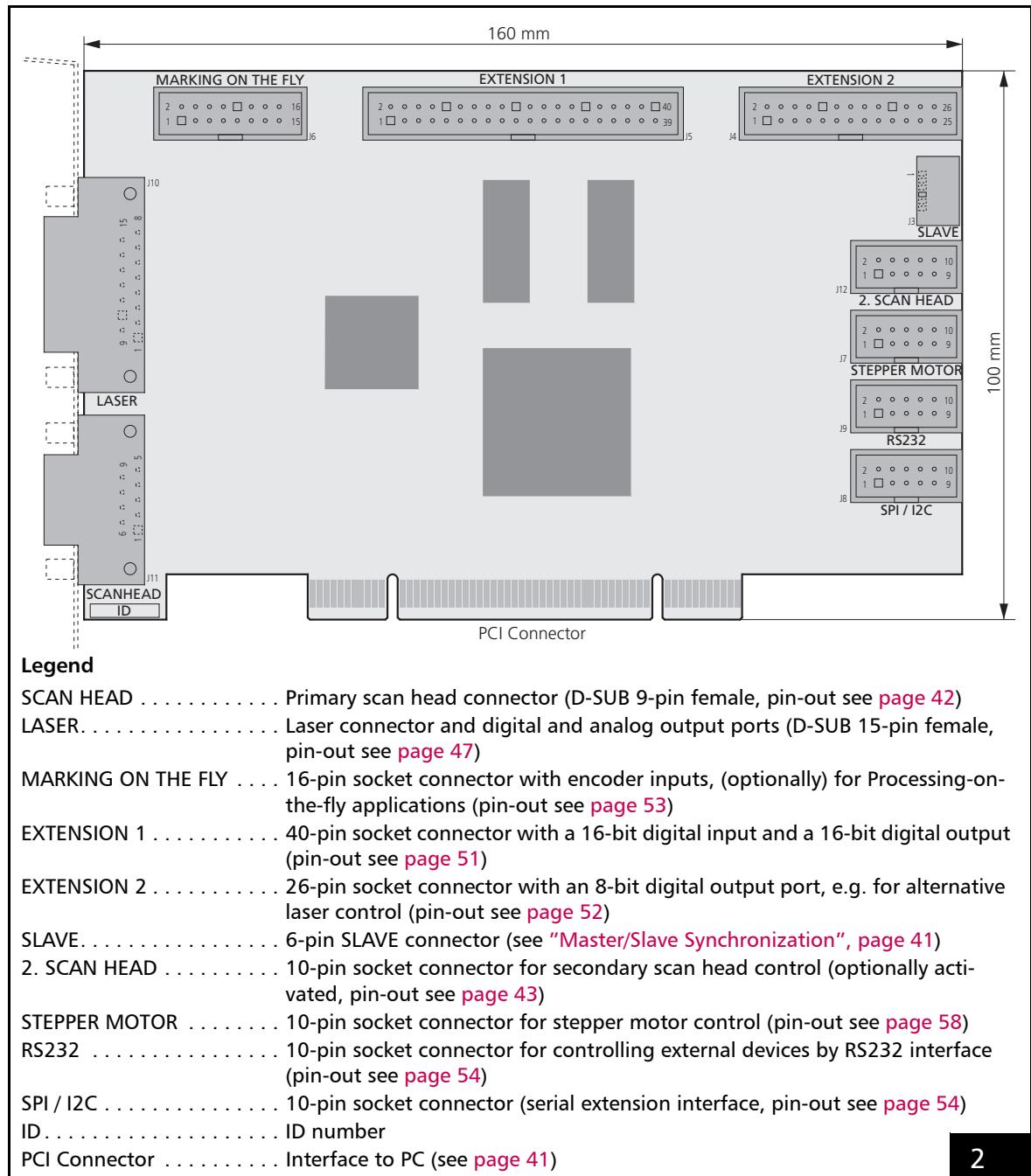
- All applicable laser safety directives must be adhered to. Safety regulations may differ from country to country. It is the responsibility of the customer to comply with all local regulations.
- Observe all laser safety instructions as described in your scan system manual, chapter "Safety during Installation and Operation".
- Always turn on the PC and the power supply for the scan head first before turning on the laser. Otherwise there is the danger of uncontrolled deflection of the laser beam. SCANLAB recommends the use of a shutter to prevent uncontrolled emission of laser radiation.

4 Layout and Interfaces

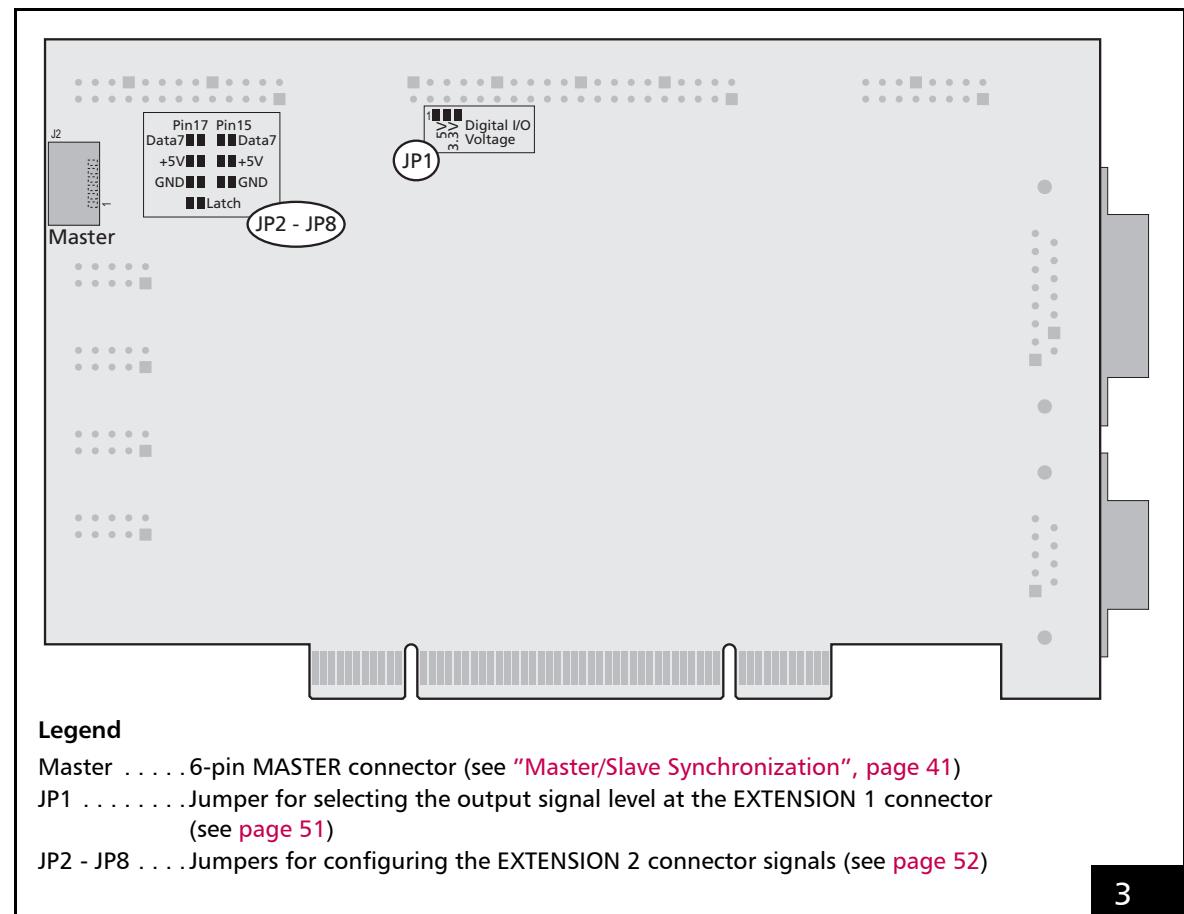
4.1 Connectors and Jumper Positions

Figure 2 and **figure 3** show the positions of the connectors and jumpers on the front and back side of the RTC5 PCI Board.

All connectors and jumper settings are described in the following sections.



Layout of the RTC5 PCI Board (front side)



Layout of the RTC5 PCI Board (back side)

3

4.2 Interface to PC

The RTC5's PCI connector is the interface to the PC.

The RTC5 can be installed into any Windows PC with a PCI bus interface and at least one free PCI slot.

Caution!

- The RTC5 Board does not support power-saving modes that switch off power to the PCI bus. Accordingly, you must disable standby or sleep modes of the operating system. See also the note on [page 26](#).

4.2.1 Master/Slave Synchronization

If multiple synchronously-clocked RTC5 Boards are to be used in a PC, then the RTC5 Boards must first be connected pairwise with each other by the MASTER and SLAVE connectors and then installed in adjacent PCI slots. Always connect a board's MASTER connector to the SLAVE connector of another board. Suitable connection cables are available from SCANLAB.

See also [chapter 6.6.3 "Master/Slave Operation", page 93](#) and ["Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 239](#).

4.3 Interfaces to Scan System

4.3.1 Scan Head Connectors and Transfer Protocol

The primary scan head connector SCAN HEAD and the optionally activated secondary scan head connector "2. SCAN HEAD" are available for digitally controlling scan systems (see [figure 2](#)). At those connectors, scan-system control values are transmitted and scan-system status signals received. Each scan head connector can transmit data for up to two axes. Consult your scan system's operating manual to determine which status signals are generated by your scan system and how they can be applied for monitoring purposes.

Data transfer between the RTC5 and the scan system is in accordance with the SL2-100 protocol. SCANLAB can supply an XY2-100 converter for signal transfer in accordance with the XY2-100 protocol (see "[XY2-100 Converter \(Optional\)](#)", [page 43](#)).

If neither the "second scan head control" option nor the 3D option is enabled (these options must be enabled by SCANLAB), only the primary scan head connector outputs signals for an XY scan system.

If the "second scan head control" option (but not the 3D option) is enabled, the secondary scan head connector outputs the same signal types as the primary scan head connector and two XY scan systems can be simultaneously controlled by one RTC5.

If, instead, the 3D option (but not the "second scan head control" option) is enabled, then a 3-axis scan system can be controlled by the two scan head connectors (if the primary scan head connector has been assigned a 3D correction table). Signals can then be outputted by the primary scan head connector to an XY scan head – and by both channels of the secondary scan head connector to the third axis (Z axis).

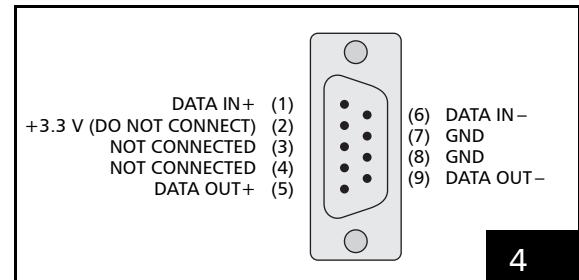
If *both* options ("second scan head control" and 3D option) are enabled, then users specify which signals (XY or Z) are to be outputted by which connector when assigning the correction table (see also "[2D and 3D Correction Files](#)", [page 137](#)).

If multiple RTC5 Boards with enabled 3D option are installed in a PC, then that many 3-axis systems can be simultaneously controlled.

For master/slave functionality, multiple RTC5 Boards can be run – synchronously clocked – in one PC if the command **load_program_file** has been executed on all boards (see "[Master/Slave Synchronization](#)", [page 41](#) and "[Initialization of the Board](#)", [page 69](#)).

Primary Scan Head Connector

[Figure 4](#) shows the pin-out of the primary scan head connector SCAN HEAD (D-SUB 9-pin female).



Pin-out of the 9-pin female D-SUB connector (SCAN HEAD) for scan head connection

The differential DATA OUT output channel transmits control values to the scan system. The differential DATA IN input channel receives the status signals returned by the scan system.

Pin 2 supplies 3.3 V power for SCANLAB's optional XY2-100 converter or a POF converter for optical data transmission (POF = Polymer Optical Fiber). This pin should not be used for other purposes.

Secondary Scan Head Connector (Optionally Activated)

The pin-out of the secondary scan head connector "2. SCAN HEAD" (10-pin socket connector) is shown in [figure 5](#).

DATA IN+ (1)	<input type="checkbox"/>	(2) DATA IN-
+3.3 V (DO NOT CONNECT)(3)	<input type="checkbox"/>	(4) GND
NOT CONNECTED (5)	<input type="checkbox"/>	(6) GND
NOT CONNECTED (7)	<input type="checkbox"/>	(8) DATA OUT-
DATA OUT+ (9)	<input type="checkbox"/>	(10) NOT CONNECTED

5

Pin-out of the on-board "2. SCAN HEAD" connector

The signals on this connector are optional and require activation by SCANLAB for controlling a second 2-axis scan system (by enabling the "second scan head control" option) and/or for controlling a 3-axis scan system (by enabling the 3D option) (see above and [chapter 2.3 "Optional Functionality", page 27](#)).

After the activation, this connector outputs – depending on the assigned correction tables (see above) – the same signal types as the primary scan head connector.

Slot cover (Optional)

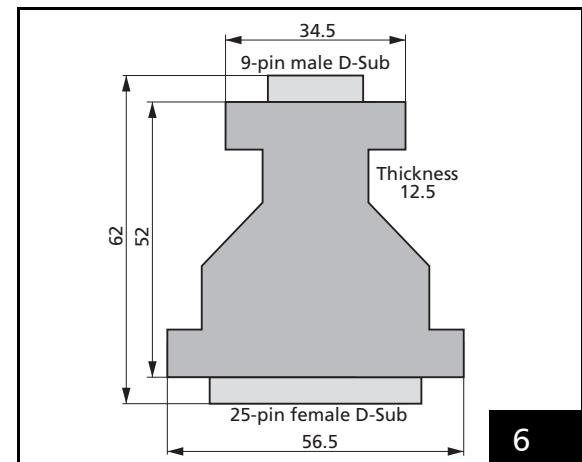
SCANLAB recommends using an additional slot cover for connecting a second scan head or a Z axis. A suitable slot cover with a 9-pin D-SUB connector – with the same pin-out as the primary scan head connector (see [figure 4](#)) – is available from SCANLAB.

4.3.2 XY2-100 Converter (Optional)

SCANLAB's optional XY2-100 converter converts the RTC5's SL2-100 control signals (20 bit) into XY2-100-compliant signals (16 bit), and converts the scan system's XY2-100 status signals into SL2-100-compliant signals (see [page 143](#)).

The XY2-100 converter introduces a 10 µs runtime latency to scan-system control. This runtime latency can be compensated by increasing the LaserOn Delay and LaserOff Delay by 10 µs each (see [set_laser_delays](#)).

[Figure 6](#) shows the dimensions of the XY2-100 converter.

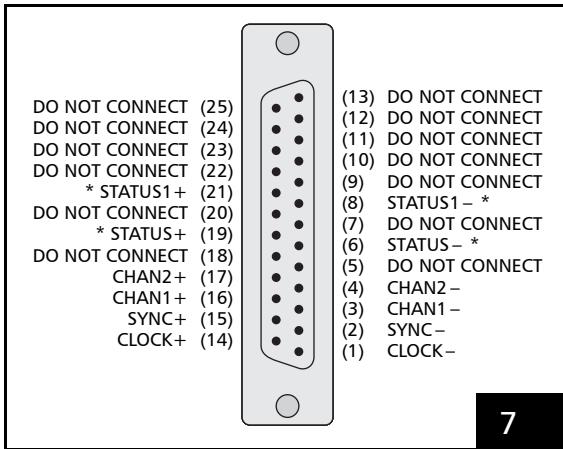


6

Dimensions of the XY2-100 converter

The XY2-100 converter's 9-pin D-SUB connector should be directly plugged into the RTC5's primary scan head connector or (by a short-as-possible 1:1 cable) connected to the corresponding pins of the RTC5's secondary scan head connector (for the pin-out, see [figure 4](#) and [figure 5](#)).

The XY2-100 converter's 25-pin D-SUB connector is compatible with scan heads that provide an XY2-100 standard interface. The pin-out is shown in [figure 7](#).

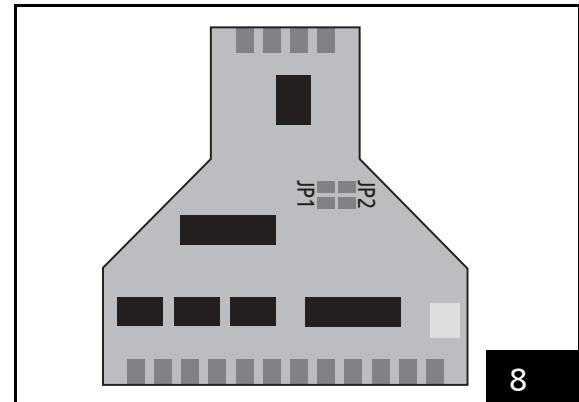


Pin-out of the XY2-100 converter's 25-pin female D-SUB connector

* For iDRIVE scan systems (intelliSCAN, intelliSCAN_{de}, intelliDRILL, intellicube, intelliWELD, varioSCAN_{de}), the STATUS \pm channel is the status channel of axis 2 (X axis; this channel is then also called STATUS2 \pm) and the STATUS1 \pm channel is the status channel of axis 1 (Y axis). For other scan systems, the STATUS1 \pm channel can not be used: "DO NOT CONNECT".

The data channels CHAN1 and CHAN2 transmit control values to the scan head. The SYNC and CLOCK channels transmit synchronization and clock signals to the scan system. The STATUS channel (and, if appropriate, the STATUS1 channel) receives XY2-100 compliant status signals returned by the scan system.

If very long cables are used for data transmission between the XY2-100 converter and the scan system, then the synchronization of bidirectional communication between the scan system and the RTC5 should be configured. This can be accomplished by two solder jumpers in the XY2-100 converter. To do so, carefully open the converter's housing by its four clip latches (e.g. using a screwdriver). The solder jumpers JP1 and JP2 are on the converter's PCB, between the two D-SUB connectors. **Figure 8** shows the positions of the jumpers on the PCB. The table below lists the possible jumper settings and the corresponding cable lengths. Other jumper settings as well as cable lengths above 25 m are not recommended.



Positions of solder jumpers JP1 and JP2 on the PCB of the XY2-100 converter

Jumper	Cable length *
JP1 closed (default configuration)	JP1 JP2 0 m to 20 m
JP2 closed	JP1 JP2 20 m to 40 m

* For each jumper configuration, the recommended cable length range depends – among others – on the used cable type and can differ from the values listed above.

4.3.3 Data Cables (Accessories)

For transmission of the data signals between the RTC5 (or the XY2-100 converter) and the scan system, appropriate cables are obtainable from SCANLAB.

For controlling SCANLAB's scan systems equipped with an SL2-100 interface (9-pin female D-SUB connector), data cables are available for either electrical transmission or optical fiber transmission. The optical fiber, too, is attached by a 9-pin D-SUB connector. Optical conversion (POF conversion) for optical data transmission takes place in the D-SUB connector. The operating voltage for POF conversion is supplied at the RTC5 scan head connectors and the scan system's digital interface.

Only electrical cables are obtainable for SCANLAB's scan systems equipped with a conventional XY2-100 interface (25-pin female D-SUB connector). Scan systems equipped with an optical interface (XY2-100-O) cannot be controlled by the RTC5.

Data cables are generally not included in the package. SCANLAB recommends the following design (for electrical data transmission):

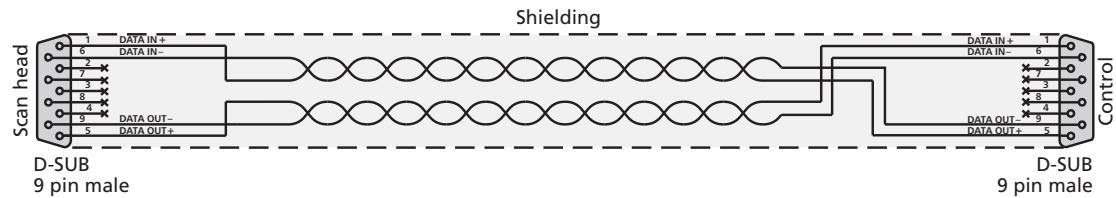
- For SL2-100-compliant data transmission, the cable should be fitted with 9-pin male D-SUB connectors at both ends. The two channels DATA IN \pm and DATA OUT \pm must consist of twisted cable pairs and be cross-connected at both D-SUB connectors (e.g. so that the RTC5's DATA OUT signal flows to the scan system's DATA IN input). The cable length should not exceed 25 m. SCANLAB recommends a cable impedance of 110 Ω , independent from the cable length.

- For XY2-100-compliant data transmission, the cable must have identical 25-pin (male) D-SUB connectors at both ends. The five (or six) channels SYNC \pm , CHAN1 \pm , CHAN2 \pm , STATUS \pm (and STATUS1 \pm) and CLOCK \pm must consist of twisted cable pairs. Together with a XY2-100 converter with standard jumper configuration, the data cable should not be longer than 10 m. If a longer data cable is needed, then the XY2-100 converter's solder jumpers possibly need to be reconfigured (see "["XY2-100 Converter \(Optional\)"](#), page 43). Cable lengths above 25 m are not recommended.
- The data cable must have coaxial copper braided shielding.
- The D-SUB connectors must have fully shielded metal housings.
- The electrical connection of the cable's braided shielding to the D-SUB housing should *not* be implemented as a wire. Instead, the cable's braided shielding should be *coaxially* connected to the D-SUB housing by shielded clamps.
- For XY2-100-compliant data transmission, the data cable's controller end must be fitted with a ferrite ring (e.g. Wuerth WE 742 711 32).

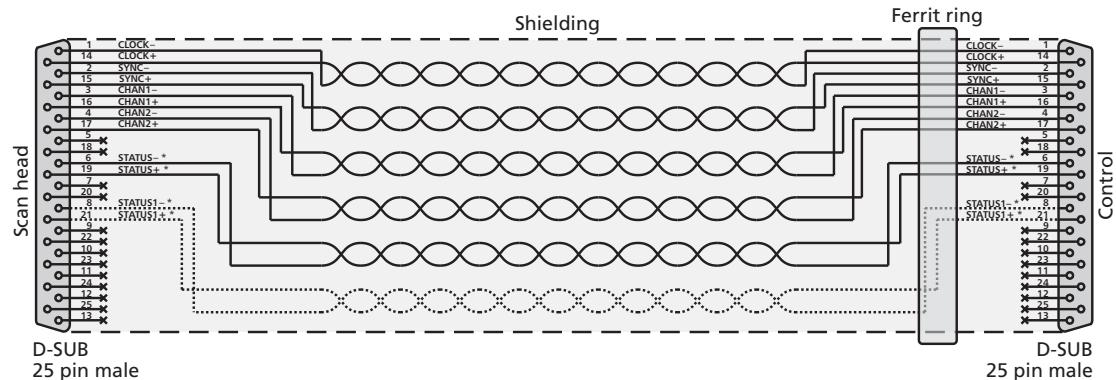
The data cable layout and pin assignments are shown in [figure 9](#).

Some scan heads use a single connector to provide both the power supply voltages and the data signals. For these scan heads, SCANLAB recommends implementing a cabling solution that allows the use of separate cables for data and power. The data-section of such a cabling solution should be designed to accommodate the data cable shown in [figure 9](#).

SL2-100-compliant data transmission



XY2-100-compliant data transmission



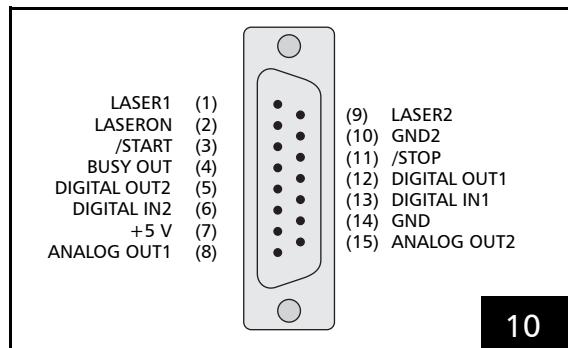
* For iDRIVE scan systems (intelliSCAN, intelliSCAN_{de}, intelliDRILL, intellicube, intelliWELD, varioSCAN_{de}), the STATUS_{2±} channel is the status channel of axis 2 (X axis; this channel is then also called STATUS_{2±}) and the STATUS_{1±} channel is the status channel of axis 1 (Y axis). For other scan systems, the STATUS_{1±} channel is not needed.

Data cable layout and pin assignments

4.4 Interfaces for the Laser and Peripheral Equipment

4.4.1 Laser Connector

Figure 10 shows the pin-out of the 15-pin D-SUB connector for the laser.



Pin-out of the 15-pin female D-SUB connector for the laser (LASER)

Laser Signals

The laser output signals LASER1 and LASER2 at pin (1) and pin (9) depend on the selected laser control mode, see the corresponding timing diagrams in chapter 7.4 "Laser Control", page 144:

	LASER1	LASER2
CO ₂ mode	Modulation pulse 1 or standby signal	Modulation pulse 2 or standby signal
YAG modes 1-3, 5	Q-Switch signal	FirstPulseKiller signal
Laser mode 4	Modulation pulse or standby signal	FirstPulseKiller signal
Laser mode 6	Modulation pulse or standby signal	–

All laser output signals (LASERON, LASER1 and LASER2) are continuously generated by the RTC5 Board. They are digital TTL level signals and are referenced to GND2.

On RTC5 Boards with the option "optoelectronic couplers" the GND2 and the laser output signal are galvanically decoupled from the PC ground (GND). Otherwise, GND and GND2 are identical, see chapter 2.3 "Optional Functionality", page 27.

The maximum current load is 20 mA.

All laser signals can be set to either *active-low* or *active-high* logic by the command `set_laser_control`. Active-low means that a logical 1 ("Laser On", for instance) is represented by a LOW level (0 V, TTL). Active-high means a logical 1 is represented by a HIGH level (+5 V, TTL). Set the TTL laser signal level according to the specifications of your laser control. Observe the documentation of your laser.

The commands `config_laser_signals` and `config_laser_signals_list` can be used to configure pins (1), (2) and (9) of the laser connector, see page 146.

External Control Signals

The external control signals /START and /STOP (TTL active-low) are referenced to PC ground (GND). Both input signals are connected internally to +3.3 V by pull-up resistors. Refer to the chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization", page 239.

BUSY Status

The BUSY status is available as the BUSY OUT signal at pin (4). The BUSY OUT signal is HIGH when the BUSY status is set, see chapter 6.4.3 "List Execution Status", page 77. The signal is referenced to GND.



Digital Input and Output

The RTC5 provides a 2-bit digital input (DIGITAL IN1 and DIGITAL IN2) and a buffered 2-bit digital output (DIGITAL OUT1 and DIGITAL OUT2).

For programming the input and output see "["2-Bit Digital Input", page 238](#)" and "["2 Bit Digital Output Port", page 233.](#)

Input signals:

- LOW level < 0.6 V, HIGH level > 2.3 V
- Max. input voltage range: -0.5 V ... +5.5 V
- Input resistance (pull-up) > 4.7 kΩ

Output signals:

- LOW level < 0.55 V, HIGH level > 3.8 V,
- Maximum current load of each signal is 20 mA

The signals are referenced to GND.

Analog Output Ports

The RTC5 provides two general purpose 12-bit analog output ports, ANALOG OUT1 and ANALOG OUT2⁽¹⁾.

For programming the outputs see "["12-Bit Analog Output Ports", page 233.](#)

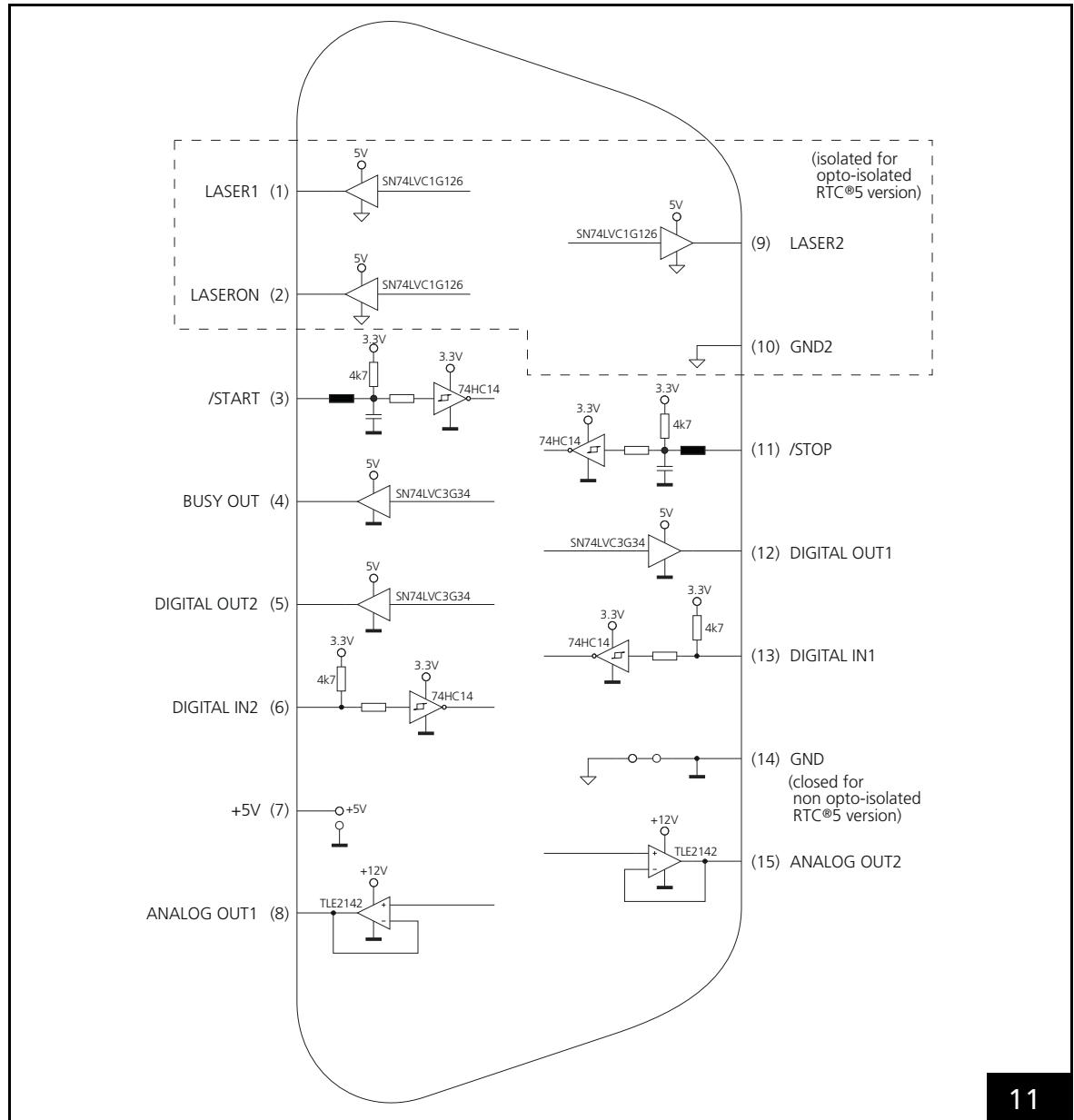
The output voltage range of both analog output ports is set to 0 V ... 10 V.

Both signals are referenced to PC ground (GND). The maximum current load per signal is 5 mA.

I/O Circuits

[Figure 11](#) shows the I/O circuits of the 15-pin D-SUB connector for the laser.

(1) The ANALOG OUT2 signal is also available by the MARKING ON THE FLY socket connector (see [page 53](#)).



I/O circuits of the 15-pin female D-SUB connector for the laser (LASER)

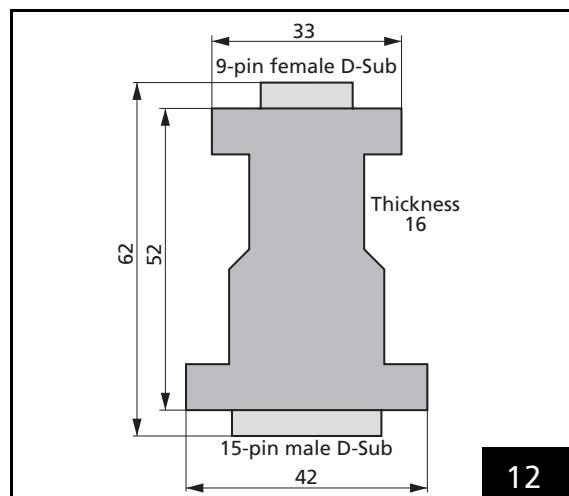
Laser Adapter (Optional)

An optional laser adapter from SCANLAB can be plugged into the RTC5's 15-pin LASER connector. This adapter's 9-pin female D-SUB connector provides the same signals and pin-out as the RTC4's 9-pin laser connector.

Notes

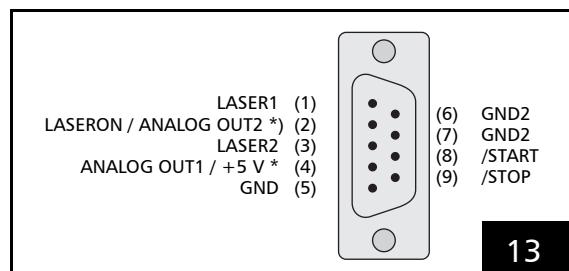
- The laser adapter can *not* be directly plugged into the RTC5 if an XY2-100 converter is already plugged in.
- The BUSY OUT signal, 2-bit digital input and 2-bit digital output are *not* available at the laser adapter's 9-pin D-SUB connector.

Figure 12 shows the dimensions of the laser adapter.



Dimensions of the optional laser adapter

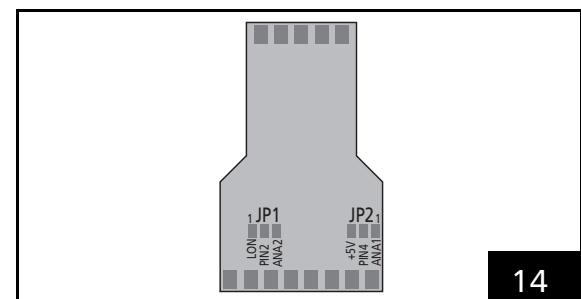
The pin-out of the 9-pin D-SUB connector is shown in **figure 13**.



Pin-out of the laser adapter's 9-pin female D-SUB connector

* dependent on the jumper configuration of the solder jumpers on the laser adapter's PCB

The signals at pins (2) and (4) of the 9-pin D-SUB connector can be selected by two solder jumpers in the laser adapter. To do so, carefully open the laser adapter's housing by its four clip latches (e.g. using a screwdriver). The solder jumpers **JP1** and **JP2** are on the laser adapter's PCB, between the two D-SUB connectors. **Figure 14** shows the positions of the jumpers on the PCB. The table below lists the possible jumper settings and the corresponding signals.



14

Positions of solder jumpers **JP1** and **JP2** on the laser adapter's PCB

Jumper JP1 (signal at pin 2)	Position 1-2  LASERON (standard configuration)	Position 2-3  ANALOG OUT2
Jumper JP2 (signal at pin 4)	Position 1-2  ANALOG OUT1 (standard configuration)	Position 2-3  +5 V

4.4.2 EXTENSION 1 Socket Connector

Figure 15 shows the pin-out of the 40-pin EXTENSION 1 socket connector.

DIGITAL OUT0	(1)	<input type="checkbox"/>	<input type="checkbox"/>	(2)	DIGITAL IN0
DIGITAL OUT1	(3)	<input type="radio"/>	<input type="radio"/>	(4)	DIGITAL IN1
DIGITAL OUT2	(5)	<input type="radio"/>	<input type="radio"/>	(6)	DIGITAL IN2
DIGITAL OUT3	(7)	<input type="radio"/>	<input type="radio"/>	(8)	DIGITAL IN3
DIGITAL OUT4	(9)	<input type="radio"/>	<input type="checkbox"/>	(10)	DIGITAL IN4
DIGITAL OUT5	(11)	<input type="radio"/>	<input type="radio"/>	(12)	DIGITAL IN5
DIGITAL OUT6	(13)	<input type="radio"/>	<input type="radio"/>	(14)	DIGITAL IN6
DIGITAL OUT7	(15)	<input type="radio"/>	<input type="radio"/>	(16)	DIGITAL IN7
DIGITAL OUT8	(17)	<input type="radio"/>	<input type="radio"/>	(18)	DIGITAL IN8
DIGITAL OUT9	(19)	<input type="radio"/>	<input type="checkbox"/>	(20)	DIGITAL IN9
DIGITAL OUT10	(21)	<input type="radio"/>	<input type="radio"/>	(22)	DIGITAL IN10
DIGITAL OUT11	(23)	<input type="radio"/>	<input type="radio"/>	(24)	DIGITAL IN11
DIGITAL OUT12	(25)	<input type="radio"/>	<input type="radio"/>	(26)	DIGITAL IN12
DIGITAL OUT13	(27)	<input type="radio"/>	<input type="radio"/>	(28)	DIGITAL IN13
DIGITAL OUT14	(29)	<input type="radio"/>	<input type="checkbox"/>	(30)	DIGITAL IN14
DIGITAL OUT15	(31)	<input type="radio"/>	<input type="radio"/>	(32)	DIGITAL IN15
LATCH OUT	(33)	<input type="radio"/>	<input type="radio"/>	(34)	SYNC OUT
VCC OUT	(35)	<input type="radio"/>	<input type="radio"/>	(36)	BUSY OUT
+5 V	(37)	<input type="radio"/>	<input type="radio"/>	(38)	+5 V
GND	(39)	<input type="radio"/>	<input type="radio"/>	(40)	GND

15

Pin-out of the 40-pin on-board EXTENSION 1 socket connector

Configuring the Output Signal Level

By Jumper JP1 on the back side of the RTC5 (see figure 3), the level of all output signals at the EXTENSION 1 socket connector (DIGITAL OUT0-15, LATCH_OUT, SYNC_OUT, BUSY_OUT, VCC_OUT) can be configured for 5 V or 3.3 V (see chapter 2.4.1 "Jumper JP1: Selecting the Output Signal Level at the EXTENSION 1 Socket Connector", page 28).

For user monitoring purposes, the selected signal level is also continuously outputted at pin (35): signal VCC_OUT. VCC_OUT is referenced to PC ground (GND). The maximum current load is 100 mA.

16-Bit Digital Input and Output

The 40-pin EXTENSION 1 socket connector provides a 16-bit digital TTL input and a buffered 16-bit digital TTL output (see figure 15). The level of all output signals must be configured with a jumper (see above).

For programming the input and output see "16-Bit Digital Input", page 238, "16-Bit Digital Output Port", page 232 and "Conditional Command Execution", page 244.

The input and output signals are referenced to PC ground (GND). The maximum current load of the output signals is 8 mA.

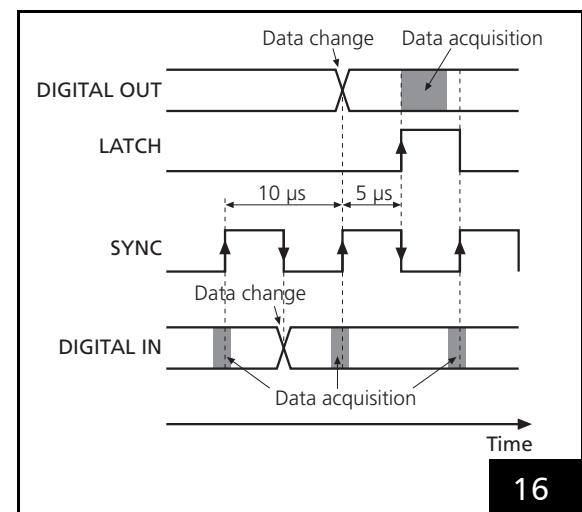
Synchronization of Data Transmission

If several bits are simultaneously transferred as a data words (i.e. if the bits are not transferred independently from each other) by the 16-bit digital output port or the 16-bit digital input port, then the LATCH or SYNC signal (respectively) should be used for synchronization of data transmission.

The LATCH signal (a 5 µs pulse, active-high) is outputted at pin (33) as a trigger signal for acquiring the output values of the 16-bit digital output port. The RTC5 automatically generates the LATCH signal when the output value at the 16-bit digital output port changes. The output value should be read-out with the rising edge of the LATCH signal, which is generated 5 µs after the value change at the 16-bit digital output port (see figure 16).

To synchronize data transmission at the 16-bit digital input port, pin (34) continuously provides a SYNC signal (a square wave with 5 µs pulse length and 10 µs period). Value changes at the 16-bit digital input port should be made with a falling edge of the SYNC signal. The RTC5's DSP always accepts the currently provided value with the rising edge of the SYNC signal (see figure 16).

The synchronization signals are referenced to PC ground (GND). The maximum current load is 10 mA.



16

Synchronization of data transmission by LATCH or SYNC signal

BUSY Status

The BUSY OUT signal at pin (36) is identical to the BUSY OUT signal at the LASER connector (see page 47). The signal is referenced to GND.

4.4.3 EXTENSION 2 Socket Connector

The 26-pin EXTENSION 2 socket connector on the RTC5 Board (on the RTC4, the corresponding socket connector is labeled LASER EXTENSION) provides a buffered 8-bit digital output port (DATA0 to DATA7). The pins (15) and (17) are configured with jumpers.

(LSB) DATA0 (1)	<input checked="" type="checkbox"/>	<input type="checkbox"/>	(2) GND
DATA1 (3)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	(4) NOT CONN.
DATA2 (5)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	(6) +5 V
DATA3 (7)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	(8) NOT CONN.
DATA4 (9)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	(10) NOT CONN.
DATA5 (11)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	(12) NOT CONN.
DATA6 (13)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	(14) NOT CONN.
+5 V / DATA7 / GND* (15)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	(16) NOT CONN.
+5 V / DATA7 / GND / LATCH* (17)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	(18) +5 V
LASER2 (19)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	(20) → (21)
(20) ← (21)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	(22) LASER1
GND2 (23)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	(24) NOT CONN.
+5 V (25)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	(26) LASERON

17

Pin-out of the on-board EXTENSION 2 socket connector

* depends on jumper settings

Jumper Setting

Pins (15) and (17) of the EXTENSION 2 socket connector have to be configured by the jumpers **JP2** and **JP8** on the back side of the RTC5 (see [figure 3](#)). The most significant bit (DATA7) of the 8-bit output value can be assigned to pin (15) or to pin (17). Alternatively, each of the two pins can be set permanently to +5 V (HIGH) or to GND (LOW level). Alternatively, pin (17) can be configured for the LATCH signal (see below) by closing the correspondingly labeled jumper (see [page 28](#)).

Examples

- If the DATA7 bit is assigned to pin (15), the full 8-bit output value is available at the output port (odd numbered pins (1) to (15) of the EXTENSION 2 socket connector).
- Setting pin (15) permanently to HIGH results in an offset of 128 for the output value and restricts the output value range to (128 ... 255).

- Setting pin (15) to LOW restricts the output value range to 0 ... 127.
- The DATA7 bit can be used for other purposes by assigning it to pin (17).

Laser Signals

Like the laser signals of the LASER connector, the output signals LASER1 and LASER2 depend on the selected laser control mode (see "[Laser Signals](#)", [page 52](#)) and are referenced to GND2. On RTC5 Boards with the option "optoelectronic couplers" the GND2 and the laser output signal are galvanically decoupled from the PC ground (GND). Otherwise, GND2 are GND identical (see [chapter 2.3 "Optional Functionality"](#), [page 27](#)).

8-Bit Digital Output Port

The buffered 8-bit digital output port (TTL level) is intended for YAG lasers with a digital lamp current control. However, it can be used for any other purpose as well. The output is in high-impedance mode (tri-state) until an initial value is assigned to it.

For programming the output see "[8-Bit Digital Output Port](#)", [page 233](#).

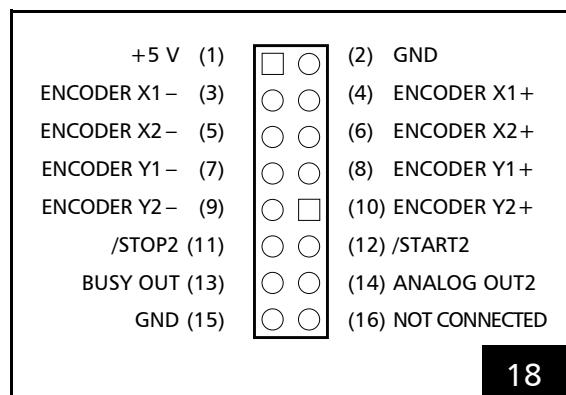
The most significant bit (MSB) (DATA7) of the output value can be used for other purposes, e.g. for controlling a shutter. To do this, the MSB can be assigned to an extra pin on the EXTENSION 2 socket connector (see above).

By an appropriate jumper setting (see above), Pin (17) can output a LATCH signal. The RTC5 automatically generates the LATCH signal (a 5 µs pulse, active-high) when the value at the 8-bit digital output port changes. If several bits are simultaneously transferred as a data word (i.e. if the bits are not transferred independently from each other) by the 8-bit digital output port, then the LATCH signal should be used for synchronization of data transmission (for example, as a trigger signal for the laser to assume a different lamp current). The value at the 8-bit digital output port should be read-out with the rising edge of the LATCH signal, which is generated 5 µs after the value change at the 8-bit digital output port (see also [figure 16](#)).

The LATCH signal is referenced to PC ground (GND). The maximum current load is 10 mA.

4.4.4 MARKING ON THE FLY Socket Connector

The MARKING ON THE FLY socket connector provides, among other things, encoder inputs for incorporating encoder signals. Together with the optional Processing-on-the-fly-functionality, this enables laser material processing of moving work-pieces (e.g. on a moving conveyer belt or rotating disk, see [page 199](#)). The pin-out is shown in [figure 18](#).



18

Pin-out of the (on-board) MARKING ON THE FLY socket connector

Encoder Inputs

The RTC5 provides two encoder inputs ENCODER X and ENCODER Y. Each of the encoder inputs are designed for a pair (1, 2) of standardized RS422 differential signals (see also [page 247](#)).

External Control Signals

The external control signals /START2 and /STOP2 (TTL active-low) are referenced to PC ground (GND). Both input signals are connected internally to +3.3 V by pull-up resistors. See also [chapter 9.3.1 "Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization"](#), page 239.

Analog Output Port

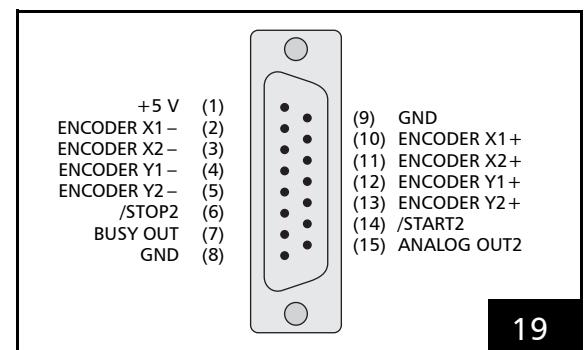
The 12-bit analog output port ANALOG OUT2 at pin (14) is identical to the ANALOG OUT2 output port at the LASER connector (see [page 48](#)). It is referenced to PC ground (GND).

BUSY Status

The BUSY OUT signal at pin (13) is identical to the BUSY OUT signal at the LASER connector (see [page 47](#)). The signal is referenced to GND.

Slot Cover (Optional)

SCANLAB recommends using an additional slot cover for using the inputs and signals of the MARKING ON THE FLY socket connector. A suitable slot cover with a 15-pin D-SUB connector is available from SCANLAB. [Figure 19](#) shows the pin-out of this D-SUB connector.



19

Pin-out of the 15-pin female D-SUB connector of the optional MARKING ON THE FLY slot cover

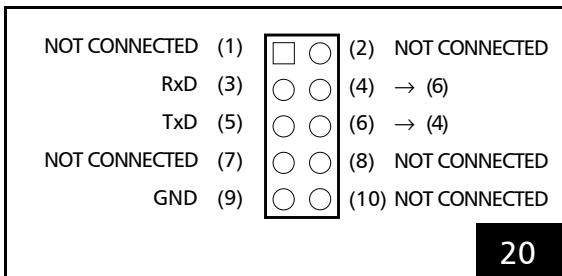
4.4.5 RS232 Socket Connector

The "RS232" socket connector provides an RS232 interface for controlling external devices. The pin-out is shown in [figure 20](#).

Max. input voltage range: -25 V ... +25 V

Max. output voltage range: -13 V ... +13 V

The signals are referenced to PC ground GND.



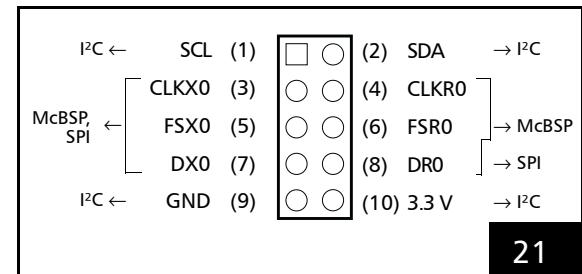
Pin-out of the (on-board) "RS232" socket connector

The command `rs232_config` can be used to set the baud rate (default setting: 9600 baud). The other parameters cannot be altered (data bits: 8, start bits: 1, stop bits: 1, parity: none).

For programming signal output and input by the RS232 interface, see [page 237](#) and [page 238](#).

4.4.6 SPI / I²C Socket Connector

The "SPI / I²C" socket connector is a hardware interface intended for several data exchange methods: McBSP (Multichannel Buffered Serial Port), SPI (Serial Peripheral Interface) and I²C (Inter-Integrated Circuit). The relevant pin-outs are shown in [figure 21](#). Only a subset of the pins is actually used depending on the chosen method.



Pin-out of the (on-board) "SPI / I²C" socket connector. The pins relevant for the several data exchange methods are indicated.

Use as McBSP Interface

The factory default data exchange method at the "SPI / I²C" socket connector is McBSP.

The "SPI / I²C" socket connector is automatically initialized with `DataDelay = 1` and 8 MHz clock frequency. Relevant for McBSP are pin (3), (5), (7) (outgoing signals) and pin (4), (6), (8) (incoming signals).

The following signals are continuously outputted after `load_program_file`:

- the clock signal at pin (3)
- every 10 µs a frame synchronization signal FSX at pin (5)
- a data signal DX at pin (7) (Default: SampleY|SampleX), see `set_mcbsp_out`.

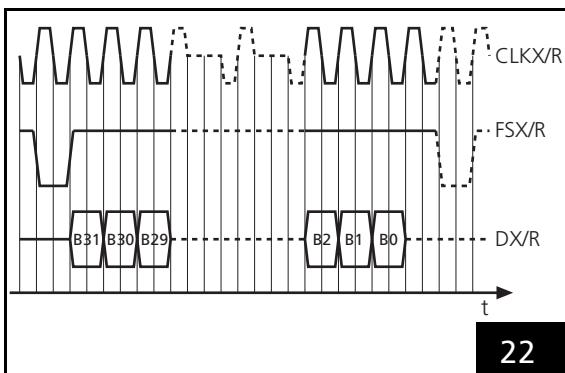
The McBSP interface functionality including associated commands are provided among other things for directly integrating position signals into applications.

- In Processing-on-the-fly applications, for example, a robot arm's position signals can be used to implement laser material processing by moving scan systems (see [page 199](#) and [page 248](#)) – as an alternative to encoder-control based Processing-on-the-fly processing of moving workpieces.
- Another possibility is to use the incoming signals at the McBSP interface to align the scan system above the workpiece with controllable timing (see "[Online Positioning](#)", [page 187](#) and [page 248](#)).

Information on using the McBSP interface functionality for data output and to feed in data can be found in [chapter 9.1.7 "McBSP/SPI Interface"](#), [page 237](#) and [chapter 9.2.4 "McBSP/SPI Interface"](#), [page 238](#).

The `mcbsp_init` command allows setting a data delay for the McBSP data transmission independently for the transmitter and receiver. Possible values range from 0 to 2 (default: 1).

All signals are referenced to PC ground GND.



RTC5 as transmitter

The following specifications apply to CLKX0, FSX0 and DX0:

- Signal level 3.3 V TTL
- McBSP mode:
 - Single phase frame
 - Single element per frame
 - 32 bits per element
 - DataDelay N bit
- The timing diagram of the McBSP signals is shown in [figure 22](#). A frame synchronization signal (active-low) is generated upon the rising edge of the clock and held for one clock cycle (1 data bit). The 32 data bits are transmitted after XDelay clock cycles at the rising edges of the clock and in the sequence Bit31...Bit0. The transmission frequency is by default 8 MHz (4 µs per data word). Alternatively, a value between 4 MHz and 16 MHz can be set by `set_mcbsp_freq`.
- The `set_mcbsp_out` and `set_mcbsp_out_ptr` commands lets you select the data signals to be transmitted. The signals are continuously transmitted every 10 µs.

Timing diagram of McBSP signals at 1 bit data delay
(default: XDelay = RDelay = 1)



RTC5 as receiver

The following specifications apply to CLKR0, FSR0, DRO:

- Signal level 3.3 V or 5 V TTL.
- McBSP mode:
 - Single phase frame
 - Single element per frame
 - 32 bits per element
 - DataDelay N bit
- The timing diagram of the McBSP signals is shown in [figure 22](#). The frame synchronization signal (active low) generated upon a rising edge (of an external clock signal) is detected upon the clock's next falling edge (trailing edge of the external clock pulse). The duration of the frame synchronization signal is irrelevant. After RDelay clock cycles the 32 data bits are detected with each additional falling edge of the clock signals in the sequence Bit #31...Bit #0, provided they are transmitted with rising edges.

Take account of the following information:

- The bit frequency (receiving frequency) is exclusively determined by the incoming clock pulses and has a maximum limit of 16 MHz.
- The last data bit (Bit #0) must be followed by transmission of at least one additional external clock cycle to ensure that the interface's DSP side acquires and buffers the data word. Simultaneously with this clock cycle, you can already initiate another new transfer by a frame synchronization signal.

- As of OUT 526: After a [load_program_file](#), the McBSP data is internally permanently acquired and buffered as soon as (new) data is transmitted. Newer data words overwrite older ones.
 - After a reset by [load_program_file](#) and/or after activation of Processing-on-the-fly correction by [set_fly_x_pos](#), [set_fly_y_pos](#) or [set_fly_rot_pos](#), the input values are stored to internal memory location 0.
 - After activation of online positioning by [set_mcbsp_x](#), [set_mcbsp_y](#) and/or [set_mcbsp_rot](#) or [set_mcbsp_matrix](#), the input values are stored to internal memory locations 1 or 2 (see [page 187](#)).
 - The most recent fully transferred values can be queried from a corresponding memory location by [read_mcbsp](#).
- The McBSP interface always ignores the first frame synchronization signal after a [load_program_file](#) or [mcbsp_init](#), so available data is not transmitted. If necessary, a dummy value should be initially sent to the interface (this applies to both Processing-on-the-fly applications and online positioning). You can use [read_mcbsp](#) to check for successful transmission.
- If using version OUT 525 or older, see also "Version info" under [get_mcbsp](#) and [get_mcbsp_list](#).
- As of OUT 532: After activation of Processing-on-the-fly correction by [set_mcbsp_in](#) or [set_mcbsp_in_list](#), the input values are transferred to internal memory locations 0 and 3.
- As of OUT 537: After activation of Processing-on-the-fly correction by [set_multi_mcbsp_in](#) or [set_multi_mcbsp_in_list](#), the input values are transferred to internal memory locations 0 through 3.

Use as I²C Interface

The "SPI / I²C" socket connector is used to transmit the digital data to the DSP, when the RTC5 ADC add-on board is operated, see also [chapter 4.4.8 "Analog Inputs \(Optional Accessory\)", page 59.](#)

At present, the I²C clock cycle signals (SCL at pin (1) and I²C data signals (SDA at pin (2) are not discretionary for other purposes.

Use as SPI Interface

The "SPI / I²C" socket connector can be set-up for a serial synchronous data transmission, if required. Then SPI functionality is used instead of the McBSP functionality (default). For this purpose the command `mcbsp_init_spi` can be called at any time. A data transmission which is in progress is cancelled by that. The previous state (McBSP functionality) can be reestablished with `mcbsp_init`.

After initialization only one data word can be transmitted every 10 µs, for example, see [set_mcbsp_in](#).

The default transmission frequency is 8 MHz (4 µs per data word). Alternatively, a value between 4 MHz and 16 MHz can be set. For this purpose the command `set_mcbsp_freq` is called (see also `mcbsp_init` or `mcbsp_init_spi`).

After `mcbsp_init_spi` has been called the RTC5 acts as SPI master device in terms of the SPI standard. The RTC5 is a master device for a single SPI slave only.

Necessary for operation are:

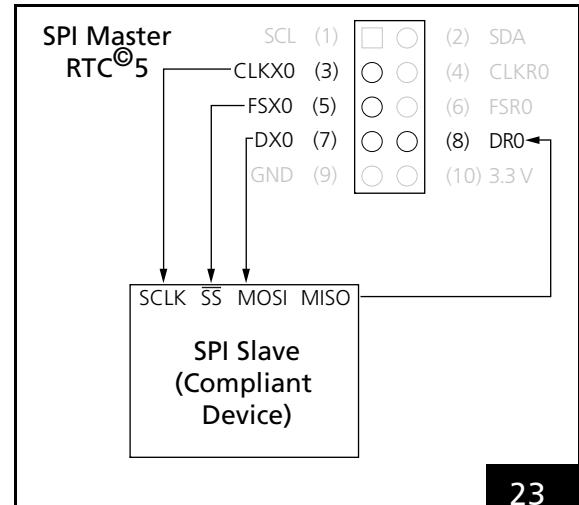
- 1 control line for the clock signal (CLKX0 at pin (3))
- 1 control line for the active-low slave select signal (FSX0 at pin (5))
- 1 data line – from a master point of view – outgoing (DX0 at pin (7))
- 1 data line – from a master point of view – incoming (DRO at pin (8))

Pin (4) CLKR0 and pin (6) FSR0 are not connected.

The specification 3.3 V TTL or 5 V TTL applies for all signal levels.

The signal sequence is the same as with the McBSP mode, though XDelay und RDelay are always = 1.

The electrical connection of the RTC5 and an SPI slave device is shown in [figure 23](#).



23

RTC5 as SPI master with SPI slave – signals and electrical connection. SCLK = Serial Clock, MOSI = Master Output - Slave Input, MISO = Master Input - Slave Output, SS = Slave Select. The relevant abbreviations and designations are inconsistent in technical references. That is, they are arbitrarily chosen in this document and a variety of synonyms does exist.

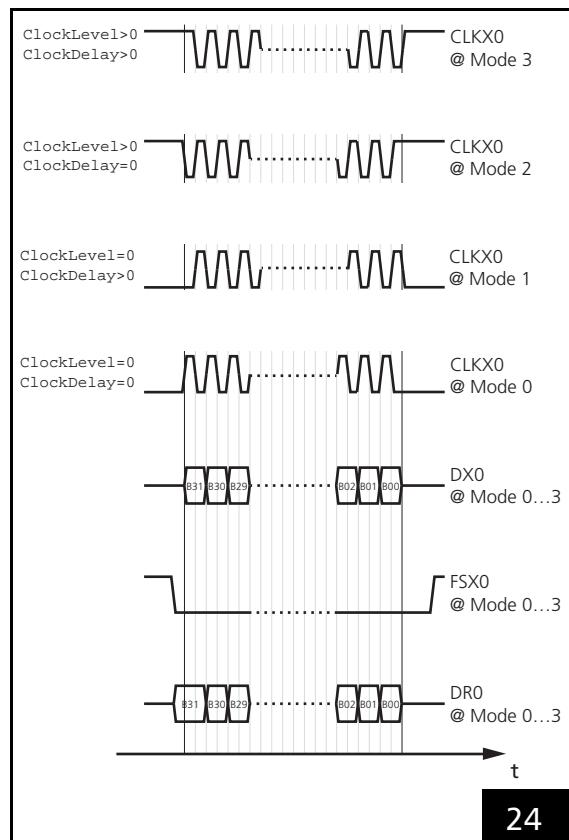
With SPI (in contrast to McBSP data transmission), data input and data output take place synchronously together with a clock signal (the clock signal is generated by the SPI master). The SPI timing diagram is shown in [figure 24](#).

The clock signal properties 'polarity' (ClockLevel) and 'phase' (ClockDelay) can be set by the command `mcbsp_init_spi`. The resulting data transmission formats are designated Mode 0...Mode 3 according to the SPI standard, see the following table.

SPI mode	Clock signal polarity	Clock signal phase
0	clock idle LOW – ClockLevel=0	is simultaneous with data bits – ClockPhase=0
1	clock idle LOW – ClockLevel=0	is delayed a half clock signal – ClockPhase>0
2	clock idle HIGH – ClockLevel>0	is simultaneous with data bits – ClockPhase=0
3	clock idle HIGH – ClockLevel>0	is delayed a half clock signal – ClockPhase>0

All signals are referenced to PC ground GND.

In contrast to the McBSP configuration, immediately after initialization all signals at the corresponding pins are static and the RTC5 does not generate a clock signal permanently.



24

Timing-Diagram of the SPI signals. The clock signal differs in regards to polarity and phase for mode 0, 1, 2 and 3.
Grid spacing is a half clock signal.

4.4.7 STEPPER MOTOR Socket Connector (Stepper Motor Control)

The “STEPPER MOTOR” socket connector can supply signals for controlling two stepper motors⁽¹⁾.

The pin-out is shown in figure 25.

SWITCH1 (1)	<input type="checkbox"/>	SWITCH2 (2)
ENABLE1 (3)	<input type="checkbox"/>	ENABLE2 (4)
DIRECTION1 (5)	<input type="checkbox"/>	DIRECTION2 (6)
CLOCK1 (7)	<input type="checkbox"/>	CLOCK2 (8)
GND (9)	<input type="checkbox"/>	GND (10)

25

Pin-out of (on-board) “STEPPER MOTOR” socket connector

All signals are referenced to PC ground GND.

The output signals (ENABLE, DIRECTION and CLOCK) are TTL-level signals (5 V). The RTC5 generates a periodic clock signal of active-high 5 µs pulses (the CLOCK signal is permanently LOW between these pulses). With the ENABLE signals a user program can, for example, switch the motor current on and off. The DIRECTION signals can set the direction and each CLOCK pulse can be used to execute a single step.

The SWITCH inputs (active low) are connected internally to +3.3 V by 10 kΩ pull-up resistors to facilitate integration of a final switch signal (3.3 V or 5 V TTL signal or input referenced to ground).

For programming the stepper motor signals, see chapter 9.1.5 “Stepper Motor Control”, page 234.

(1) Stepper motor signals are available only for RTC5 Boards with DSP version numbers 2 and higher (see [get_RTC_version](#) bits #16-23). For older RTC5 Boards, stepper motor commands do not execute.

4.4.8 Analog Inputs (Optional Accessory)

When the RTC5 PCI Board is equipped with the ADC add-on board (optional accessory only for the RTC5 PCI Board, #121126), the two 10 V analog inputs ANALOG IN0 and ANALOG IN1 become available (for ANALOG IN2, see section [Notes, page 59](#)).

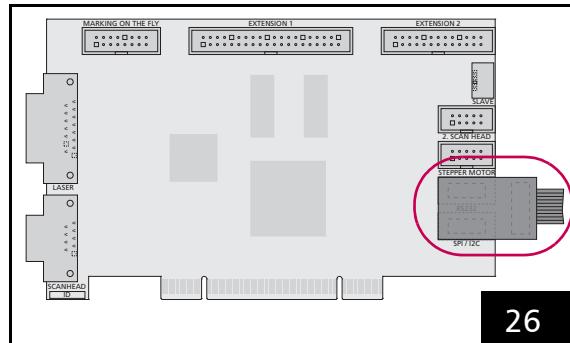
The voltages at both analog inputs are automatically (without any command call) converted in an endless series (duration approx. 0.3 ms) and transferred to the DSP as 12-bit digital values (I^2C bus with 400 kHz, fast mode).

The current input values can be read by the control command [read_analog_in](#) at any time.

As of version DLL 543, OUT 543, RBF 524 the analog input values (ANALOG IN0 and ANALOG IN1, see [read_analog_in](#)) can be recorded with trigger signal 54 (see [set_trigger](#) or [set_trigger4](#)). However, old bits are not recorded. The format of the data is ((ANALOG IN1 << 16) + ANALOG IN0).

Installation and Pinout

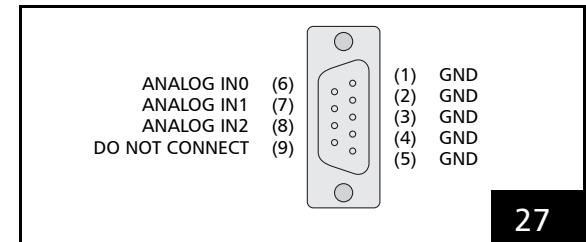
The ADC add-on board must be plugged into the "SPI / I²C" and "RS232" socket connectors of the RTC5 PCI Board. When installing, ensure that the add-on board is correctly oriented (see [figure 26](#)): the soldered-on flat ribbon cable must point outward.



ADC add-on board on RTC5 PCI Board

Internally, the ADC add-on board is only electrically connected to pins 1, 2, 9 and 10 of the "SPI / I²C" socket connector. Attachment to the "RS232" socket connector is only for mechanical stability.

The analog inputs are then available by a 9-pin D-SUB connector at the soldered-on flat ribbon cable. Its pinout is shown in [figure 27](#).



ADC add-on board: pinout of the 9-pin D-SUB connector

Specifications

- Input voltage range: 0 V ... 10 V
- Input impedance: 4 kΩ
- ADC resolution: 12 bit

The input signals are referenced to PC ground GND.

Notes

- The ADC add-on board even provides a third 10 V analog input ANALOG IN2 with identical specifications (see [figure 27](#)). However, the input values of this third analog input can neither be queried by [read_analog_in](#) nor recorded by [set_trigger](#) or [set_trigger4](#).

5 Installation and Start-Up

Installation of the RTC5 consists of the following steps (proceed in the described order):

- (1) Configuring the RTC5 jumpers (see [page 60](#))
- (2) Installing the RTC5 Board (see [page 60](#))
- (3) Installing the RTC5 drivers (see [page 61](#))
- (4) Installing the RTC5 software (see [page 62](#))
- (5) When turning on the laser system, observe the safe start-up sequence (see [page 63](#)).
- (6) The included HPGL converter program can be used for conducting a post-installation functionality test (see [page 63](#)).

5.1 Jumper Settings

SCANLAB ships RTC5s in various jumper configurations. The jumpers are soldered junctions and customers can subsequently reconfigure them by using a soldering iron (see [page 28](#)).

If you do not want to change the factory settings, proceed with [chapter 5.2 "Installing the RTC5 Board", page 60](#).



Caution!

- When altering a jumper configuration, take care not to damage the board's electronics!

5.2 Installing the RTC5 Board



Caution!

- Store the RTC5 in an electrostatically neutral environment using the supplied anti-static bag.
- Carry out the installation in an area that complies with Electro Static Discharge (ESD) directives. When installing the board, observe the instructions given in the instruction for use of the PC.
- Do not touch the contacts of the RTC5.
- Protect the board from humidity, dust, corrosive vapors and mechanical stress.

Perform the following steps to install the RTC5 in a PC:

- (1) Remove all diskettes and CDs from your PC.
- (2) Shut down the operating system and switch off the PC. Disconnect the PC from the mains.
- (3) Disconnect all peripherals (e.g. monitor, mouse, keyboard, printer etc.) from the PC.
- (4) Place the PC on a work table that complies with ESD directives.
- (5) Remove the housing of the PC. Locate a free PCI slot and remove the slot cover. A height from the slot connector's top of at least 105 mm is required.
- (6) Remove the RTC5 from the antistatic bag. Do not touch the contacts of the board.

- (7) If you want to use the signals on the RTC5's socket connectors, then attach the appropriate cables. SCANLAB recommends installing additional slot covers with suitable connectors, so that the signals are accessible even when the PC's housing is closed. All RTC5 connectors and signals are described in [chapter 4 "Layout and Interfaces", page 40](#).
- (8) Install the RTC5 into the PCI slot. Observe the instructions in the manual of your PC⁽¹⁾.
- (9) Close the housing of the PC.
- (10) Place the PC at its operational location and connect all peripheral equipment.
- (11) Connect the 9-pin D-SUB connector on the RTC5 – using the XY2-100 converter, if necessary – to the scan system by a data cable (see ["Interfaces to Scan System", page 42](#)).
- (12) Connect the 15-pin D-SUB connector on the RTC5 to the laser by a suitable interface (see ["Interfaces for the Laser and Peripheral Equipment", page 47](#)).
- (13) If necessary, connect one or more of the RTC5's socket connectors (using additional slot covers, see above) to the laser, a handling system or other supplementary equipment of the overall system.
- (14) Check all connections and turn on the PC.

(1) If multiple master/slave-synchronized RTC5 Boards are to be used in a PC, then the RTC5 Boards must first be connected pairwise with each other by the MASTER and SLAVE connectors and then installed in adjacent PCI slots. (see also ["Master/Slave Synchronization", page 41](#)).

5.3 Installing the Drivers

To install the RTC5 drivers for Windows 10, 8, 7, Vista, XP SP2, XP SP3, proceed as follows:

- ▶ After the RTC5 Board has been installed, start the computer.

For Windows 7, Vista, XP SP2 XP SP3:

- ▶ If the "Add Hardware Wizard" does not come up automatically, call it from the Control Panel.
- ▶ In the "Add Hardware Wizard" specify the directory that includes the RTC5 drivers. During installation, the operating system automatically selects the appropriate driver file (32- oder 64-bit).

For Windows 10, 8:

- ▶ Open the Device Manager to display the device tree. Look for the "PCI device" entry and update its driver by specifying the directory that includes the RTC5 driver.



Caution!

- The RTC5 Board does not support power-saving modes that switch off power to the PCI bus. Accordingly, you must disable standby or sleep modes of the operating system. Particularly disable the Windows sleep mode option (some Windows operating systems enable this option by default). See also the note on [page 26](#).



5.3.1 Software Package Upgrades

If you currently use an RTC5 software package version 2012_09_05 or earlier (still contains WDM driver) and you want to upgrade to an RTC5 software package version 2013_02_21 or later (contains already WDF driver), then proceed as follows:

- (1) First install only the RTC5 software files, see [chapter 5.4 "Installing the RTC5 Software", page 62.](#)
- (2) Test the proper functioning of your user program.
- (3) Only afterwards install the new driver (this is a WDF driver). This is because quickly changing back to older RTC5 software files is then only possible if you also change the driver (see also note on [page 26](#)).
- (4) Right-click on the delivered script `ScanlabClassChecker.cmd` and choose 'Run as Administrator' from the appearing list. For further information, refer to the file `ReadMe_ScanlabClassChecker.pdf`.

5.3.2 Exchange of RTC Boards and Update of RTC Board Drivers

After the exchange of any RTC board of type RTC3, RTC4, RTC SCANalone, RTC5 for an RTC5 (also: RTC6 PCI Express Board) or, to update an (outdated) WDM driver to a (newer) WDF driver proceed as follows:

- (1) Install the latest driver for the new board (this is a WDF driver; downloadable from the SCANLAB homepage).
- (2) Right-click on the delivered script `ScanlabClassChecker.cmd` and choose 'Run as Administrator' from the appearing list. For further information, refer to the file `ReadMe_ScanlabClassChecker.pdf`.

5.4 Installing the RTC5 Software

Additionally to installing the drivers, the RTC5 software must be installed (copied or unzipped onto the PC's harddisk). The following files are required:

- RTC5DLL.dll or RTC5DLLx64.dll (Win32- or Win64-based user program-support DLL file)
- RTC5OUT.out (DSP program file)
- RTC5RBF.rbf (firmware file)
- RTC5DAT.dat (binary support file)
- (*.CT5) (correction file/s)

The files are included in the RTC5 software package. For easy identifying and archiving of different software versions, some of the files are also delivered zipped (the zip file names RTC5<..>_<Version>.zip include the version numbers).

To install the RTC5 software, follow these steps:

- ▶ Copy or unzip the files RTC5DLL.dll (or RTC5DLLx64.dll), RTC5OUT.out, RTC5RBF.rbf and RTC5DAT.dat (of desired version) to the harddisk of your PC⁽¹⁾. When replacing individual software files, note that differing program versions cannot be arbitrarily combined with another (each zip file includes a text file with corresponding version information).
- ▶ Copy the correction file(s) (*.CT5) to the harddisk of your PC (existing correction files can still be used; do not overwrite customized correction files!).

SCANLAB recommends storing these files in the directory in which the user program is started. When loading the correction files, you then only have to provide the filename without needing to specify a path.

(1) As of version DLL 527, MS Visual Studio runtime libraries for C and C++ are no longer necessary.

5.5 Safe Start-up and Shutdown Sequences

To assure safety during start-up, turn on the components of your laser system exactly in the following order:

- (1) Turn on the PC containing the RTC5 Board and start up the control software.
- (2) Turn on the desired peripheral equipment.
- (3) Turn on the power supply for the scan system.
- (4) Turn on the laser.

When shutting down the laser system, turn off the components exactly in reverse order:

- (1) Turn off the laser.
- (2) Turn off the scan system's power supply.
- (3) Turn off the peripheral equipment.
- (4) Close the control software and turn off the PC containing the RTC5 Board.



Danger!

- While the PC is turned on and off, short-term level variations at the RTC5 Board's output ports, for instance short-term arbitrary variations of the laser control signals can occur. Therefore always observe the above listed start-up and shutdown sequences. Otherwise there is the risk that the laser unexpectedly turns on for a short term.
- Always turn on the scan system after the PC and control software and turn it off prior to the PC and control software. Otherwise arbitrary scan movements of the scan system could occur. Always turn on the laser as the last component and turn off the laser as the first component. Otherwise there is the risk that the laser beam might be deflected in an uncontrolled direction.

5.6 Functionality Test



Danger!

- Always turn on the PC and the power supply for the scan head first before turning on the laser. Otherwise there is the danger of uncontrolled deflection of the laser beam. SCANLAB recommends the use of a shutter to prevent uncontrolled emission of laser radiation

The HPGL conversion program `HPGL.EXE` is supplied for testing control of the scan head, see also section "HPGL Converter Program", page 22. This program lets you load graphics files in Hewlett Packard's HPGL format (vector graphic plotter files *.`PLT`) for transfer to the RTC5.

- (1) Copy the HPGL converter and the supplied demo file into the same directory as the RTC5 DLL, the RTC5 DSP program file(s) and correction file(s).
- (2) Start the HPGL converter.
- (3) Under Options | Correction select a correction file and under File | HPGL-File select a plot file (demo file).
- (4) Start output by Mark | Start Marking.

5.7 User Programs and Demo Software

The DLLs for RTC5 user programs (`RTC5DLL.dll`, `RTC5DLLx64.dll`) support the RTC5 under 32-bit and 64-bit Microsoft operating systems Windows 10, 8, 7, Vista, XP SP2, XP SP3. The DLLs provide all required functions for operating the RTC5.

To integrate the desired DLL (Win32- or Win64-based, see below) into a user program, call the desired DLL and initialize its functions. Programming of applications is described in detail in [chapter 6](#). [Chapter 6.2, page 66](#) describes the calling convention of the DLL and shows how to initialize its functions and procedures in applications written in Pascal, C, C++ or C#.

On a 64-bit operating system, the 64-bit variant of the RTC5 driver supports function calls from Win64-based applications as well as from Win32-based applications⁽¹⁾.

Therefore, existing Win32-based applications for the RTC5 are able to execute even on 64-bit systems, if the included Win32-based DLL file `RTC5DLL.dll` is used. For Win64-based applications, the Win64-based DLL file `RTC5DLLx64.dll` is included in the software package. In case a user program utilizes implicit linking to the `RTC5DLLx64.dll`, it must be linked with the Win64-based import library `RTC5DLLx64.Lib`.

To help programmers get started, some example codes are listed on [page 71](#), [page 100](#) and [page 109](#). In addition the RTC5 software package includes a variety of demo programs (see [page 647](#)).



Caution!

- Carefully check your user program before running it. Programming errors can cause a system breakdown. In this case, neither the laser nor the scan head can be controlled.

5.8 Exchanging RTC5 Boards

If an already installed RTC5 board of an older type is replaced by an RTC5 board of a newer type, then it is recommended to carry-out a software update.

This is particularly important when RTC5 Boards with DSP version 0 get replaced by RTC5 Boards with DSP version 2 (see `get_RTC_version` bits #16-23). The suitable RTC5 software package can be found on the delivered CD. You can also download the latest files From the SCANLAB website you can also download the latest RTC5 software package.

See also [chapter 5.3.2 "Exchange of RTC Boards and Update of RTC Board Drivers", page 62](#).

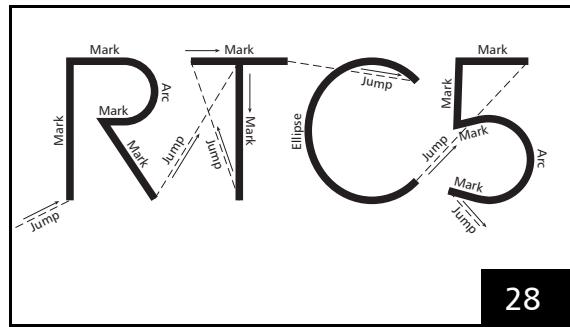
(1) The driver differentiates whether a function call originates from a 32-bit or a 64-bit-DLL.

6 Developing User Programs

6.1 RTC5 Software Basics

6.1.1 Controlling Scan Systems and Lasers – An Introductory Example

The SCANLAB RTC5 PCI Board and its related software are designed for controlling scan systems and lasers. To illustrate the principle of operation, figure 28 shows a simple laser marking sample⁽¹⁾.



A laser marking sample

The lettering is made up of straight line segments (vectors) and arc segments. The RTC5 DLL provides a set of jump, mark, arc and ellipse commands for laser processing along such segments. Each of these commands describes one vector or arc. Additional software commands are available for controlling the laser during the marking process. The RTC5 processes the commands it receives and precisely transmits the required marking signals to the scan head (using a pre-defined 10 µs time raster) and to the laser. The scan head's galvanometer scanners accurately position their deflection mirrors in synchronization with the incoming control signals.

Appropriate RTC5 commands also allow querying of the current scan head status information.

For laser control, the RTC5 provides various analog and digital signal outputs freely available for tailoring laser control to customer-specific requirements. The customer assumes responsibility for use of these signals.

6.1.2 List Commands and Control Commands

The RTC5 command set consists of *control commands* and *list commands*.

Control commands are executed immediately. They are used, for instance, for initializing, controlling execution of lists, setting some general parameters, or for directly controlling the laser and scan head.

Before **list commands** can be sent to the RTC5, a control command must define the input pointer to which subsequent list commands are transferred (this corresponds to opening a list, see [chapter 6.4.1 "Loading Lists", page 75](#)). List commands sent to the RTC5 afterwards are not executed immediately, but stored in a list buffer. The RTC5 begins reading the commands from the list buffer and processing them in real time only after the list is started.

Some list commands, which are called **short list commands**, do not require the full 10 µs clock period for command execution - instead, they execute along with the next list command, one directly after the other within a single 10 µs clock period, during which control commands cannot execute. The total list processing time is thereby reduced.

List commands include jump commands, mark commands and arc commands, as well as commands for setting various scanning parameters such as laser power, jump speed and marking speed.

Short list commands include **list_jump_pos**, **list_call** etc. With polylines, for example, the laser power can be set individually for each vector by the short list command **write_da_x_list** without interrupting the polyline (the laser remains on).

(1) In this manual, laser marking is mentioned only as an example of the many possible laser materials processing applications.

Some commands exist in two versions: as a list command and as a control command. Among these dual-version commands are the I/O commands.

All RTC5 commands are described in detail in [chapter 10 "Commands and Functions"](#). An overview is provided in [chapter 10.1, page 250](#).

6.2 Initialization and Program Start-Up

To use the RTC5's commands and functions in a user program:

- The RTC5 must be fully installed – this includes the RTC5's hardware, driver and software (see ["Installation and Start-Up", page 60](#))
- The desired DLL (Win32- or Win64-based) must be called by the user program (see ["DLL Call", page 66](#))
- The DLL's functions and procedures must be initialized in the user program (see ["Importing Commands", page 66](#))

At the beginning of each user program, commands must be called:

- that initialize the DLL for the calling user program and assigns access rights for the installed RTC5 Boards (see ["Initializing the DLL and Board Management", page 68](#))
- that place the installed RTC5 Boards into the desired default state (see ["Start of Operation", page 69](#)).

Only afterward can the user program load its command lists into the RTC5's list memory and start processing.

These steps are described individually in the following sections and summarized by a simple example program in the last section of this subchapter (see [page 71](#)).

6.2.1 DLL Call

If a user program is to use the DLL's commands and functions, then the desired DLL (Win32- or Win64-based) must be made available (as described in ["Installation and Start-Up", page 60](#)) and called by the user program.

The DLL calling convention is **stdcall**. The structure alignment is 4 byte (for Win32) or 8 byte (for Win64).

6.2.2 Importing Commands

To facilitate importing the commands of the DLL into a C, C++, C# or Pascal user program, the RTC5 software package contains corresponding utility files.

The following sections describe how to use these files in your particular software environment⁽¹⁾.



Caution!

- Some of the commands included in the utility files can only be used with the appropriate optional hardware configuration of the RTC5 – see also [chapter 2.3 "Optional Functionality", page 27](#).

Make sure to use only the commands supported by your version of the RTC5. Refer to the descriptions of the individual commands in [chapter 10](#).

The command **get_RTC_version** provides information about the options currently installed on your RTC5 Board.

Pascal

Use the file **RTC5Import.pas** as a unit and call the RTC5 commands you need, like

```
goto_xy(1000, 2500);
```

for performing a jump to location 1000, 2500.

⁽¹⁾ Other than with implicit linking under C/C++, the way to proceed does not depend on which DLL is used (Win32- or Win64-based). Nevertheless, you must always comply with the formats and value ranges of data types used in RTC5 commands (see [page 260](#)).

C

In C, you can choose either implicit linking – also known as static load or load-time dynamic linking – or explicit linking – also known as dynamic load or run-time dynamic linking.

Implicit Linking

To accomplish implicit linking, include the header file `RTC5impl.h` and link with the (C) import library `RTC5DLL.LIB` (for Win32-based applications or with `RTC5DLLx64.LIB` for Win64-based applications) for building the executable.

Call the RTC5 commands you need like

```
goto_xy(1000, 2500);
for causing a jump to location 1000, 2500.
```

Explicit Linking

To accomplish explicit linking, include the header file `RTC5expl.h`. Before calling any RTC5 function, initialize the DLL by calling the function `RTC5open` (which is defined in the file `RTC5expl.c`).

When you are finished using the RTC5, close the DLL by calling the function `RTC5close` (also defined in the file `RTC5expl.c`).

For building the executable, link with the file `RTC5EXPL.OBJ`, which you can generate from the source code `RTC5expl.c`.

Call the RTC5 commands you need, like

```
goto_xy(1000, 2500);
for causing a jump to location 1000, 2500.
```

Pros and Cons of Implicit and Explicit Linking

	Implicit Linking	Explicit Linking
Necessary Files	<code>RTC5impl.h</code> or <code>RTC5impl.hpp</code> , <code>RTC5DLL.LIB</code> or <code>RTC5DLLx64.LIB</code>	<code>RTC5expl.h</code> , <code>RTC5expl.c</code>
Advantage	Easiest linking method	Eliminates the need to link the user program with an import library
Negative Aspect	Need to link the user program with a compiler-specific import library	Need to initialize (<code>RTC5open</code>) and close (<code>RTC5close</code>) the DLL

C++

If you want to use references instead of pointers for returning values to function parameters in C++ (for instance `UINT&Pos` instead of `UINT*Pos`), use the files `RTC5impl.hpp` instead of `RTC5impl.h`. Otherwise the command `import` is realized as in C (see above).

C#

To accomplish implicit linking in C#, import declarations of the wrapper class are provided.

As of DLL 537: The wrapper class also supports the 'Any CPU' option for simultaneous usage with 32-bit and 64-bit programs.



6.2.3 Initializing the DLL and Board Management⁽¹⁾

Any number of RTC5 Boards can be used simultaneously in one PC. Moreover, the RTC5 DLL allows multi-threading as well as multi-processing. Therefore, also any number of applications (user programs) can be used simultaneously.

However, no board can be simultaneously used by multiple applications. Access rights (even if temporary) to existing boards are assigned on an exclusive basis by the DLL. Multiple threads of one user program can use the same board, but can not send commands to it at the same time (the DLL automatically serializes the command calls).

DLL-internal board management coordinates usage of different boards by different applications. Board management is initiated by the command **init_rtc5_dll**. This command is a prerequisite for RTC5 access and must be called at the beginning of every user program – even if only one RTC5 Board is to be used by only one user program.

The command **init_rtc5_dll**:

- searches for all existing RTC5 Boards
- establishes corresponding board management
- automatically assigns the user program access rights to the found boards (as long as access rights are not already assigned to another user program)
- assigns DLL-internal numbers for all found RTC5 Boards (important for multi-board commands)
- sets one board as the active board, which is the target for non-multi-board commands

For a detailed description, see [page 348](#).

Further commands enable subsequent changes to access rights and changing the active board. Usage of multiple boards is described in [chapter 6.6 "Using Multiple RTC5 Boards in One Computer"](#), page 92; usage by multiple applications is described in [chapter 6.7 "Usage by Multiple Applications"](#), page 96.

After DLL initialization by **init_rtc5_dll**, users can additionally select one of two operation modes (see [set_rtc4_mode](#) and [set_rtc5_mode](#)). The default is RTC5 mode. An RTC4 compatibility mode is also provided so that applications written for the RTC4 can be processed by the RTC5 (to a large extent) without modification. However, a prerequisite here is that the program can only contain RTC4 commands that also exist with unchanged functionality as RTC5 commands. This user manual's list of commands ([page 261](#)), when applicable, notes such changes in the "RTC4→ RTC5" section (see also [section "Increased Parameter Resolution"](#), page 33).

(1) The usage of all RTC5 commands is identical under both the 32-bit and 64-bit versions of Windows. Nevertheless, you must always comply with the formats and value ranges of data types used in RTC5 commands (see [page 260](#)).

6.2.4 Start of Operation

To start RTC5 operation after successful initialization of the DLL (see preceding section), the following steps should be carried out at the beginning of every RTC5 user program:

Initialization of the Board

(1) Call the command **load_program_file**. This command resets the RTC5 (the equivalent of a hardware reset), performs a DSP memory check, initializes the memory configuration (in the default configuration), loads the firmware (RTC5RBF.rbf), the program file RTC5OUT.out and a binary support file (RTC5DAT.dat), and starts the signal processor. After execution of this command, the position of the scanner is automatically set to the null position (0|0) (i.e. the laser focus is centered in the image field) and laser control is *deactivated*.

Assorted versions of the RTC5 DLL and the program files RTC5OUT.out, RTC5RBF.rbf and RTC5DAT.dat cannot be arbitrarily combined with another. The command **load_program_file** performs a version compatibility check. If a version error exists (error code 7 and **get_last_error** return code

RTC5_VERSION_MISMATCH), the board is released by **release_rtc** and thus is not available for further commands other than those not requiring access rights. RTC5 Boards can only be operated if all software files are available with a compatible combination of versions (see chapter 5.4 "Installing the RTC5 Software", page 62).

If the version compatibility check detects an error and the board's access rights are not assigned to another user program, the multi-board command **n_load_program_file** can be used to load a correct program version, followed by **select_rtc** or **acquire_rtc** to access the board.

If multiple RTC5 Boards are connected as master and slave, then the command **load_program_file** must be executed on all boards and a clock phase synchronization should be performed prior to initializing and operating the individual boards with further commands (see chapter 6.6.3 "Master/Slave Operation", page 93).

Configuration of the Board

(2) If necessary, configure the RTC5's list memory by **config_list**. By default SCANLAB preconfigures the list memory such that "List 1" and "List 2" can each store 4,000 list commands. The protected memory area, "List 3", receives the remaining 1040576 of the 2^{20} storage positions.

Initialization of the Scan System Control

(3) Use **load_correction_file** to download the necessary correction file(s) to the RTC5 (you can load a correction table before or after **load_program_file**⁽¹⁾ but you should load it before **select_cor_table**; you should at least load a 1:1 correction table). See chapter 8.5, page 192, for information about using several different correction tables.

(4) Assign the previously loaded correction table(s) to the scan head control port(s) by **select_cor_table**⁽²⁾. This causes the intended image field correction to also be applied to the default scanner position (0|0) previously set by **load_program_file** (in some circumstances, image field correction can even shift the null position).

After the command **load_program_file**, the default assignment is:

- The first scan head control port uses correction table #1.
- The second scan head control port is off. The desired image field correction becomes active only after a subsequent **select_cor_table**, **select_cor_table_list**, jump or marking command.

(5) Define the scanner delay mode (for instance variable polygon delay or constant polygon delay; command **set_delay_mode**).

(6) Load a table for the variable polygon delay if necessary (command **load_varpolydelay**).

The remaining settings (scanner delays, jump speed and marking speed) are set by further control commands or list commands.

(1) **load_program_file** deletes correction tables number 3 and 4.

(2) As of Version DLL 521, OUT 521, **load_correction_file** automatically calls **select_cor_table** after loading the correction table (if the command follows **load_program_file**) (see notes on page 138).



Initialization of the Laser Control

- (7) Set the laser mode
(command `set_laser_mode`).
- (8) Set the laser control signals appropriate to your laser system (command `set_laser_control`).
- (9) Set the FirstPulseKiller length (YAG only)
(command `set_firstpulse_killer`).
- (10) Set the delay of the Q-Switch signal (only for YAG modes, in particular for YAG mode 5)
(command `set_qswitch_delay`).
- (11) Set the stand-by pulses (usually CO₂ only)
(command `set_standby`).
- (12) Enable the laser control signals (see command `enable_laser`), if they have been disabled by `set_laser_control`.

The remaining settings (laser timing or laser delays) are set by further control commands or list commands, see example in chapter 7.1.4 "Example Code", page 109 or section ""Laser Active" Signals", page 145.

Loading and Executing Lists

- (13) Load the list(s).
- (14) Enable the external start input if necessary
(command `set_control_mode`).



Caution!

- Carefully check your user program before running it. Programming errors can cause a break down of the system. In this case, neither the laser nor the scan head can be controlled.



6.2.5 Example Code

The following C source code for a console user program (environment: Win32) illustrates the programming fundamentals of DLL and RTC5 initialization (for complete demo programs, see [page 647](#)).

Necessary sources: RTC5impl.h, RTC5DLL.LIB (for implicit linking) or RTC5expl.h, RTC5expl.c (for explicit linking). Additionally, the DLL must be called by the program (see [page 66](#)). If the operating system does not find the RTC5DLL.dll on program startup, it produces a corresponding error message and terminates the program.

```
// System header files
#include <windows.h>
#include <stdio.h>
#include <conio.h>

// RTC5 header file for implicitly linking to the RTC5DLL.dll (for building the executable, also link with the (Visual C++) 
// import library RTC5DLL.LIB):
#include "RTC5impl.h"
// Alternatively: RTC5 header file for explicitly linking to the RTC5DLL.dll (for building the executable, link with the file
// RTC5EXPL.OBJ, which you can generate from the source code RTC5expl.c):
// #include "RTC5expl.h"

void __cdecl main( void*, void* )
{

    // only for explicitly linking:
    // if ( RTC5open() ) // error detected, RTC5open returns 0 for no error
    // {
    //     printf( "Error: RTC5DLL.dll not found\n" );
    //     terminateDLL();
    //     return;
    // }

    printf( "Initializing the DLL\n\n" );

    UINT ErrorCode;

    // Initializing the DLL (the following command must be called as the first RTC5 command)
    ErrorCode = init_RTC5_dll();

    // Following the init_RTC5_dll command you should include a program code to catch an error during initialization, e.g.
    // for the case the desired RTC5 Board is not detected, access is denied or for another error (e.g. a version mismatch).
    // A corresponding example code is listed on chapter 6.8.3 "Example Code", page 100.

    // Initializing the RTC5:
    // - Selecting the RTC5 Board number 1 as the active RTC5 Board for this user program
    // - If desired: Selecting the RTC4 compatibility mode as operation mode (default: RTC5 mode)
    // - Stopping any list running on RTC5 Board number 1 (if this board has been used previously by another user program,
    //   a list might still be running. This would prevent load_program_file and load_correction_file from being executed).
    // - Calling load_program_file for resetting the board, loading the program file, etc. (here also a program code should be
    //   included to catch possible errors – e.g. file or system errors – during initialization;
    //   see chapter 6.8.3 "Example Code", page 100)
    // - Clearing all previous error codes (stop_execution might have created an RTC5_TIMEOUT or RTC5_BUSY error)
    // - Configuring the RTC5 Board's list memory, default: 4000 storage positions for list 1, 4000 for list 2.
}
```



```
(void) select_rtc( 1 );
set_rtc4_mode();
stop_execution();
ErrorCode = load_program_file( 0 ); // pPath = 0: path of current working directory
if ( ErrorCode )
{
    printf( "Program file loading error: %d\n", ErrorCode );
    free_rtc5_dll();
    return;
}
reset_error( -1 );
config_list( 4000, 4000 );

// Following the above initialization code you can include the program code defining the laser scan process.
// An example code is listed on page 109

// End of main program
terminateDLL();
return;
}

void terminateDLL()
{
    printf( "- Press any key to shut down \n" );
    while( !kbhit() );
    (void) getch();
    printf( "\n" );
    // Close the RTC5.DLL
    free_rtc5_dll();
    // only for explicitly linking:
    // RTC5close();
}
```



6.3 List Memory

List memory serves as intermediate storage for list commands. Before list commands can be sent to the RTC5, a control command must define the input pointer to which subsequent list commands are transferred (this corresponds to opening a list, see "[Loading Lists](#)", page 75). Additional control commands start the processing of transferred list commands.

6.3.1 Lists and the Protected Buffer Area

The RTC5's list memory contains 1M (= $1048576 = 2^{20}$) storage positions. The memory is basically divided into three areas of variable (configurable) size:

- Two list buffers ("List 1" and "List 2"), later simply referred to as "lists"
- A third "protected buffer area" ("List 3"), which is protected against unintended overwriting during loading of normal command lists

Lists

In principal, both list buffers ("List 1" and "List 2") can be used in a manner identical to the two list buffers of the RTC4, RTC3 or RTC2 Boards, e.g. for continuous loading and processing of command lists. But the RTC5 provides significantly more memory, and the size of each buffer area is freely configurable (see "[Configuring the List Memory](#)", page 74).

Plus, the RTC5 has enhanced command functionality for list handling, as detailed in [chapter 6.4 "List Handling"](#), page 75.

Protected Buffer Area

The third buffer area ("List 3") – an RTC5 innovation – is a protected memory area intended for storing frequently needed list-command sequences as subroutines or character set definitions. This area is protected against unintended overwriting during loading of normal command lists.

There are principally two ways to utilize this protection feature:

- (1) As with the RTC5's predecessors, subroutines can be written to the upper positions of the list buffers. What is new with the RTC5 is that these subroutines can be subsequently assigned to the protected buffer area. Such subroutines are called – both initially in the list-buffer area as well as subsequently in the protected buffer area – by specification of an absolute memory location.
- (2) Special commands allow subroutines and character set definitions to be loaded directly in the protected buffer area as indexed subroutines or definitions. They can then be called by providing the corresponding index, thus potentially simplifying later usage. Indexed character set definitions can, for example, be used in conjunction with the newly introduced `mark_text` command for directly marking text.

SCANLAB strongly recommends *not* intermixing usage of these two methods. Otherwise, unintended data loss by overwriting could occur even in the protected buffer area.

The RTC5 command set includes appropriate conversion commands so that you are not forced to continuously use only one of the two methods.

The defining of subroutines and character sets, as well as their management and subsequent conversion options are detailed in [chapter 6.5 "Structured Programming"](#), page 81.

6.3.2 Configuring the List Memory

For compatibility, SCANLAB preconfigures the RTC5's list buffer area such that "List 1" and "List 2" can each accept 4,000 list commands.

The protected "List 3" then owns the remaining 1040576 of the 2^{20} storage positions (see Configuration (a) in [figure 29](#)). Users can also reconfigure the buffer area by the command `config_list`.

For example, if a user program only needs one list, then the RTC5's entire memory can be treated as a single list with a total capacity of 2^{20} positions (see Configuration (e) in [figure 29](#)). In this case, the single list ("List 1") can be loaded with up to 2^{20} commands.

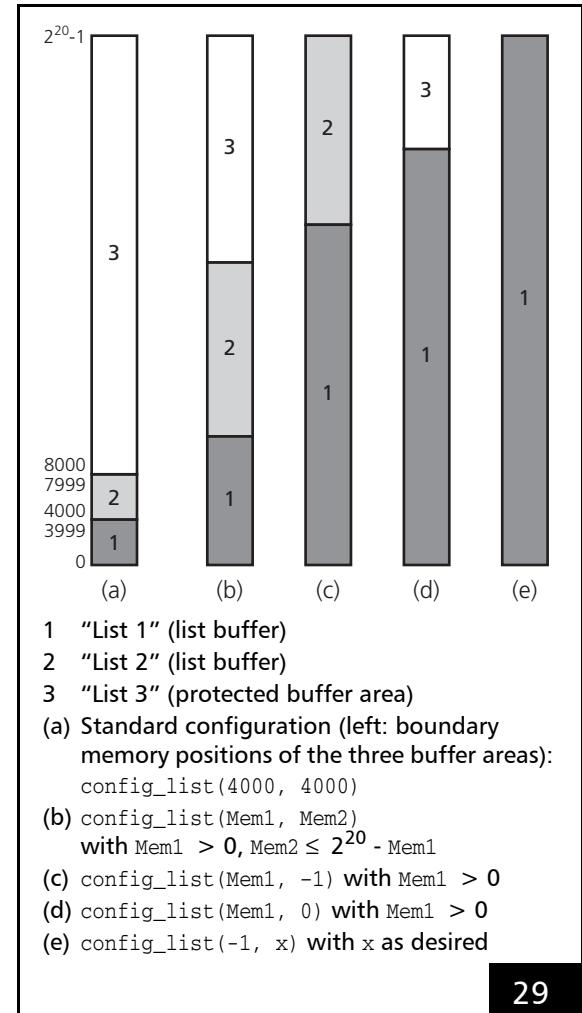
The buffer areas are generally configured so that "List 1" occupies the lower positions, "List 2" the intermediate positions, and "List 3" the uppermost positions.

When configuring the buffer areas, note that "List 1" must contain at least one storage position and that the total sum of storage positions for "List 1" and "List 2" must not exceed 2^{20} . Other than that, the sizes of "List 1" and "List 2" can be configured as desired (e.g. 0 positions for "List 2", see Configuration (d) or (e) in [figure 29](#)). During configuration, the protected "List 3" buffer area automatically receives remaining memory not assigned to "List 1" or "List 2".

The configuration process does not alter the contents of the buffer memory. Repeating the call with differing parameters is therefore nondestructive.

When subsequently altering the configuration, you should observe the following:

- List boundaries should not be moved to within an eventual subroutine.
- If a "List 3" region is assigned to the list area ("List 1" or "List 2"), the region's protection is removed.
- Valid jump addresses specified in jump commands might become invalid if the configuration is altered (see "[Jumps](#)", page 89).
- If the protected buffer area ("List 3") is made larger, defragmentation might be needed to make the newly assigned memory area usable (see "[Index Management and Defragmentation](#)" on page 85).



29

Examples of valid list buffer configurations



6.4 List Handling

The two list areas ("List 1" and "List 2") serve as intermediate storage for the continuous loading and processing of list commands. This chapter describes the commands that control this data transfer.

6.4.1 Loading Lists

The commands `set_start_list_pos`, `load_list` and other control commands (see below) are used to enable a list to be filled with list commands. An input pointer is thereby defined for the selected list. This input pointer specifies the memory position to which the subsequent list commands are transferred.

Lists are self-contained memory blocks for list commands. Accordingly, if a list is loaded and the end of the list is reached without setting the input pointer to another list, then the input pointer is automatically reset to the start of the current list, where loading continues. An automatic change of the input pointer to another list never occurs, particularly not to the protected buffer area ("List 3").

When list commands are loaded into list buffer positions, any commands previously stored there are thereby overwritten, even if they have not yet been processed or are currently being executed. Be sure not to overwrite commands still needed by your user program (see below).

PCI transfer of the list commands into list memory is buffered to increase the speed for continuous downloads. The buffer's capacity is 16 commands.

Whenever the buffer is full or when the commands `set_end_of_list`, `list_return`, `set_input_pointer` (and related commands), `execute_list_pos` (and related commands), `auto_change`, `auto_change_pos`, `start_loop` or `release_rtc` are issued, this automatically results in a flush, so that the buffered list commands are transferred to list memory. Flushing of an incomplete buffer can also be initiated at any time by `set_input_pointer(get_input_pointer())`. This is only necessary in some circumstances when list commands should be processed and list input is not yet finished (e.g. for external starts).

"Unconditional" Loading

The command `set_start_list_1/_2` sets the input pointer to the start of the selected list and the command `set_start_list_pos` or `set_input_pointer` sets the input pointer to the specified address of the selected list. Regardless of the selected list's current status (see "[List Status](#)", page 76), the next list command is written to this address.

If needed, the current positions of the input and output pointers can be queried by the commands `get_input_pointer` or `get_list_pointer` and `get_status` or `get_out_pointer` - e.g. to ensure that not-yet-processed list commands are not overwritten.

Loading with Protection

The loading process is initialized by the command `load_list`, which sets the input pointer to the specified address in the selected list (just like the `set_start_list_pos` command), but only if the selected list is not currently in use. Alternatively, you can simply let the input pointer be set to a currently non-active or already processed list by `load_list` (the RTC5 automatically determines the corresponding appropriate list).

The return value of the `load_list` command reveals if, and in which list, the loading procedure was successfully initialized. Otherwise, the input pointer is set to an invalid position, in which case no further list commands can be input until the input pointer is correctly set (e.g. by repeating the `load_list` command with a positive result or by the `set_start_list_pos` command).

This automatically prevents unintentional overwriting of not-yet-executed commands.

The `load_list` command is useful in scenarios such as alternating list changes, where you want to wait specifically for a list to be processed (see "[Alternating List Changes](#)", page 80).



Terminating Lists

Command lists can be, but need not necessarily be, terminated by the `set_end_of_list` command.

However, if an unterminated command list is executed and the output pointer thereby encounters the last possible position in the list, the output pointer automatically resets to the start of the list and processing continues there. Automatic list changing after a list is processed can only occur if the list was terminated by the `set_end_of_list` command (see "Automatic List Changing", page 79).

The loading of a `set_end_of_list` command does *not* stop the loading procedure itself. Additional commands in the same command list can be further loaded directly after the `set_end_of_list` command.

6.4.2 List Status

Dependent on the command input and output statuses, lists can receive particular status values. To examine the current statuses of the lists, the control command `read_status` can be used – separately for both lists.

- A list's LOAD status (LOAD1 or LOAD2) indicates that the input pointer is currently in this list. The LOAD statuses of the other lists are then *not* set.
- A list's READY status (READY1 or READY2) is set when the `set_end_of_list` command is written during the loading procedure. It is reset when the list's LOAD status is newly set.
- A list's BUSY (BUSY1 or BUSY2) status indicates that the output pointer is currently in this list after list execution (of "List 1" or "List 2") was started. The BUSY statuses of the other lists are then *not* set. Execution of the `set_end_of_list` command causes a reset of the BUSY status (or alternating setting, if automatic list changing was previously activated). If a list is opened for loading while still being processed, its BUSY status still remains set.
- A list's USED status (USED1 or USED2) is set when the `set_end_of_list` command is reached during processing. It is reset when the list's LOAD status is set.



Notes

- If the list status is queried during processing of a subroutine in the protected buffer area ("List 3"), then the status is returned of the list ("List 1" or "List 2") in which the output pointer most recently resided (typically from where the subroutine was originally called).
- If list execution is interrupted (by `pause_list`, `stop_list` or `set_wait`), then the above-mentioned status values remain unchanged.
- If list execution is aborted (by `stop_execution` or an external list stop), the USED status is set for both lists (as by initialization). See also `load_list(ListNo=3)`.
- If you want to explicitly set the USED status for a list `ListNo` (e.g. after abortion by `stop_execution` or an external list stop), then load a `set_end_of_list` command to a free position `Pos` of this list by `set_start_list_pos(ListNo, Pos)` and execute by `execute_list_pos(ListNo, Pos)`. If no other list has been active at this moment, then list `ListNo` has the status USED afterward.
- When interpreting the status values returned by `read_status`, always take into account the programmed loading or execution processes of the lists (see command description).

6.4.3 List Execution Status

The provided list status values are supplemented by three additional status values that indicate whether one of the two lists is currently being executed (BUSY), if its execution is currently interrupted (PAUSED) or if the RTC5 is busy with executing control commands beyond list execution (INTERNAL-BUSY). These status values can be queried by the `get_status` command.

- The BUSY status is set when a list is currently being processed or when a list has been halted by the control commands `pause_list` or `stop_list`. In contrast, the BUSY status is not set if a list has been paused by the list command `set_wait` (and is then again set by a subsequent `release_wait` command).
- The PAUSED status is set when a list has been halted by `pause_list`, `stop_list` or `set_wait`. It is reset by a subsequent `restart_list` or `release_wait` command (see also "Interrupting Lists for Synchronization of Processing", page 79).
- The INTERNAL-BUSY status is set when the RTC5 is busy with executing a control command, which needs more than 10 µs for executing a scan movement (e.g. `goto_xy` or possibly `set_offset`) or while a home jump or home return is executed (with `set_wait`, `set_end_of_list` or `release_wait`, if the home jump mode has been previously activated by `home_position` or `home_position_xyz`). The INTERNAL-BUSY status and the BUSY status cannot be set at the same time.

Notes

- The BUSY status is also available as the BUSY OUT signal at the 15-pin D-SUB LASER connector (see [figure 10](#)), and the EXTENSION 1 socket connector (see [figure 15](#)) as well as the MARKING ON THE FLY socket connector (see [figure 18](#)).
- Some control commands are ignored (not executed), when the BUSY status and/or INTERNAL-BUSY status are set (e.g. `auto_cal`, `goto_xy`, `load_correction_file`) or – with set INTERNAL-BUSY status – are only executed with delay after the INTERNAL-BUSY status has been reset again (e.g. `execute_list_pos`, `set_offset`).



6.4.4 Starting and Stopping Lists

List processing ("List 1" or "List 2") can be started either by the control command `execute_list` or by an external start signal (see the section "[Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization](#)", page 239).

The command `execute_at_pointer` can be used to start output of a list at a specified address. If an external start signal is used (see the section "[Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization](#)", page 239), then the command `set_extstartpos_list` allows definition of a start address for external starting of the list.

Output by the RTC5 begins immediately. Even during 10 µs-clocked execution of the list commands, you can still send control commands to the RTC5, which immediately executes them without hindering execution of the list. This is useful, for example, for loading a second list while the first list is being processed (the PC and scan head then work in parallel). But the second list can only be started after the first list has finished processing. During list processing, `execute_list` commands and external start signals are ignored.

Execution of a list can also be stopped at any time, e.g. for implementing an emergency shutdown or for any other purpose. Use of the `stop_execution` command (see [page 596](#)) immediately aborts the currently running list and turn off (but not deactivate) all "laser active" laser control signals. The `range_checking` command can be used to cancel a list execution automatically (as with `stop_execution`) as well.

If, during list processing, the end of the list is reached without encountering a `set_end_of_list` command, then processing continues at the beginning of the list. This is repeated until either the `stop_execution` command is called or an external stop signal is transmitted to the RTC5.

If, on the other hand, a `set_end_of_list` command is encountered during list processing, then list execution stops – unless an `auto_change`, `auto_change_pos` or `start_loop` command was previously issued. In the latter case, a list change takes place (see "[Automatic List Changing](#)", page 79). The change occurs only upon reaching a `set_end_of_list` command.

Notes

- Lists are not automatically started. Regardless of how many commands are loaded, a list must be started as described above in order to be processed.
- To also enable starting and stopping of a list by external signals, the RTC5 provides corresponding control inputs (see "[Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization](#)", page 239).



6.4.5 Interrupting Lists for Synchronization of Processing

The list command **set_wait** makes it possible to set *wait markers* (break points) within a list. Each marker is associated with a number greater than zero.

When the RTC5 reaches a wait marker during list execution (see "Structured Programming", page 81), processing of the list is temporarily interrupted and the laser is turned off.

The command **get_wait_status** ascertains whether processing is currently interrupted at a marker. If processing is interrupted, the command **get_wait_status** returns the number (*wait_word*) of the corresponding marker. Otherwise the command returns zero.

The wait markers are provided for synchronization purposes. The user program should perform a handling routine for each wait marker. When that handling routine is finished, processing of the list can be resumed by the control command **release_wait**.

The **set_wait** command causes the PAUSED status (queryable by **get_status**) to be set and the BUSY status to be reset. The opposite occurs after a subsequent **release_wait** command.

List execution can be interrupted at any desired point in time by the control command **pause_list** (or by the synonym command **stop_list**) and resumed by the **restart_list** command. The **pause_list** command causes the laser control signals to be disabled⁽¹⁾ and keeps the scan system in the most recently defined state – even if in the middle of microvectoring. After a subsequent **restart_list** command, the scan system resumes the planned movements (of the current command) and the laser control signals are reenabled and switched back on (the laser is started with the standard laser parameters, i.e. without any softstart or similar parameters that might have been defined prior to **pause_list**).

The **pause_list** command sets the PAUSED status (queryable by **get_status**) and the **restart_list** command resets it. The BUSY status is not affected by either of these commands.

6.4.6 Automatic List Changing

If the memory was configured for two list buffer areas "List 1" and "List 2" (see page 74), then a second list can be loaded while the first list is executing. It typically takes substantially longer to process a list than to load it into the memory. Continuous execution of arbitrarily long lists (when divided into command blocks) is therefore also possible.

Continuous command output, which requires switching between two lists, can be achieved by automatic list changing – aided by a few commands as described below.

The commands for automatic list changing only take effect upon execution of the next encountered **set_end_of_list** command – i.e. automatic list changing after processing a list can only occur if that list was terminated with a **set_end_of_list** command. Otherwise, processing would resume at the start of the same list.

If the size of the second buffer area ("List 2") was set to 0 during memory configuration, then all automatic list change commands cause a repetition of "List 1" (at the specified position, if applicable).

One-Time List Change

The commands **auto_change** and **auto_change_pos** activate an automatic, one-time list change between "List 1" and "List 2". After execution of the current list (when the **set_end_of_list** command is encountered), output of the next list is thereby automatically started. The **auto_change** command causes the next list to be started at position 0; the **auto_change_pos** command causes starting at a specified position (list buffer address as an offset to the beginning of the list).

(1) If the signals are disabled, the laser output ports (LASERON, LASER1 and LASER2) are in the (high impedance) tristate mode.



Alternating List Changes

Another way to achieve continuous data transfer is by alternatingly repeating output of the two lists. To do so, issue the command **start_loop**. This causes a continuous, automatic and alternatingly repeating output of both lists, assuming both lists were terminated with **set_end_of_list** commands. The alternating output repeats until issuance of a **quit_loop** command, which terminates continuous output as soon as the current list is finished.

The currently non-active list can be newly reloaded even as the other list executes. This allows continuous alternating output of two lists with not only fixed content, but also constantly new content.

Notes

- The commands for starting alternating list changes, and the **start_loop** command, can be issued at any point in time.
- When loading a list while another is executing, make sure no still-needed commands are thereby overwritten. Useful here is the **load_list** command, which only starts loading a list if it is non-active or has already finished execution (see "Loading Lists", page 75).
- Moreover, the currently new list should have made a certain amount of loading progress before the list change occurs. Otherwise, "old" commands might be unintentionally executed. The input pointer should always be adequately ahead of the output pointer to enable PCI transfer of buffered list commands (see page 75) and possible use of so-called short list commands (see page 65).
- The RTC5 does not support the circular queue mode. Nevertheless, this operational mode can be effectively emulated by the alternating list change commands described above and the **load_list** command.

6.5 Structured Programming

The RTC5 command set supports structured programming and output of list commands by numerous ways to define subroutines and character sets, as well as several list commands for controlling program flow. These are described below.

6.5.1 Subroutines

As list-command sequences, subroutines can theoretically be located in any part of the list memory. Preferably, they should be written to a list buffer's upper portion (see "[Protected Buffer Area](#)", page 73). List boundaries should not run through a subroutine.

Subroutines must be terminated with a **list_return** command.

Indexed as well as non-indexed subroutines can be defined (non-indexed subroutines could already be defined in the RTC5's predecessor boards).

Non-Indexed Subroutines

As with normal list-command sequences, non-indexed subroutines are loaded into a list buffer ("List 1" or "List 2") by list-loading commands (see "[Loading Lists](#)", page 75). Each subroutine must be terminated with a **list_return** command and can be invoked by calling the **list_call** command with a parameter specifying the absolute memory address.

After the subroutine (including the terminating **list_return** command) has been processed, execution continues with the command that follows the subroutine-call command.

Nested calls are also possible, to a depth of 63.

Notes

- Though non-indexed subroutines cannot be written to the protected buffer area ("List 3"), they can be subsequently assigned protection (see also "[Subsequent Protection and Conversion of Non-Indexed Subroutines](#)", page 84). Therefore, the subroutine can be subsequently referenced by the command **set_sub_pointer** (only) if its starting address is known. Prior to loading a non-indexed subroutine into the list buffer area ("List 1" or "List 2"), you should therefore always read out the start address by the command **get_input_pointer**.
- Make sure that there is no **list_return** in a normal list flow or in a body of a subroutine, for which there has not been a corresponding subroutine call. With nested subroutine calls the integrity of the nesting is destroyed. If there is no still active subroutine call, list processing is continued at the absolute position 0.
Valid as of version OUT 540: If a subroutine begins directly after a **list_call**, **list_call_abs**, **list_call_repeat**, or **list_call_abs_repeat**, then the return address is automatically set from `Pos(list_call) + 1` to `Pos(list_return) + 1`. That is, the next processed command is the one which follows after **list_return**. It is not the command which follows after **list_call** (which would process the subroutine once again having an uncorrelated **list_return**).



Indexed Subroutines

The RTC5 lets you define not only non-indexed subroutines, but also indexed subroutines. The **load_sub** command assigns a desired index to a subroutine defined by subsequent list commands, and loads the subroutine into the protected buffer area ("List 3"). Indexed subroutines must be terminated with a **list_return** command (otherwise they are not stored) and can be invoked by calling the **sub_call** command with a parameter specifying the index.

Nested calls are also possible, up to a depth of 63.

Up to 1024 subroutines can be stored. Memory management of the indexed subroutines is automatic. The RTC5 stores each indexed subroutine's index, corresponding memory address (start position) and number of list commands in an internal management table. User management of indexes for indexed subroutines is described in the section "[Index Management and Defragmentation](#)", page 85.

The memory address of an indexed subroutine can be queried by **get_sub_pointer**. This allows invocation of the indexed subroutine as if it were a non-indexed subroutine by supplying the absolute memory address.

An indexed subroutine is only stored by the **load_sub** command when the following conditions are fulfilled:

- If, prior to loading, configuration of "List 1" and "List 2" resulted in a protected buffer area ("List 3") of sufficient size. For example, if all memory is assigned to one or both lists, then no indexed subroutines can be stored. The command **get_list_space**, if called after a **load_sub** command (but before the next **load_list** or similar command), can be used for querying the amount of still-available memory in the protected buffer area.
- If the indexed subroutine was terminated with a **list_return** command.
- If **list_return** is preceded by no other command for positioning the input pointer (e.g. another **load_sub** command, **set_input_pointer**, or **set_start_list_pos**).
- If the index is within the valid range (0 ... 1023).

After the **list_return** command, the input pointer becomes invalid and any subsequent list commands are no longer stored.

Indexed subroutines are written to "List 3" by the **load_sub** command in order of entry. The smallest possible starting address is thereby automatically determined, in accordance with already registered indexed subroutines (and character sets). If an indexed subroutine is stored using an already-existing index, then the prior subroutine with that same index is not overwritten, and remains in the protected buffer area ("List 3"), though it can then no longer be accessed through its index by **sub_call** (but can still be called through its absolute memory address).

Use **get_sub_pointer** to query whether a subroutine is referenced by a particular index. If no subroutine is referenced, the return value is "-1" (i.e. $2^{32}-1$).



To load an indexed subroutine into a protected buffer area that is already fully loaded with indexed subroutines, you must first appropriately expand the protected buffer area's size with **config_list** and then defragment it with **save_disk/load_disk** (expanding the size alone is not sufficient; see [page 85](#)).

Observe the following guidelines when programming indexed subroutines:

- In an indexed subroutine, the **set_end_of_list** command is replaced with a **list_nop** command.
- Absolute jumps within or out from the protected buffer area ("List 3") are ignored during execution (see "[Jumps](#)", [page 89](#)). Therefore, absolute jumps cannot be used in indexed subroutines.
- Relative jumps that exceed the boundaries of an indexed subroutine and a jump command which initiates a jump on the command itself is ignored during execution, too.

Calling Subroutines

Subroutines are not started directly, but instead by a **list** command within a list or from a subroutine.

Nested calls up to a maximum depth of 63 are also possible.

"Normal" Calls

As previously described, non-indexed subroutines can be called by the **list_call** command and a parameter specifying the absolute memory address.

Indexed subroutines can be called either by the **sub_call** command along with the index or by the **list_call** command along with the absolute memory address.

Notes

- Index management or defragmentation (see [page 85](#)) can result in a change of the indexed subroutine's absolute memory address. Therefore, SCANLAB recommends not calling indexed subroutines by the **list_call** command.

"AbsCalls"

With the RTC5's predecessor boards, subroutines could only use *relative* vector and arc commands (see [page 103](#)) if the corresponding processes were intended to be repeated in different parts of the image field.

Thanks to "AbsCalls", the RTC5 has removed this restriction. "AbsCalls" pass on the current position in the form of an offset that is usable for subsequent execution of absolute vector and arc commands in the subroutine. Nested calls are taken into account when the offset is determined. This can be used, for instance, to define character sets by absolute vectors.

"AbsCalls" from subroutines can be made with the commands **list_call_abs** and **sub_call_abs**.

Conditional Calls

To enable calling of subroutines ("normal" calls and "AbsCalls") dependent on external control signals, additional commands are available for conditional branching during program execution (see "[Conditional Command Execution](#)", page 244).

Repeatedly Executed Calls

The commands **sub_call_repeat** and **sub_call_abs_repeat** can be used to automatically execute the body of a subroutine several times.

Subsequent Protection and Conversion of Non-Indexed Subroutines

Non-indexed subroutines can be (directly) written only to a list buffer area ("List 1" or "List 2"), but not to the protected buffer area ("List 3").

There are essentially two methods to subsequently assign protection:

(1) Change the configuration:

The region of the list buffer in which the non-indexed subroutine was written (in case this region lies in the upper range of the memory) is assigned to the protected buffer area by the **config_list** command. Subroutines subsequently protected in this manner remain non-indexed (with unaltered memory addresses) and can, as before, be called by the **list_call** command.

To enable non-indexed subroutines to be placed in this manner in the uppermost range of the memory, the corresponding memory area might need to be temporarily assigned to the list buffer by **config_list** prior to loading the non-indexed subroutine.

(2) Converting to indexed subroutines:

A non-indexed subroutine can be referenced by the **set_sub_pointer** command (the non-indexed subroutine is thereby assigned an index and is included in the management of indexed subroutines). It can subsequently be copied as an indexed subroutine to the protected buffer area ("List 3") by **save_disk/load_disk** (**save_disk** automatically stores all indexed subroutines and non-indexed subroutines subsequently-referenced by **set_sub_pointer** onto on a PC storage medium binarily, ordered by index - **load_disk** copies all those subroutines from there to the protected buffer area). A subsequent call by **sub_call** along with the index results in invocation of the copy residing in the protected buffer area rather than the original.

If you want to subsequently protect a non-indexed subroutine – either by method 1 or method 2 – then be aware that absolute jumps within and out from the protected buffer area are not allowed (see "["Jumps"](#), page 89).

Indexed subroutines subject to conversion (method 2) must observe all programming rules regarding indexed subroutines (see [page 83](#)).

Always try to use only one of the two methods in order to avoid unintended data loss in the protected buffer area by overwriting. If you begin working with method 1 but later want to also use indexed subroutines, then you should convert into indexed subroutines all of the protected buffer area's non-indexed subroutines created with method 2 before you define the first indexed subroutine by the **load_sub** command.

When doing so, observe the following guidelines:

- If method 1 is used and you remove overwrite-protection for a part of the protected buffer area, then you risk overwriting indexed subroutines or previously protected subroutines that might still be needed.
- Non-indexed subroutines subsequently protected with method 1 can under some circumstances be unintentionally overwritten by a later **load_sub** or **load_disk** command.
- The **set_sub_pointer** command links the supplied index with the supplied start address even if an indexed subroutine had already been previously defined for this index. The original indexed subroutine with this index is then no longer referenced and no longer callable by the index.

- If using method 2, you should use it fully. If the **set_sub_pointer** command alone is executed, then the subroutine is already callable by **sub_call** and its index, but the subroutine remains unprotected against overwriting. Protection is obtained only after the subroutine is subsequently copied as an indexed subroutine by **save_disk/load_disk** into the protected buffer area. Likewise, the number of list commands in the subroutine is included in the internal management table only by **save_disk/load_disk**.
- The **save_disk** command ignores unreferenced non-indexed subroutines, even those subsequently protected in the protected buffer area by method 1. Be aware that they can be overwritten in the protected buffer area by **load_disk**.
- The **save_disk/load_disk** command automatically replaces unallowed commands (e.g. **set_end_of_list**) with **list_nop** commands.
- Indexed subroutines repeatedly referenced with **copy_dst_src** are duplicated in the list buffer by a subsequent **save_disk/load_disk**. This could lead to buffer overflow in the protected buffer area.
- Before executing the **load_disk** command, be sure the protected buffer area ("List 3") is of sufficient size after configuration of "List 1" and "List 2" (**save_disk** returns the number of stored list commands). An indexed subroutine is *not* stored by **load_disk** if there is not sufficient memory.
- Conversion of a subroutine by method 2 changes the absolute memory address of the subroutine.

Deprotecting Subroutines

The protection of a subroutine stored in the protected buffer area is removed if it is assigned by **config_list** to one of the list buffers ("List 1" or "List 2").

The subroutine can then still be called using the same parameters (index or absolute memory address), but no longer has protection against unintentional overwriting.

Index Management and Defragmentation

Subroutines can be copied, renumbered or converted by the **copy_dst_src** command, which creates an additional reference (index) to an indexed subroutine (that can also be called with this new index). The command only alters the corresponding entry in the internal management table and does not modify the list buffer's memory contents.

Unneeded references (unneeded entries in the internal management table) can be deleted by the **load_sub** command directly followed by a **list_return** command. Here, too, deletion occurs only in the internal management table, while the list commands of the previously referenced subroutine continue to reside in the list buffer.

The **get_sub_pointer** command can be used to query whether a subroutine is referenced for a particular index. If no subroutine is referenced, the return value is "-1" (i.e. $2^{32}-1$).

A real copy of an indexed subroutine in the protected buffer area can be created (after the **copy_dst_src** command) with **save_disk/load_disk**. Subroutines with multiple references are thereby written several times to the list memory. Keep this in mind in order to prevent unintended buffer overflow of the protected buffer area.



Memory for new indexed subroutines can sometimes be blocked by no-longer-needed subroutines residing at lower protected-buffer-area positions than those of still-needed subroutines – this is because the **load_sub** command always places a new indexed subroutine after the referenced indexed subroutine with the highest position in the memory. For this reason, simply increasing the protected buffer area's size by **config_list** fails to produce further usable memory for storing additional indexed subroutines (the protected buffer area can only be expanded downward, not upward).

This situation can be resolved by defragmenting with **save_disk/load_disk**. All indexed subroutines and non-indexed subroutines subsequently referenced by **set_sub_pointer** are thereby rewritten, starting at the protected buffer area's lowest memory position and ordered by index. The now-available upper memory can be used for storing additional indexed subroutines.

Notes

- Before using **load_disk**, be sure the protected buffer area ("List 3") is of sufficient size after configuration of "List 1" and "List 2". Indexed subroutines without sufficient space there are *not* stored by **load_disk**. The **save_disk** command returns the number of stored list commands. No-longer-needed subroutines should have been previously dereferenced by **load_sub** directly followed by **list_return**.
- In some circumstances, index management or defragmentation can alter the absolute memory address of an indexed subroutine. It is therefore not advisable to call an indexed subroutine by the **list_call** command.
- **save_disk** stores subroutines starting from the referenced address to the first-encountered **list_return**. Relative jumps are not evaluated. So do not use branches to several **list_return** commands. Instead, reclose eventual branches in front of only one single **list_return** command.



6.5.2 Character Sets and Text Strings

For marking tasks, it is convenient to use the RTC5's list memory for storing command lists as separate subroutines that define how the scan system should mark the needed characters and text strings.

To simplify management of characters and text strings, the RTC5 provides the possibility of storing indexed character definitions and text string definitions in its protected buffer area ("List 3") and calling them by simple commands.

Indexed character definitions and text string definitions are essentially indexed subroutines, but definable and callable by their own commands, and managed by a dedicated internal RTC5 management table – separately from indexed subroutines.

The individual character and text string definitions must specify the shape and orientation (e.g. parallel to the X or Y axis) of the characters or text strings. Both relative and absolute vector commands can be used for this. The character's or text string's end position should be chosen to serve as the start position of a subsequent character. Each character definition or text string definition must be terminated with **list_return**.

Defining Indexed Character Sets

A sequence of character-defining list commands can be directly stored in the protected buffer area by the **load_char** command (the resultant automatically-assigned memory address can be queried with **get_char_pointer**). Alternatively, a non-indexed subroutine can be subsequently referenced with the **set_char_pointer** command and then copied by **save_disk/load_disk** as an indexed character in the protected buffer area.

The RTC5 manages up to 4 character sets, each with 256 indexed characters.

Other than that, the same rules as for indexed subroutines are applicable (see "["Indexed Subroutines", page 82](#)" and "["Subsequent Protection and Conversion of Non-Indexed Subroutines", page 84](#)").

Notes

- \0 (NUL) is a markable character, too. \0 also serves as a text-output delimiter (for text strings), in which case it is not marked.
- Indexed character set definitions cannot use **mark_text**, **mark_time**, **mark_date** or **mark_serial** commands; otherwise, improper marking might occur during processing of indexed characters.



Calling Indexed Characters

The marking of an individual character is started by calling the **mark_char** command (or the "AbsCall" command **mark_char_abs**) along with the index of the corresponding indexed character.

Individual serial numbers can be marked by indexed characters (numbers) by calling the **mark_serial** command (see "[Marking Dates, Times and Serial Numbers](#)" on page 170).

The marking of entire text passages can be started by the **mark_text** command (or the "AbsCall" command **mark_text_abs**). The desired character set can be selected in advance by the **select_char_set** command. When a **mark_text** command is loaded, the to-be-marked text (if more than 12 characters in length) is split into blocks of 12 characters, with each block receiving its own **mark_text** command in the list memory. Keep this in mind to prevent unintended overflow of the corresponding buffer area.

Defining Indexed Text Strings for Times, Dates and Serial Numbers

For the marking of times, dates and serial numbers, it can be useful to define text strings such as months ("January" ... "December", "Jan." ... "Dec.", "/01/" ... "/12/" etc.) and days of the week ("Sunday" ... "Saturday" or "Sun." ... "Sat." etc.).

Here, you can likewise use previously-defined character sets with the **mark_char** and **mark_text** commands.

With **load_text_table**, a sequence of list commands defining a text string can be loaded directly into the protected buffer area as an indexed text string (the resultant automatically-assigned memory address can be queried by **get_text_table_pointer**). Alternatively, a non-indexed subroutine can be subsequently referenced with the **set_text_table_pointer** command and then copied by **save_disk/load_disk** as an indexed text string in the protected buffer area

The RTC5 can manage up to 42 indexed text strings.

Other than that, the same rules as for indexed subroutines are applicable (see "[Indexed Subroutines](#)", page 82 and "[Subsequent Protection and Conversion of Non-Indexed Subroutines](#)", page 84).

Notes

- The **set_char_table** command is synonymous with the **set_text_table_pointer** command.



Calling Indexed Text Strings

Indexed text strings can be called for marking times, dates and serial numbers by the commands **mark_time**, **mark_date** and **mark_serial** (or the "AbsCall" commands **mark_time_abs**, **mark_date_abs** and **mark_serial_abs**) (see "Marking Dates, Times and Serial Numbers" on page 170).

Management of Indexed Characters and Text Strings

The RTC5's management of indexed characters together with the management of indexed text strings occurs by one internal management table maintained separately from the internal index management of indexed subroutines.

Index management by users (renumbering, copying, ...) resembles index management of indexed subroutines (see "Index Management and Defragmentation" on page 85) using the commands **copy_dst_src**, **load_char**, **load_text_table**, **get_char_pointer**, **get_text_table_pointer** and **save_disk/load_disk**. Defragmentation of the protected buffer area also includes indexed characters and text strings.

6.5.3 Jumps

The **list_jump_pos** (synonymous with **set_list_jump**) and **list_jump_rel** commands allow the definition of runtime jumps by the RTC5 to a specified address.

With the **list_jump_pos** command, an absolute memory address within the configured list area ("List 1" and "List 2") can be specified. Jumps within or out from the protected buffer area ("List 3") are not allowed with these commands. Any **list_jump_pos** command having such an unallowed jump address are ignored during execution.

With **list_jump_rel**, jump widths (i.e. relative memory addresses) can be specified. This command can be used in all memory areas, even the protected buffer area ("List 3"). Nevertheless, when specifying jump addresses, you should be sure the jump does not exceed the boundary of the corresponding memory area. Otherwise, the RTC5 ignores the command during execution.

If the command is used in an indexed subroutine, you must further ensure the jump does not exceed the subroutine's boundaries. During processing of indexed subroutines, relative jumps that exceed a subroutine's boundaries are ignored by the RTC5.



Notes

- Reconfiguration of the list memory or conversion of a subroutine can result in an originally-valid jump address becoming invalid due to new list boundaries or an altered subroutine position in the memory. In this case, the RTC5 ignores the corresponding jump command – hence, the program does probably no longer function as intended. Therefore, exercise care when programming jump commands.
- When conditional jump commands are used, execution of a jump is dependent on an external control signal (see "[Conditional Command Execution](#)", page 244).
- Jump commands initiating a jump to themselves as `list_jump_rel(0)` are ignored at runtime to prevent an infinite loop that excludes further activities.
On the other hand, conditional jump commands as `list_jump_rel_cond(Mask1, Mask0, 0)` are allowed, e.g. to wait for confirmation of a signal.

6.5.4 Circular Queue Mode

The RTC5 does not support the circular queue mode that could be activated on the RTC3/RTC4 by [`set_list_mode`](#).

But the RTC5 can effectively emulate this operational mode by an alternating list change and the command [`load_list`](#).

`load_list(3, 0)` ensures that – as in the RTC3/RTC4's circular queue mode – new commands are loaded only into an already processed list (that is not BUSY), without needing to explicitly supply the number of the list. And continuous processing of list commands is not hindered by the simultaneous loading of new commands (see also "[Alternating List Changes](#)", page 80 and "[Loading with Protection](#)", page 75).

Users who only want to assign the list memory's full capacity to a single list – as realized in the RTC3/RTC4's circular queue mode – can accomplish this very simply by [`config_list`](#).



6.5.5 Loops

Although list jumps (see [page 89](#)) and conditional jumps (see [page 244](#)) let you repeat any number of list commands limitless or under external control, precisely specifying the number of executions is not always reliably possible. But this can be achieved with the command pair **list_repeat** and **list_until**. The command sequence between these two short list commands execute exactly as often as specified with the **list_until** command's parameter, but at least once. Here, nesting up to 8 loops deep is allowed.

The commands **list_repeat** and **list_until** must always be used in pairs. Unpaired or superfluous commands (**list_until** without an associated **list_repeat**, as well as **list_repeat** commands leading to a nesting depth greater than 8) are ignored. Empty loops (e.g. **list_repeat** directly followed by **list_until**) terminate immediately and are not repeated.

You can place such command pairs not only within lists, but also within subroutines.

In subroutines, **list_until** performs a **list_jump_rel** to the address directly after the associated **list_repeat**. Loops do not function beyond the bounds of a subroutine, because list jumps into or out of subroutines are not permitted (see [page 89](#)).

Within lists, however, **list_until** executes a **list_jump_pos** (to the address directly after the associated **list_repeat**). Here, **list_repeat** and **list_until** can even reside in two different lists, provided that list changing is ensured by either an explicit list jump or an automatic list change. If, on the other hand, a list actually terminates (e.g. when using **auto_change_pos**), then the **list_repeat** stack gets automatically deleted and the started loop can no longer be ended because the next **list_until** no longer has its associated **list_repeat**.

set_end_of_list deletes the entire loop management if no automatic list change exists, but **list_return** does not.

Explicit list jumps to a **list_repeat/list_until** loop's body or from inside a loop to a loop-external location are allowed, but cannot be monitored. Careless use could therefore compromise loop management integrity so severely that started loops do not execute as expected (but subroutine calls from inside a loop are always reliably possible as long as the subroutine itself contains no unpaired **list_repeat/list_until** commands).



6.6 Using Multiple RTC5 Boards in One Computer

The RTC5 DLL supports simultaneous control of any number of RTC5 Boards in one PC.

The RTC5 Boards work completely independently of each other. Command lists for each board can be loaded and executed at any time.

There are two different methods for writing user programs using several RTC5 Boards:

- the multi-board programming and
- the sequential programming.

6.6.1 Multi-Board Programming

In this programming method, the *multi-board* versions (command names with prefix `n_`) of the RTC5 commands are used.

Each of these multi-board commands allows specifying the number of the RTC5 Board that shall receive the command.

To use a multi-board command, proceed as follows:

- ▶ Add the prefix `n_` to the command name.
- ▶ Include the number of the RTC5 Board to which the command shall be sent as the **first** parameter (unsigned 32-bit value). The remaining parameters of the command – if any – are the same as for the normal (single-board) command.

The installed RTC5 Boards are numbered in the order found during initialization (starting with 1). The multi-board command `n_get_serial_number` (see page 320) can be used to determine which RTC5 Boards have been assigned numbers (see example (3) below). The command `RTC5_count_cards` (see page 453) can be used to check how many boards are detected by the RTC5 DLL.

Notes

- If the specified number exceeds the number of RTC5 Boards found during initialization (see `rtc5_count_cards`) or if 0 is specified (real boards begin at 1), multi -board commands are sent to the active board.

All multi-board commands are listed in chapter 10.1, page 250. Nearly all single-board commands are also available in multi-board form. Exceptions are explicitly noted in the description list (in chapter 10.2).

Examples in Pascal

(1) `n_jump_abs(1, 500, 500)`

writes a jump command to the point (500, 500) into the current list of RTC5 Board #1.

(2) `n_execute_list(RTC5_no, list_no)`

executes list number `list_no` (1 or 2) on the RTC5 Board with the number specified by the variable `RTC5_no`.

(3) `sn_1 := n_get_serial_number(1)`

returns the serial number of board #1.

6.6.2 Sequential Programming

For sequential programming, the command `select_rtc` activates one of the installed RTC5 Boards for the respective user program. In this programming method, the normal (single board) commands can be used. The multi-board commands are not affected by the command `select_rtc` (presumed a valid board number is specified, see above). All commands following the `select_rtc` command are sent to the selected board until the command `select_rtc` is used again. The specified RTC5 number must not be larger than the total number of RTC5 Boards. See also [rtc5_count_cards](#).

Care must be taken if a process uses multiple boards by multiple threads, because the command `select_rtc` immediately redirects the output of *all* currently running threads of a process (not of any other process) to the specified RTC5 Board. In multi-threaded applications, this can result in programming errors. For such applications, only the multi-board commands should be used, see [chapter 6.6.1 "Multi-Board Programming", page 92](#).



Caution!

- The command `select_rtc` defines the active RTC5 Board for all threads of one user program that are currently running. In multi-threaded applications, this can result in programming errors. For such applications, only multi-board commands should be used.

6.6.3 Master/Slave Operation

If multiple synchronously-clocked RTC5 Boards are to be used in a PC, then the RTC5 Boards must first be connected pairwise with each other by the MASTER and SLAVE connectors and then installed in adjacent PCI slots. Always connect a board's MASTER connector to the SLAVE connector of another board. Suitable connection cables are available from SCANLAB.

An RTC5 Board automatically gets the master board of a master/slave chain if a further RTC5 Board is connected to its MASTER connector but no further RTC5 Board is connected to its SLAVE connector. All other RTC5 Boards automatically get slave boards.

`get_master_slave` can be used to query separately for each board the master/slave status, i.e. whether the board is operated as a master, slave or single board.

For a source code example on how to check which RTC5 Board is the master and which one is slave, see section "[Example Code](#)", page 95.

Initialization

The commands `load_program_file` and `load_correction_file` must be executed on all RTC5 Boards of a master/slave chain. The synchronous timing with stable phase position of a master/slave chain is severed by the first not-initialized board. If an RTC5 Board is initialized by `load_program_file` but connected as slave to a board, then it is subject to its own clock with an arbitrary phase position.



Clock Phase Synchronization

If the RTC5 Boards of a master/slave chain shall be synchronously clocked with a defined relative clock phase, then the boards must be correspondingly synchronized by the command **sync_slaves**.

Therefore, it is only necessary to send the command **sync_slaves** (one-time) to the master board.

SCANLAB recommends performing the synchronization immediately after all boards have been initialized (by **load_program_file** and **load_correction_file**). It is sufficient to call **load_correction_file(0,1,2)** or to temporarily detach all scan heads.

After synchronization, the clock phase of each slave board is (reproducibly) delayed by approx. 0.16 µs in relation to the clock phase of the preceding (upstream) board. Without synchronization, delays of up to 10 µs can occur. You can use **get_sync_status** to check if a slave board is synchronized to the master board (or to the preceding board in the master/slave chain).

Notes

- The master board does not pass the encoder signals to the slave board(s). They must always be individually supplied to the slave board(s). Here, you need to take into account the 0.16 µs clock phase shift.

Matching of Short-Command Counts (as of Version DLL 523, OUT 524)

The maximum allowed number of directly consecutive short list commands within a 10 µs clock period depends on the DSP version. To ensure that short list commands execute synchronously even when using multiple RTC5 Boards with differing DSP versions in a master/slave chain, the **sync_slaves** command (if sent to the master board) reduces the short list count on all faster boards in the master/slave chain to equal that of the slowest board (i.e. the board with the lowest DSP version number).

Notes

- The CPU clock frequency is not altered, only the count of short list commands.
- You can also use the **set_dsp_mode** command to make appropriate individual adjustments for each RTC5 Board.
- But note that some commands (e.g. **mark_ellipse_abs**) are only available on boards with higher-numbered DSP versions. Adjusting the short-command count does not change this fact. To ensure that the master/slave chain remains synchronized, only use commands that are available even for the board with the lowest DSP version number.

Synchronous Starts and Stops

Within a master/slave chain, external list starts (if enabled with **set_control_mode**) and external list stops are passed on from one board to all downstream slave boards. Therefore, a synchronous list start of all boards (with presettable track delays) can be induced by an external start signal, a **simulate_ext_start** or a **simulate_ext_start_ctrl** at the master board; and a synchronous list stop of all boards can be induced by an external stop signal or a **simulate_ext_stop** at the master board.

In contrast, list starts by **execute_list** or **execute_at_pointer** as well as list stops by **stop_execution** are *not* passed on. They must be separately executed even at master/slave-synchronized boards.

See also "[Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization](#)", page 239.



Example Code

The following example Delphi source code shows how to check which RTC5 Board is the master and which one is slave.

The code must be included in a user program (see page 71).

```
if init_RTC5_dll() <> 0 then halt; // Initialize the RTC5 DLL
if RTC5_Count_Cards() <> 2 then halt; // Are 2 RTC5 boards in the PC?

for CardNo := 1 to 2 do // Load firmware and 3D correction files onto both boards
begin
  n_Stop_Execution(CardNo); // Stop RTC if a task is currently still running
  if n_Load_Program_File(CardNo, nil) <> 0 then halt;
  if n_Load_Correction_File(CardNo, 'D3_nnn.ct5', 1, 3) <> 0 then halt;
end;

// Detect master board
// Is board 1 the master and board 2 a single slave?
if (n_Get_Master_Slave(1) = 2) and (n_Get_Master_Slave(2) = 1) then
begin
  Master := 1;
  Slave := 2;
end else
// Is board 2 the master and board 1 a single slave?
if (n_Get_Master_Slave(1) = 1) and (n_Get_Master_Slave(2) = 2) then
begin
  Master := 2;
  Slave := 1;
end
else
halt; // Something wrong with master-slave configuration

n_Select_Cor_Table(Master, 1, 0);
n_Select_Cor_Table(Slave, 1, 0);

n_Set_Control_Mode(Master, 1 + 8); // Master slave, activate Start Stop control
n_Set_Control_Mode(Slave, 1 + 8); // For external control, you must use the master
// board's control inputs
n_Sync_Slaves(Master); // Synchronize master and slave boards
// check synchronization status at any time
// provide an external /START or a simulated external start before checking
n_Simulate_Ext_Start_Ctrl(Master);
Result = n_Get_Sync_Status(Master); // must be 640
Result = n_Get_Sync_Status(Slave); // must be <11
```



6.7 Usage by Multiple Applications

Usage of installed RTC5 Boards by multiple applications (user programs) is coordinated by the DLL's internal board management, which is initiated by the command `init_RTC5_dll`.

By `acquire_rtc`, the `init_RTC5_dll` command automatically assigns access rights to the found boards, as long as access rights are not already assigned to another user program (any number of RTC5 Boards or applications can be used simultaneously, but no board can be simultaneously used by multiple applications). Access rights (even if temporary) to existing boards are assigned on an exclusive basis by the DLL. Multiple threads of one user program can use the same board, but can not send commands to it at the same time (the DLL automatically serializes the command calls).

Without access rights, a board can only be accessed by a user program through purely DLL-internal functions that do not require access rights – e.g. `get_error`, `get_last_error` or `select_rtc`. Most multi-board commands, too, are only executable for boards for which access rights exist.

If a user program has gained access rights for a board, then this board can be accessed by another user program only after the original user program explicitly releases its access rights by `release_rtc` or `free_RTC5_dll`. Acquisition of a released board can then be achieved by `acquire_rtc` (or `init_RTC5_dll` or `select_rtc`).

When a board is acquired by `acquire_rtc` (or `init_RTC5_dll` or `select_rtc`), a version compatibility check is performed on the RTC5 DLL and the program files `RTC5OUT.out`, `RTC5RBF.rbf` and `RTC5DAT.dat`. If no program files have been loaded, then a version check cannot be explicitly performed but the check is still regarded as successful and does thus not hinder the board's acquisition. If program files have been loaded and the version check detects an error, then access is denied (`get_last_error` return code: `RTC5_ACCESS_DENIED` | `RTC5_VERSION_MISMATCH`).

6.7.1 Board Acquisition by a User Program

Though the commands `init_RTC5_dll`, `acquire_rtc`, `free_RTC5_dll`, `release_rtc` and `select_rtc` affect the access rights or activation of the installed RTC5 Boards, they do not initiate a board reset (board resets can only be initiated by the command `load_program_file`). They therefore affect neither the list memory, nor the board's settings by previously issued control commands. Likewise not affected is the board's execution of a previously-started and currently-running list program. These commands also do not delete the related DLL data.

Applications subsequently acquiring a board by `acquire_rtc` (or `init_RTC5_dll` or `select_rtc`) therefore inherit the board's unadjusted memory contents and operational state. The user program therefore has use of the board's stored data and settings and can intervene in the flow of any program started by the previous user program.

If a user program releases a board by `release_rtc` and subsequently reacquires it – without it having been acquired in the meantime by another user program – then the RTC5 Board can be further used without changes, because in this situation all DLL configuration data remain unaltered. However, the above does not apply if the acquired board was released by `free_RTC5_dll` and reacquired with `init_RTC5_dll`.

When an RTC5 Board is acquired by another user program, some of the previous user program's key data managed only in the DLL is *not* (automatically) inherited. The acquiring user program thereby lacks information related to memory configuration, protected-area management, or the operational status.

If board acquisition is followed by a board reset by `load_program_file`, then all settings are newly defined anyway and such missing information would not be relevant.



On the other hand, if the acquiring user program intends to further use the RTC5 Board's inherited state, then it may need to explicitly query the missing DLL information, receive it from the previous user program and explicitly re-establish it so that the board's DLL remains consistent. In this regard, observe the following:

- For a correct behavior of the input pointer at the list borders, the memory configuration currently set in the DLL for the acquiring user program must be consistent to the acquired board's current memory configuration. If this is not the case, use the command `get_config_list`. This command obtains the board's current memory configuration and appropriately sets the DLL's board management for the acquiring user program.
- The management table of protected functions (indexed subroutines, character sets and text strings) is saved on the board and therefore inherited through board acquisition. All protected functions stored on the board therefore remain callable. On the other hand, information on where the next protected function should be loaded is lost. This information can only be restored by `save_disk/load_disk` (see page 85). An alternative restoration method is not possible. The intermixed loading of protected functions by differing applications should therefore be avoided.
- If, during loading a protected function, the command `release_RTC` is called before a subsequent `list_return` command is transferred, then the function is not stored on the RTC5 Board.
- The input pointer is generally not inherited (the input pointer location currently saved in the DLL for the acquiring user program is used, maybe corrected by `get_config_list`). On the other hand, output pointers of lists can be queried after an acquisition by `get_status`.
- After acquisition and until the next `load...` command, the list status (with reference to LOAD and READY) might be incorrect. But for further execution this is not important, and the status is newly set after the next `load...` command.
- Other settings such as `start_loop` or laser settings are not relevant to the DLL. Though settings used by the previous user program can not generally be queried, new settings can of course be established as desired.
- Error handling is performed separately for each board and each user program. When access rights are exchanged, this data is not included.



6.8 Error Handling

So that RTC5 errors can be caught at program runtime by suitable programming, the RTC5 performs general error handling. In addition, some commands allow for specific error handling.

General errors occur, for example, if a user program has no access rights for the board (RTC5_ACCESS_DENIED), or if the board fails to respond to a control command (RTC5_TIMEOUT), or if PCI communication problems occur during loading (RTC5_SEND_ERROR).

Examples of specific errors are: calling a command with an unallowed (uncorrectable) parameter (RTC5_PARAM_ERROR, e.g. see [get_value](#) or [write_da_x](#)), or rejected loading of a list command (RTC5_REJECTED, e.g. due to an illegal input pointer), or transmitting a control command at an improper time (RTC5_BUSY, e.g. [goto_xy](#), when a list is still being processed).

In such cases, the control commands are not executed; list commands are typically replaced with [list_nop](#) commands (e.g. for RTC5_PARAM_ERROR or RTC5_IGNORED, see [set_end_of_list](#) as an example).

For each command, the DLL sets and accumulates the bits corresponding to these errors in the (board-specific) error variables `LastError` and `AccError`.

- The error code `LastError` is automatically reset at the beginning of every command and therefore is a listing of occurred errors from the most recently executed command. `LastError` can be queried by [get_last_error](#)⁽¹⁾.
- The cumulative error code `AccError`, on the other hand, only gets reset when the DLL is initialized. But it can also be reset by the user program by the [reset_error](#) command. `AccError` is a listing of errors occurring since the last error reset ([reset_error](#)) and can be queried by [get_error](#)⁽¹⁾.

Error handling takes place separately for each board and each user program. A [reset_error](#) does not delete the error code of another user program with current access rights to the specified board. If access rights are exchanged, this data is not also exchanged (neither is any other board data exchanged).

Error handling only takes place during initialization and when loading onto the RTC5, but not during execution of a list program.

An example program of how to incorporate board-specific error variables is provided in the description of the [get_error](#) command.

Some control commands (e.g. [init_RTC5_dll](#), [load_correction_file](#) or [load_program_file](#)) additionally return a special error code that is not buffered and must therefore be immediately evaluated or discarded by the user program.

(1) The described mechanism only applies for commands that establish communication with the RTC5. Commands that do not establish communication with the RTC5 (e.g. [rtc5_count_cards](#), [set rtc4_mode](#) or [get_serial_number](#)) neither generate nor alter `LastError` or `AccError` (see also comments in the corresponding command descriptions).



6.8.1 Download Verification

Verification of RTC5 communication is vital particularly in medical applications. For this purpose, you can activate download verification separately for each board by [set_verify](#). This, of course, results in extended download times.

If download verification is activated and an inconsistency is found, then the error code

RTC5_VERIFY_ERROR is set, which can be queried by [get_last_error](#) or [get_error](#). Certain operations might immediately abort and the board would then no longer be functional (e.g. if the [load_program_file](#) command aborts).

With download verification activated, the following checks are performed (also note the comments in the [set_verify](#) command description):

Loading of List Commands

For list-command downloads, each download is read back and compared (for equality) against the sent command. Here, only transfer to the board itself is checked; automatic parameter adjustments (e.g. clipping) are not taken into account.

Loading of Control Commands

For control commands, the corresponding parameters are read back and compared for equality against the sent parameters. Automatic parameter adjustments are not taken into account.

[load_program_file](#)

For sending of the command [load_program_file](#), the following are checked:

- The binary support file RTC5DAT.dat is tested by a checksum for file correctness and PCI-transfer correctness.
- The firmware RTC5RBF.rbf is only checked by a bitwise transfer handshake. No other checking is possible.
- Each loaded section of the program file RTC5OUT.out is immediately read back for checking. If an error is detected, then the loading process aborts. This functionality requires driver version 1.0.4.0 or higher (see [set_verify](#) command description).

Loading of Correction Files

For loading by [load_correction_file](#), the integrity of the to-be-loaded correction file is checked (by the checksum) and the transfer itself checked for correctness by an immediate read back of the correction table. For this function, the correction file must contain a checksum (see [set_verify](#) command description).

Loading of Tables

For loading other tables (e.g. by [load_varpolydelay](#)), the transfer is checked for correctness by an immediate read back of the table. In addition to the [get_last_error](#) error code RTC5_VERIFY_ERROR, the load command's corresponding error return value might also get set.

6.8.2 Checking for Overruns

With [get_overrun](#), you can check whether overruns of the 10 µs clock period have occurred (see also [page 142](#)).



6.8.3 Example Code

The following example C source code shows how to catch an error during initialization. It ensures the program terminates with an error message

- if an error occurs during initialization with **init_RTC5_dll** (e.g. if no RTC5 Board was detected),
- if the desired RTC5 Board (here: the board with serial number 12345) is not detected,
- if access is denied to the desired RTC5 Board,
- if an error occurs during **load_program_file** (e.g. a version mismatch, file or system error).

The code must be included in a user program (see page 71).

```
UINT ErrorCode;

ErrorCode = init_RTC5_dll();

if ( ErrorCode )
{
    // Reading the number of RTC5 Boards detected during initialization with init_RTC5_dll
    const UINT RTC5CountCards = rtc5_count_cards();
    if ( RTC5CountCards )
    {
        // Detailed error analysis for all detected boards
        UINT AccError( 0 );
        for ( UINT i = 1; i <= RTC5CountCards; i++ )
        {
            // Errors which occurred during execution of init_RTC5_dll
            const UINT Error = n_get_last_error( i );
            if ( Error != 0 )
            {
                AccError |= Error;
                const UINT SerialNumber = n_get_serial_number( i );
                printf( "RTC5 Board number %d (serial number %d): Error %d detected\n",
                        i, SerialNumber, Error );
                n_reset_error( i, Error );
            }
        }
        if ( AccError )
        {
            free_RTC5_dll();
            return;
        }
    }
    else
    {
        printf( "Initializing the DLL: Error %d detected\n", ErrorCode );
        free_RTC5_dll();
        return;
    }
}
else
```



```
// Reading the internal board number for the desired RTC5 Board
const UINT SerialNumberOfDesiredBoard ( 12345 );
const UINT RTC5CountCards = rtc5_count_cards();
UINT InternalNumberOfDesiredBoard ( 0 );
for ( UINT i = 1; i <= RTC5CountCards; i++ )
{
    if ( n_get_serial_number( i ) == SerialNumberOfDesiredBoard )
    {
        InternalNumberOfDesiredBoard = i;
    }
}
if ( InternalNumberOfDesiredBoard == 0 )
{
    printf( "RTC5 Board with serial number %d not detected.\n", SerialNumberOfDesiredBoard );
    free_rtc5_dll();
    return;
}
// Selecting the desired RTC5 Board as the active RTC5 Board for this user program
if ( InternalNumberOfDesiredBoard != select_rtc( InternalNumberOfDesiredBoard ) )
{
    // Errors which occurred during execution of select_rtc
    ErrorCode = n_get_last_error( InternalNumberOfDesiredBoard );
    if ( ErrorCode & 256 )      // RTC5_VERSION_MISMATCH
    {
        // Here the multi-board command n_load_program_file must be used for initializing the board
        // (the single-board command load_program_file( 0 ) would deny execution).
        if ( ErrorCode = n_load_program_file( InternalNumberOfDesiredBoard, 0 ) )
        {
            printf( "n_load_program_file returned error code %d\n", ErrorCode );
        }
        else
        {
            printf( "No access to RTC5 Board with serial number %d\n", SerialNumberOfDesiredBoard );
            free_rtc5_dll();
            return;
        }
        if ( ErrorCode )
        {
            printf( "No access to RTC5 Board with serial number %d\n", SerialNumberOfDesiredBoard );
            free_rtc5_dll();
            return;
        }
        else
        {
            // if n_load_program_file was successful, select the desired board
            (void) select_rtc( InternalNumberOfDesiredBoard );
        }
    }
}
```



6.9 Miscellaneous

6.9.1 Free Variables

Eight⁽¹⁾ so-called “free” variables are available for users to freely set with data by the control command `set_free_variable` or the short list command `set_free_variable_list`.

These variable values can be outputted by the McBSP/SPI interface (see [chapter 9.1.7 “McBSP/SPI Interface”, page 237](#)) and you can query them by `get_free_variable` and `get_value` as well as log them by `set_trigger/set_trigger4` (see command descriptions).

The free variables let you, for example, transmit control commands over the McBSP/SPI interface to user hardware or document the board’s operational states (e.g. when branching during program execution).

Notes

- `set_free_variable` and `set_free_variable_list` allow you to assign any 32-bit unsigned value to a variable. However, the McBSP/SPI interface only outputs 24-bit values by `set_mcbsp_out_ptr` and 16-bit values by `set_mcbsp_out` (but `get_free_variable`, `get_value` and `set_trigger/set_trigger4` return full 32-bit values).

(1) applies to ≥DLL 539, ≥OUT 539. Previously: four.

7 Basic Functions for Scan Head and Laser Control

7.1 Marking Points, Lines and Arcs

7.1.1 Marking with Vector and Arc Commands

As explained in [chapter 6.1 "RTC5 Software Basics", page 65](#), positioning of the scan system's axes (and thus of the laser beam) under RTC5 control is achieved by calling jump, mark, arc and ellipse commands.

Each of these commands describes one vector or arc. With the RTC5, hence, only points, lines and arcs are marked⁽¹⁾.

Even numeric and alphabetic characters ultimately consist of the constituent lines, points and arcs that define them (see [chapter 7.5 "Marking Dates, Times and Serial Numbers", page 170](#)).

Vector commands (jump, mark) require as parameters the coordinates of the *end point* of the corresponding vector⁽²⁾. Each vector starts at the *current output position*, which is usually the end point of the preceding vector or arc.

Arc and ellipse commands require parameters for the X and Y coordinates of the arc center and the arc angle(s). Circular arcs start at the current output position, but elliptical arcs do so only with appropriate parameters.

The initial output position at start-up (or after a reset of the RTC5) is the center of the image field, i.e. the point (0|0). Refer to [chapter 7.3 "Scan Head Control", page 134](#) for a description of the image field coordinate system.

At runtime, each vector or arc to be traced by the scan system gets divided by the RTC5 into microsteps (microvectorization, see [section "Microsteps", page 107](#))⁽³⁾.

Jump commands principally serve to move the scan system's axes (while the laser is off) as quickly as possible to a new starting position⁽⁴⁾. In contrast, marking (i.e. mark, arc and ellipse) commands perform marking motions while the laser is switched on (see also following description).

To mark a point, the laser (i.e. the "laser active" laser control signals) must be switched on for the desired time period after a jump or mark command (see [chapter 7.1.3 "Marking Points", page 108](#)).

For line and arc marking, the RTC5 automatically switches the laser (the "laser active" laser control signals) on at the beginning of a marking command and later switches it back off (e.g. at the beginning of a subsequent jump command).

Here, users can specify delays to optimize the timing of scan head and laser control signals for their particular applications (see [chapter 7.2 "Delay Settings for Synchronizing Scan Head and Laser Control", page 111](#)).

Adjustment of laser parameters is described in [chapter 7.4 "Laser Control", page 144](#).

At the end of this chapter ([page 109](#)) a thoroughly-commented sample code for a basic marking task is listed.

(1) Here, wider line widths can be specified by `set_wobble_mode`.

(2) The coordinates must be specified as digital control values (without units). To avoid confusion with coordinates in [mm], SCANLAB uses the expression "coordinate values [in bits]".

(3) iDRIVE scan systems let you execute jump and `goto_xy` commands in either the (preconfigured and microvectorized) vector mode or (after enabling and activation) in jump mode (see [page 176](#)).

(4) Outside a list, repositioning can be achieved by `goto_xy` or `goto_xyz` (even while the laser control signals are on).



Jump Commands

A jump command (`jump_abs` or `jump_rel`⁽¹⁾) causes a (usually) fast movement of the scanner mirrors. Thereby the focus position (intended for the laser beam) "jumps" from the starting point to the end point of a vector. In general, the laser is switched off during the jump (if necessary, the "laser active" laser control signals are therefore automatically switched off at the beginning of the vector – see also [section "Scanner Delays", page 113](#)). The jump speed can be defined with the commands `set_jump_speed` and `set_jump_speed_ctrl`⁽³⁾.

If the laser system does not allow fast switching, the jump speed must be set high enough to prevent a visible marking effect on the workpiece. See also the commands `home_position` and `home_position_xyz`.

Mark Commands

The RTC5 automatically turns on the "laser active" laser control signals at the beginning of a mark command. During a mark command (`mark_abs` or `mark_rel`⁽²⁾), the laser focus moves along the specified vector with a constant *marking speed*, producing a straight mark on the workpiece.

If another mark (or arc) command follows immediately afterward, the RTC5 leaves the "laser active" laser control signals on. Therefore, a sequence of individual mark (and arc) commands creates a continuous marking (polyline). Turn-off of the "laser active" laser control signals occurs at the beginning of the jump (or generally: non-mark/non-arc) command that follows the final mark (or arc) command of a polyline (see also ["Edgelevel", page 119](#)).

The commands `set_mark_speed` and `set_mark_speed_ctrl` define the marking speed. The marking speed can be changed anywhere in a list or by the corresponding control command if no list is currently being processed.

Arc Commands

The arc commands `arc_abs` and `arc_rel` can be used for marking circular arcs⁽³⁾. These commands require parameters for the X and Y coordinates of the arc center and the arc angle. The circular arc starts at the current output position, with angles counted positively and clockwise (in contrast to mathematical convention).

At the beginning of an arc command, the RTC5 also automatically turns on the "laser active" laser control signals. During an arc command, the laser focus moves with the defined marking speed along the specified arc. The "laser active" laser control signals are turned off at the beginning of the subsequent jump (or generally: non-mark, non-arc or non-ellipse) command, provided no further arc commands (or a series of arc, mark or ellipse commands) follow.

(1) For using abs and rel commands see "AbsCalls" [page 83](#). Additionally, the RTC5 provides timed vector commands (see [page 223](#)), para vector commands (see [page 166](#)) and – if the 3D option is enabled – 3D vector commands (see [page 194](#)).

(2) For using abs and rel commands see "AbsCalls" [page 83](#). Additionally, the RTC5 provides timed vector commands (see [page 223](#)), para vector commands (see [page 166](#)) and – if the 3D option is enabled – 3D vector commands (see [page 194](#)).

(3) For using abs and rel commands see "AbsCalls" [page 83](#). Additionally, the RTC5 provides timed arc commands (see [page 223](#)).

Ellipse Commands

The RTC5 driver also provides commands for marking elliptical arcs. Here (unlike marking of vectors or circular arcs), you generally need two commands per arc: `set_ellipse` and the arc command `mark_ellipse_abs` or `mark_ellipse_rel`⁽¹⁾.

The `set_ellipse` command is used for specifying the arc's shape in the following manner (see figure 30):

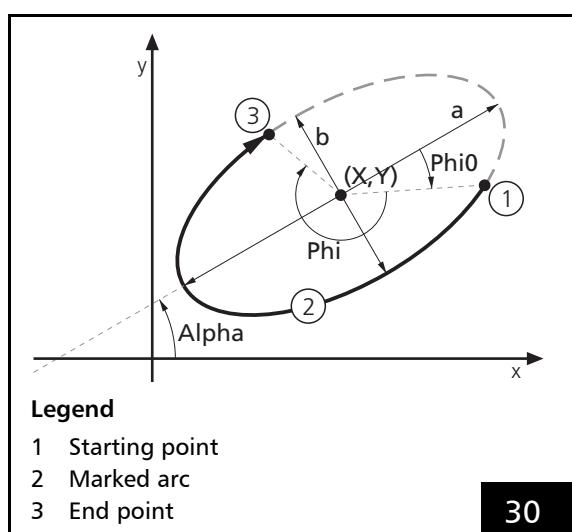
- Lengths a and b of the ellipse's half-axes
- The beginning phase angle $\text{Phi}0$ (and thereby the arc starting point's position relative to the end point of half-axis a)
- The arc angle Phi (and thereby the length of the to-be-marked ellipse section)

The arc commands `mark_ellipse_abs` or `mark_ellipse_rel` specify the to-be-executed arc's position and orientation in the following manner (siehe figure 30):

- The coordinates (X, Y) of the ellipse's midpoint
- The angle Alpha between the ellipse's half-axis a and the X axis.

Notes

- By a , you can specify either the short or long half-axis (then use b for the other axis). $\text{Phi}0$, Phi and Alpha are always relative to axis a .
- $\text{Phi}0$ and Phi are counted positively clockwise (in contrast to mathematical convention). In contrast, Alpha is counterclockwise (in accordance with mathematical convention).
- As with mark and arc commands, during an ellipse command, the laser focus moves with the defined marking speed along the specified arc. The "laser active" laser control signals are automatically turned on at the beginning of an ellipse command and turned off at the beginning of the subsequent jump (or generally: non-mark, non-arc or non-ellipse) command, provided no further ellipse commands (or a series of arc, mark or ellipse commands) follow. Because `set_ellipse` is a short list command, it can be called between a mark command and a subsequent ellipse arc command without thereby interrupting the polyline (the laser remains on) (see also page 250).
- In contrast to mark and arc commands (which automatically begin marking at the current output position), elliptical arc commands always begin marking at the starting point determined by the above-mentioned parameters. If the arc starting point does not equal the current position, then a hard jump to the starting point is executed at the beginning of marking (and jump delays are ignored).



Marking ellipse-shaped arcs

(1) For using abs and rel commands see "AbsCalls" page 83.



- Elliptical arcs can also be marked by circular arc commands (e.g. `arc_abs`) if an appropriate coordinate transformation (e.g. scaling that differs in the x/y directions) was specified with `set_matrix`. Here, though, the speed varies along the arc (see also note on [page 186](#)).

This contrasts with `mark_ellipse_abs` and `mark_ellipse_rel`, where in 10 µs intervals the step length gets adjusted for the ellipse's shape at the current position such that the arc is marked with a (largely) constant speed (the currently set marking speed).

For very large eccentricities and also at high marking speeds, however, such stepwise ellipse approximation by a 10 µs clock can produce numerical inaccuracies in the end point regions of the large half-axis. Consequently, the speed there might not be precisely constant (e.g. an eccentricity of $a/b = 2$ and 100 microvectors per circumference would produce a speed deviation of approx. 3.7%).

Moreover, as closed equations do not exist for calculating an ellipse arc length, the step length of the finally-marked microvector is generally shorter and the marking speed correspondingly lower than specified. Nevertheless, the end position is always exact.

Likewise, the Sky writing option might produce run-in/run-out irregularities at the large half-axis. Users should therefore check if their chosen parameters are compatible with their precision requirements.

Para Commands

If the vector-defined laser control is activated, also the para-mark and para-jump commands (`para_jump_abs`, `para_jump_rel`, `para_mark_abs`, `para_mark_rel`) can be used.

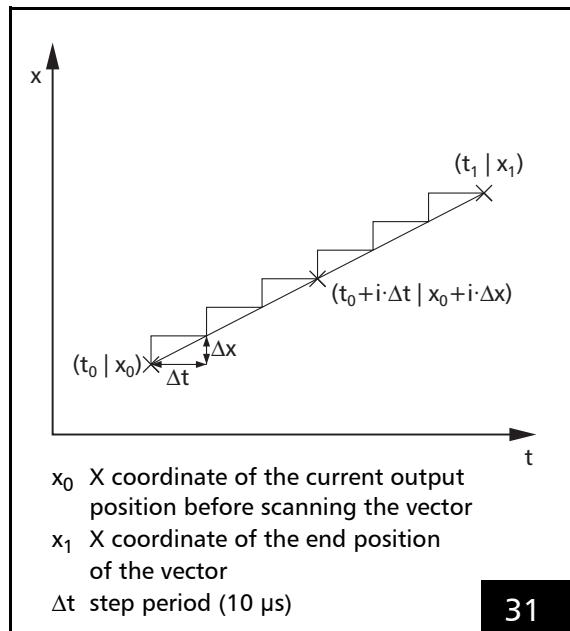
These commands simultaneously vary a signal parameter linearly along the mark or jump vector (see [page 166](#))⁽¹⁾.

(1) Additionally, the RTC5 provides timed para vector commands (see [page 166](#) and [page 223](#)).

7.1.2 Microsteps

Each vector defined by a jump, mark or arc command is divided into a number of small steps by the RTC5. These microsteps are transferred to the scan head at a constant time rate (*output period* Δt). In controlling its galvanometer scanners, the scan head implements the steps by an analog servo loop.

Figure 31 shows how the X component of a vector is divided into microsteps. The Y component is split up in the same way.



The X component of a vector is split up into microsteps.

The length Δs of each microstep is

$$\Delta s = v \times \Delta t,$$

where v is the current jump speed (marking speed).

The output period Δt of the position update is usually fixed at 10 µs. It is the same for 2D and for 3D applications. The output period cannot be set by users.

Notes

- You can implement direct execution of vectors (without microvectorization) with the help of microvector commands (see [page 221](#)).
- **iDRIVE** scan systems let you execute jump and **goto_xy** commands in either the (preconfigured and microvectorized) vector mode or (after enabling and activation) in jump mode (see [page 176](#)).



7.1.3 Marking Points

To mark a point outside of a polyline, you must switch on the laser (i.e. the “laser active” laser control signals) for the desired time period after a jump or mark command (see [laser_on_list](#), [laser_on_pulses_list](#), [para_laser_on_pulses_list](#) and [chapter 7.4 “Laser Control”, page 144](#)).

Outside or at the start of a polyline, you can also mark a point (as of RTC5OUT.out version 527) by following a jump command with a mark (or arc) command of zero length (see [page 115](#)).

Within a polyline (as of RTC5OUT.out version 526), points can also be marked by incorporating into the polyline a timed vector or arc command with a zero vector or arc length (see [page 223](#)).



7.1.4 Example Code

The following example C source code shows the commands of a simple laser scan application:

A point, a square and a circle are marked in CO₂ mode. Here it is assumed that the RTC4 compatibility mode is activated.

The code must be included in a user program (see [page 71](#)).

```
// Scan system initialization:  
// Loading and assigning a correction file  
ErrorCode = load_correction_file(0, // initialize like "D2_1to1.ct5",  
                                1, // table (#1 is used by default)  
                                2 ); // use 2D only  
if ( ErrorCode )  
{  
    printf( "Correction file loading error: %d\n", ErrorCode );  
    free_rtc5_dll();  
    return;  
}  
select_cor_table( 1, 0 ); // assigning table #1 to the primary scan head connector (default)  
  
// Laser control initialization (see page 144):  
// Setting the CO2 laser mode  
set_laser_mode( 0 );  
  
// Setting and enabling the "laser active" laser control signals  
set_laser_control( 0x18 ); // All laser signals LOW active (Bit #3 and #4)  
// This command must be called at least once to activate laser signals. Later on enable_laser/disable_laser would be  
// sufficient.  
  
// The following are list commands  
// Opens List 1  
set_start_list( 1 );  
  
// Setting the standby pulses  
set_standby( 800, 8 );  
// In RTC4 compatibility mode the standby parameters are specified in units of 1/8 µs as with the RTC4 and the  
// RTC5 multiplies the specified values by 8 to convert in integer-multiple of 1/64 µs.  
// Half of the standby output period = 100 µs  
// Pulse length of the standby pulses = 1 µs  
  
// Timing, delay and speed preset  
// Setting the scanner delays (see page 113):  
set_scanner_delays( 25, 10, 5 );  
// Jump delay = 250 µs (specified in [10 µs])  
// Mark delay = 100 µs (specified in [10 µs])  
// Polygon delay = 50 µs (specified in [10 µs])  
  
// Setting the jump and marking speed:  
set_jump_speed( 1000.0 );  
set_mark_speed( 250.0 );  
// In RTC4 compatibility mode the speed values are specified as with the RTC4 and the RTC5 multiplies the specified  
// values by 16.  
// Jump speed = 1000.0 bits/ms  
// Marking speed = 250.0 bits/ms
```



```

// Setting the laser timing (see page 144):
set_laser_pulses( 800, 400 );
    // In RTC4 compatibility mode the timing parameters are specified in units of 1/8 µs as with the RTC4 and the RTC5
    // multiplies the specified values by 8 to convert in integer-multiple of 1/64 µs.
    // Laser HalfPeriod = 100 µs
    // Laser PulseLength = 50 µs

// Setting the laser delays (see page 111):
set_laser_delays( 100, 100 );
    // In RTC4 compatibility mode the laser delays are specified in units of 1 µs as with the RTC4 and the RTC5
    // multiplies the specified values by 2 to convert in integer-multiple of 0.5 µs.
    // LaserOn delay = 100 µs
    // LaserOff delay = 100 µs

// Defining the end of the list and the end of command transfer to the RTC5 Board
set_end_of_list();

// Execute the list commands for initialization
execute_list( 1 );

// Marking procedure
// Waiting for list 1 to be not busy (load_list( 1, 0 ) returns 1 if successful, otherwise 0); if list 1 is not (no longer) busy:
// opening the list buffer for writing of list commands and setting the input pointer to the start of list 1
while ( !load_list( 1, 0 ) );

// In the following the list commands for marking point, square and circle are defined and transferred to the RTC5 Board.

// Marking the center point of the image field:
jump_abs( 0, 0 ); // Jump to center point
    // A jump delay is automatically inserted after the jump.
laser_on_list( 5 ); // Turning on the laser control signals for 50 µs

// Marking a square around the center point:
jump_abs( -20000, -20000 ); // Jump to the bottom left corner of the square
    // A jump delay is automatically inserted after the jump.
mark_abs( -20000, 20000 ); // Marking the left edge of the square
mark_abs( 20000, 20000 ); // Marking the top edge of the square
mark_abs( 20000, -20000 ); // Marking the right edge of the square
mark_abs( -20000, -20000 ); // Marking the bottom edge of the square
    // The laser control signals are automatically switched on with the first mark command after a LaserOn delay
    // and remain on for all four mark commands. A polygon delay is automatically inserted after the first three
    // mark commands, each. Initiated by the following non-marking command (jump command, see below), a mark delay
    // is automatically inserted after the last mark command and the laser control signals are automatically switched off
    // after the last mark command and a LaserOff delay (afterwards standby pulses are outputted).

// Marking a circle around the center point:
jump_abs( 0, -10000 ); // Jump to the bottom edge of the circle
    // A jump delay is automatically inserted after the jump.
arc_abs( 0, 0, 360.0 ); // Marking the circle
    // The laser control signals are automatically switched on with the arc command after a LaserOn delay. Initiated by
    // the following non-marking command (set_end_of_list command, see below), a mark delay is automatically inserted
    // after the arc command and the laser control signals are automatically switched off after the arc command and a
    // LaserOff delay.

// Defining the end of the list and the end of command transfer to the RTC5 Board
set_end_of_list();

// Starting the transferred list (and thereby the marking process)
execute_list( 1 );

```

7.2 Delay Settings for Synchronizing Scan Head and Laser Control

The timing of the scan head and laser control signals must be compatible with the dynamic behavior of the system components, i.e. the response of the scanners and the laser, and the specific interaction between the workpiece and laser radiation.

To accomplish this, users can set the following delays⁽¹⁾:

- Laser delays:
 - LaserOn delay
 - LaserOff delay
- Scanner delays:
 - Jump delay (optional variable)
 - Mark delay
 - Polygon delay (optional variable).

All delays are described in detail in this chapter.

7.2.1 Laser Delays

There are two different laser delays:
LaserOn delay and LaserOff delay.

The laser delays affect when the “laser active” laser control signals (i.e. the laser) are turned on or off before or after a mark or arc command or a series of mark and arc commands (if applicable, the RTC5 thereby switches from “laser standby” to “laser active” signals or vice versa). Laser delays do not affect the total marking time, except when they are negative.

The LaserOn delay and the LaserOff delay are set by the list command `set_laser_delays`. The time resolution for the laser delays is 0.5 µs.

The delays must be appropriate for the defined jump speed and marking speed (see also “[Notes on Optimizing the Delays](#)”, page 121).

LaserOn Delay

The LaserOn delay defines the moment when the RTC5 turns on the laser. LaserOn delay is automatically inserted at the start of a mark or arc command (first microstep). Thus, the laser is switched on only after execution of the first few microsteps. This delay can be used for several purposes:

- Many applications require laser marking without variations of intensity, especially without burn-in effects at the start or end of a vector. To achieve homogenous marking results, it is essential to scan the vectors with a constant velocity. At the beginning of a mark or arc vector, however, the mirrors first have to be accelerated up to the defined marking speed. [Figure 34](#) shows that the laser focus initially moves only very slowly. A burn-in effect may occur. To avoid this, the LaserOn delay must be set to a suitable, *positive* value. Thus the mirrors have already reached a certain angular velocity when the laser eventually turns on. However, if the LaserOn delay is too long, the first part of the vector is cut off.
- Some materials take some time until they react to the exposure to laser radiation. In this case, it can be useful to “preheat” the starting point of a mark or arc vector before marking. This can be done by setting the LaserOn delay to a *negative* value. A negative LaserOn delay extends the total marking time, because it is inserted *before* the actual mark or arc command. Additionally, the scanner delay is automatically extended if a preceding LaserOff delay has not yet finished (see “[Automatic Delay Adjustments](#)”, page 121).

(1) Furthermore, for high-precision optimization needs, an automatic readjustment of “laser active” laser control signals – and thus of the laser power – even during execution of vector and arc commands can be realized by the automatic laser control commands (see [page 159](#)).



LaserOff Delay

- Due to the acceleration phase at the beginning of the movement, a difference (a lag called tracking error) occurs between the set position and the real position of each mirror – see [figure 34](#).

After execution of a mark or arc command, the laser should not be turned off immediately because the scanners have not yet reached the final set position. Therefore a LaserOff delay is inserted automatically before the laser is turned off (see also notes [page 115](#)). For short marking vectors, if a preceding LaserOn delay has not yet finished then the LaserOff delay is automatically temporarily extended by a corresponding amount (see "[Automatic Delay Adjustments](#)", [page 121](#)).

7.2.2 Scanner Delays

There are three different types of scanner delays: *jump delay*, *mark delay* and *polygon delay*.

After each vector or arc command, the RTC5 inserts one of these delays before the next command is started.

The command `set_scanner_delays` defines the scanner delays. The time resolution for the scanner delays is 10 µs.

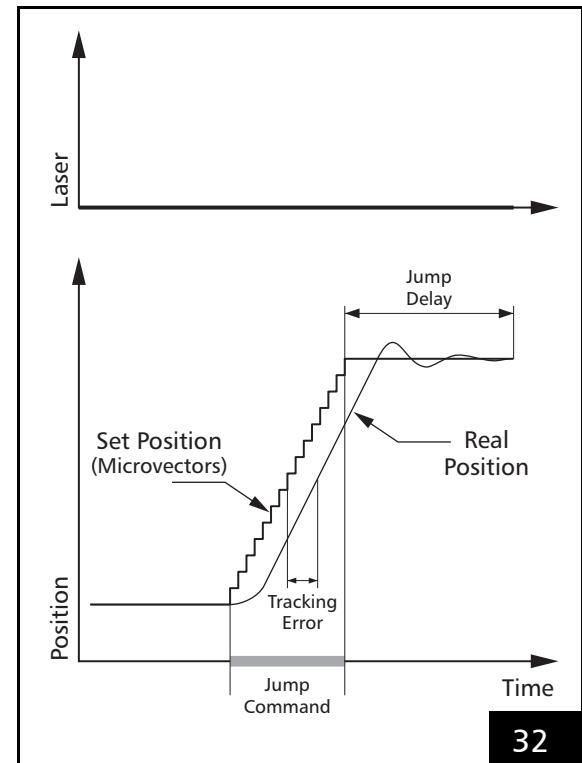
Jump Delay

When executing a jump command, the mirrors first have to be accelerated up to the defined jump speed. Because of the inertia of the mirrors, a lag called *tracking error* occurs between the set position and the real position – see [figure 32](#).

At the end of the jump, a certain settling time is necessary for the mirrors to reach the set position within some accuracy. To allow for the settling time and for the lag, the RTC5 inserts a jump delay after each jump command (but not after `goto_xy` or `goto_xyz`), before the next command is executed.

Note that the necessary settling time depends on the selected jump speed. A higher jump speed usually requires a longer jump delay. The jump delay value can be defined by users with the list command `set_scanner_delays`.

The total time needed for the entire jump command is the sum of the actual jump time and the jump delay. It can be minimized by optimizing the jump speed and the jump delay.



32

Scan head control timing during a jump command with a jump delay. The laser is not turned on.

Variable Jump Delays

During a jump vector, the laser focus (output position) usually moves with a constant linear velocity, the jump speed. Since the jump speed is the same for each jump vector, a constant jump delay is required for settling of the mirrors.

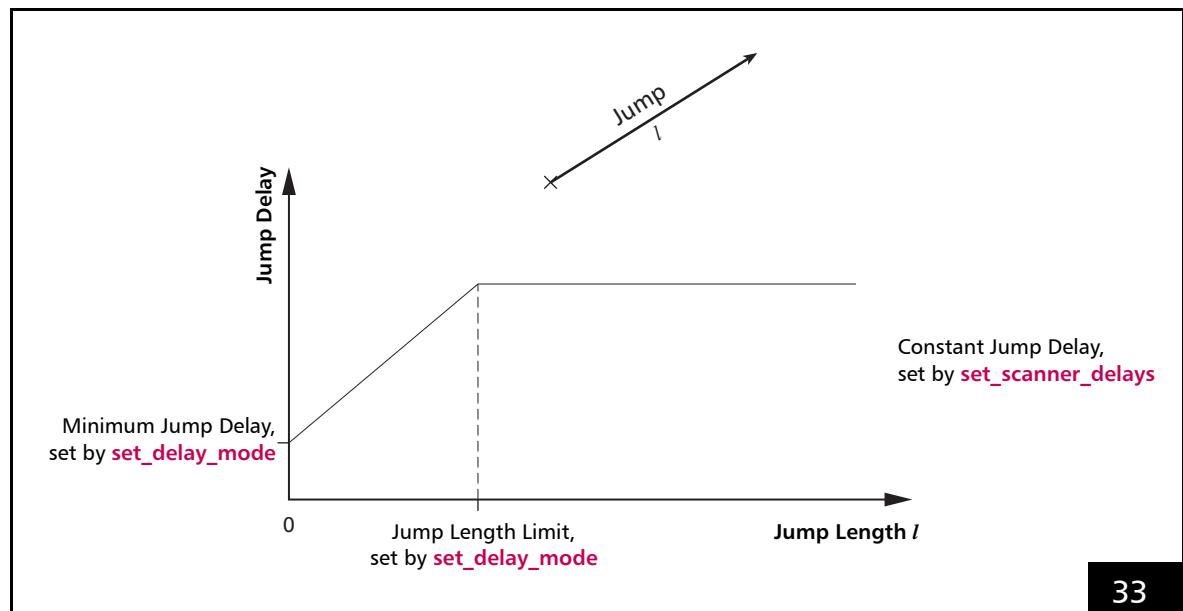
However, if a jump vector is very short, the scanners might not reach the full jump speed during the jump because of the inertia of the mirrors. In this case, a shorter jump delay might be sufficient for settling.

To make use of this, the RTC5 offers a variable jump delay mode. In this mode, the jump delay for short jump vectors are reduced in time, as shown in [figure 33](#). The minimum jump delay (for a jump vector of zero length) and the jump length limit are set by users with the command `set_delay_mode` or `set_delay_mode_list`.

When using the variable jump delay mode, total marking time can be reduced, especially in applications involving many short jumps.

Notes

- To turn off the variable jump delay mode, simply set the parameter `JumpLengthLimit` to zero.
- For jump vectors with a length of zero, the (variable) jump delay is not executed.



Variable Jump Delay

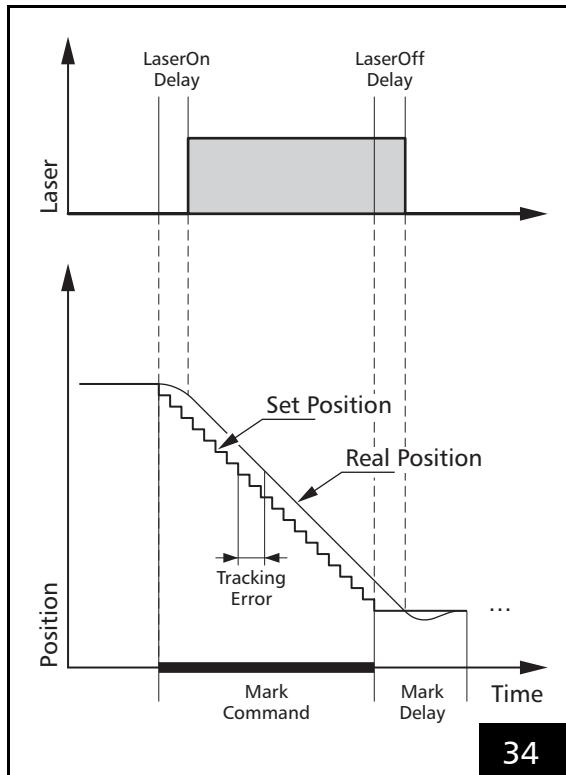
Top: length l of the jump vector

Bottom: Variation of the jump delay

Mark Delay

Although the marking speed is usually lower than the jump speed, a lag between the set position and the real position occurs not only during a jump, but also during a mark or arc command. Also a small settling time can be necessary for the mirrors to reach the set position.

To make sure that the scanners reach the final set position properly before the next command starts, the RTC5 inserts a mark delay after a single mark or arc command or after the last mark or arc command of a polyline (see figure 34).



Notes

- If a mark or arc command is *not* followed by further mark or arc commands, then a mark delay is inserted and the laser is switched off (after a LaserOff delay) (see figure 34).
- If a mark or arc command is directly followed by a further mark or arc command, then a polygon delay or a variable polygon delay is inserted instead of a mark delay (see figure 35 and figure 37). The laser then remains on unless a correspondingly smaller `edgelevel` was set with `set_delay_mode` or `set_delay_mode_list` (see page 118).
- Mark or arc commands with a length of zero that execute with *switched-on* "laser active" laser control signals (i.e. within a polyline):
 - Are short list commands (if not timed for an execution duration > 5 µs, see below). They do not interrupt a polyline even when several such commands directly follow each other.
 - Are ignored in calculating delays (as of RTC5OUT.out version 515): The mark delay or (variable) polygon delay executes in accordance with the commands that directly precede or follow the zero-length command(s). If the command is executed individually (i.e. not within a polyline), then no mark delay is performed.
 - Do not change the laser control signals (as of RTC5OUT.out version 518): if the laser is still off, then it is not switched on; likewise, if the laser is already on, then it remains on.
- But if a mark or arc command of zero length executes with *switched-off* "laser active" laser control signals (i.e. as a single command or at the start of a polyline), then it behaves (as of RTC5OUT.out Version 527) as a timed mark or arc command with an execution time of 10 µs (see following note).

Scan head and laser control timing during a mark or arc command with a mark delay. Grey shaded areas indicate that the laser is on.



- Timed mark or arc commands with zero spacial length (as of RTC5OUT.out version 526 and if the specified execution duration is > 5 μ s) behave like mark or arc commands of finite spacial length:
 - If necessary, the laser control signals are switched on (at the beginning of a polyline) and off (at the end of a polyline).
 - Mark and polygon delays execute (but variable polygon delays are always 0).
 - These commands require at least a 10 μ s clock cycle for execution.

Notes on earlier versions

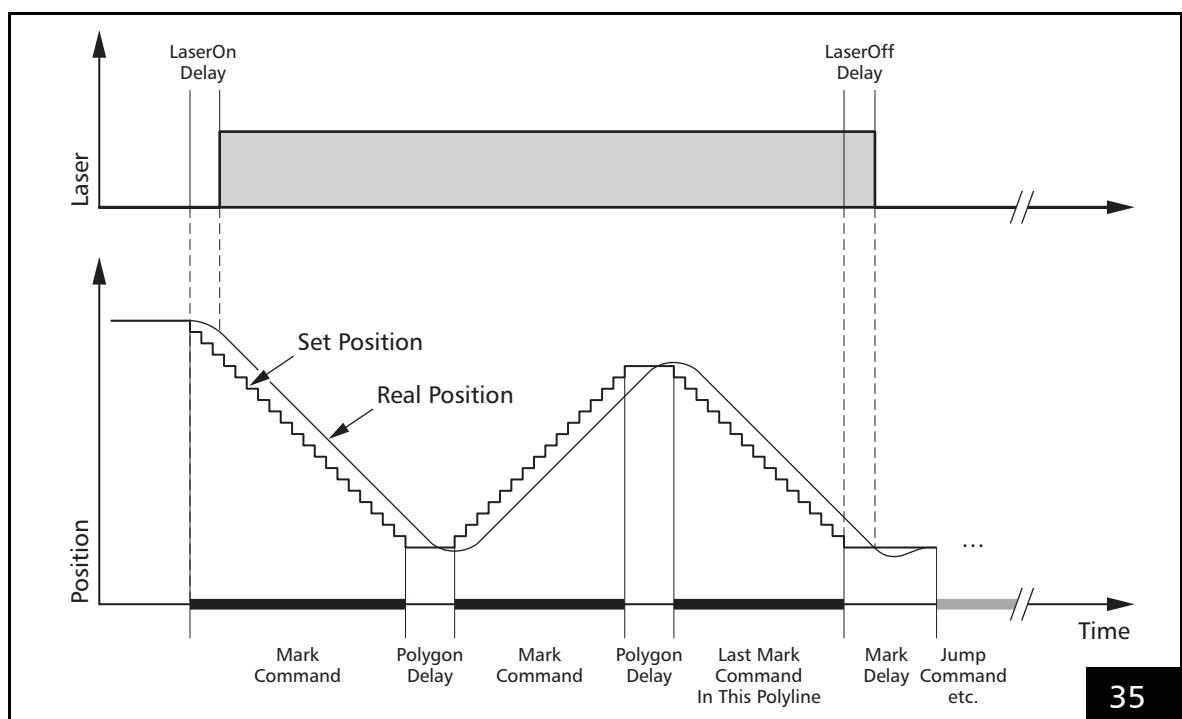
- With RTC5OUT.out version up to 514, only the subsequent mark delay or (variable) polygon delay of a zero-length mark or arc command is suppressed for a zero-length mark or arc command; but the preceding one is executed (a preceding (variable) polygon delay is executed as a mark delay).
- With RTC5OUT.out version up to 517, the laser is switched on after a LaserOn delay for zero-length mark or arc commands (as is usual for mark or arc commands with a length greater than 0).
- For RTC5OUT.out version up to 525, zero-length timed mark or arc commands behave like non-timed mark or arc commands with a length of zero.
- For RTC5OUT.out versions up to 526, the behavior of mark or arc commands with a length of zero is independent of the laser control signals.

Polygon Delay

Between two successive mark or arc commands, there is no need for a complete stop of the scanners.

Therefore, the mark delay between two successive mark or arc commands is replaced by a polygon delay, see [figure 35](#). Here, the laser remains on unless a correspondingly smaller edgelevel was set by `set_delay_mode` or `set_delay_mode_list`, see section "[Variable Polygon Delay](#)" on page 118.

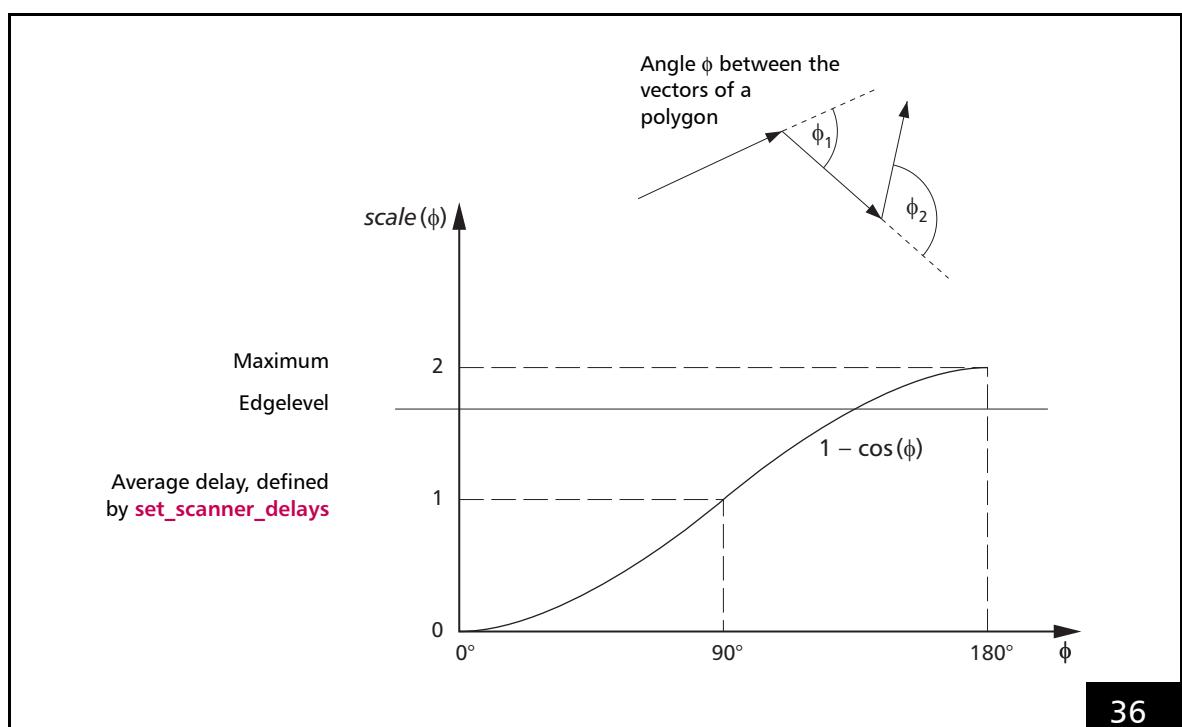
The mark delay and the polygon delay can be set independently. In addition, the RTC5 is able to vary the length of the polygon delay, depending on the angle between two marking vectors or the tangents of the arcs, see section "[Variable Polygon Delay](#)" on page 118.



Scan head and laser control timing during a polyline with a constant polygon delay

Variable Polygon Delay

A variable polygon delay mode can be activated by the command `set_delay_mode` or `set_delay_mode_list`. In this mode, the RTC5 allows varying the length of the polygon delay, depending on the angle ϕ between the two successive marking vectors – see [figure 36](#).



36

Variable Polygon Delay

Top: Definition of the angle ϕ

Bottom: Variation of the polygon delay (default curve)

For each corner of the polyline, the RTC5 calculates the variable polygon delay $v_delay(\phi)$ as follows:

$$v_delay(\phi) = scale(\phi) \times polygon_delay,$$

where $scale(\phi)$ is a scaling function ($0 \leq scale(\phi) \leq 2$).

The parameter $polygon_delay$ is set by the command `set_scanner_delays`.

[Figure 36](#) (bottom) shows the default scaling function.

This standard curve can be replaced by a customized curve. See “Customizing the Variable Polygon Delay” on page 120.

Edgelevel

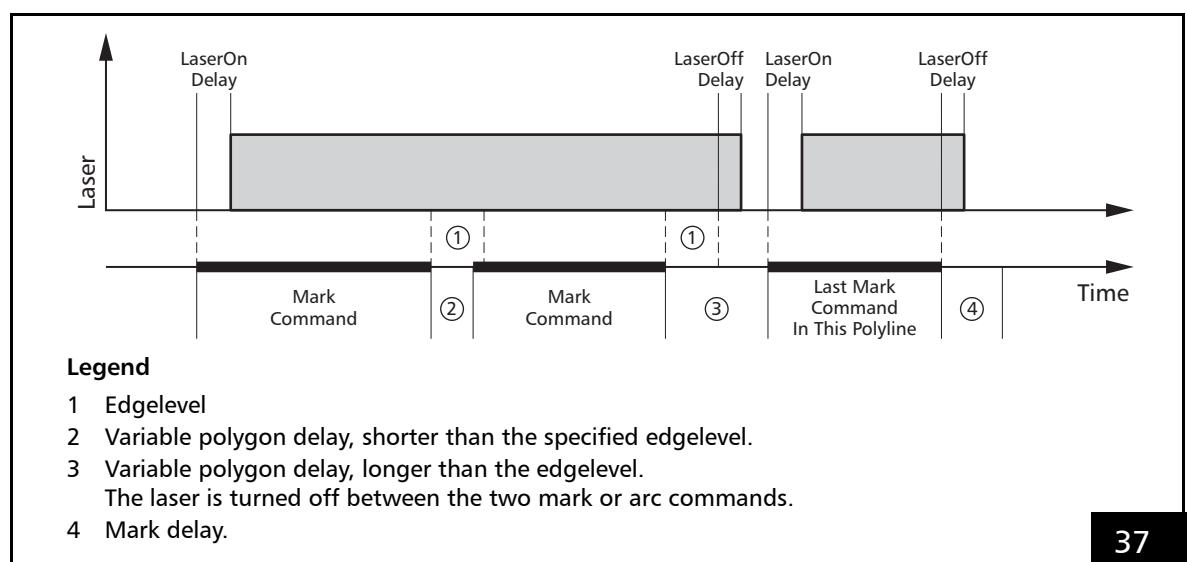
The variable polygon delay becomes quite long if the angle ϕ is close to 180° (as [figure 36](#) shows).

This might lead to burn-in effects in the sharp corners of the polyline.

To avoid this, users can define a so-called edgelevel:

If the polygon delay between two mark or arc commands is longer than or equal to this value, the RTC5 turns the laser off after the first mark or arc command upon reaching the edgelevel (after inserting a LaserOff delay) and starts a new polyline at the beginning of the next mark or arc command. See also [figure 37](#). Before the laser is switched on, the polygon delay is also (if required) extended until the LaserOff delay has completely finished.

For further details see [set_delay_mode](#).



37

Laser control timing during a polyline with variable polygon delays.
 An edgelevel was defined with the command [set_delay_mode](#).

Customizing the Variable Polygon Delay

The command **load_varpolydelay** loads a table for the scaling function $scale(\phi)$ from an ASCII text file. The text file can contain one or more tables.

Each table can contain up to 50 data points ($\phi | scale(\phi)$) for various angles ϕ . The RTC5 determines the scaling function $scale(\phi)$ from this data by linear interpolation.

Figure 38 shows a sample table and the corresponding scaling function.

The following rules apply for these tables:

- Each table must start with the instruction (Caption)


```
[VarPolyTable<No>]
```

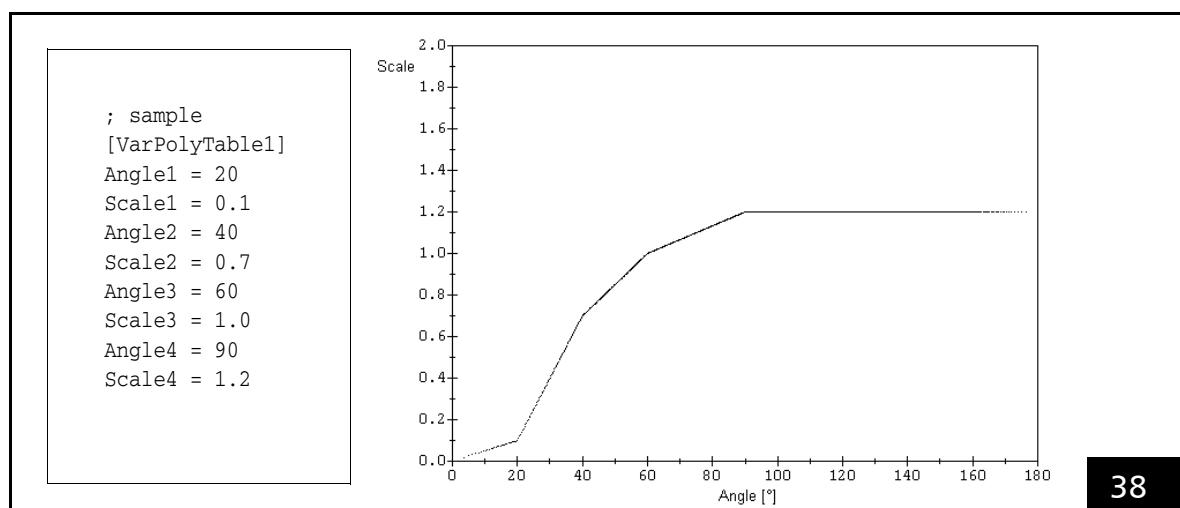
 where <No> must be replaced by a nonnegative integer which denotes the table number.
- If the table contains multiple [VarPolyTable<No>] entries with the same <No>, then only the instructions after the first entry is used; instructions that follow further entries are ignored. Only instructions up to the next '[' character (that is not preceded by a semicolon) are used.
- Each data point ($\phi | scale(\phi)$) is described by two instructions:


```
Angle<n> = <Value>
Scale<n> = <Value>
```

where <n> must be replaced by an integer ($1 \leq <n> \leq 50$) which denotes the number of the data point. The values <Value> for the angle ϕ (in

degrees) and for the scaling factor can be specified as (unsigned) floating point numbers. Use the period (.) as the decimal separator.

- If the table contains multiple data points with the same Index <n>, then the most recently read one is used and the previous ones are ignored.
- If the table contains multiple data points with the same angle ϕ , then the data point with the largest Index <n> is used and the others ignored. Equality is checked to within $\pm 0.01^\circ$.
- For <Value>, the following ranges apply:
 - $0.0^\circ \leq \phi \leq 180.0^\circ$ and $0.0 \leq scale(\phi) \leq 2.0$.
- Each instruction must be in a separate line.
- Spaces and tabs in a line (e.g. between '=' and <Value>) are ignored.
- Empty lines are ignored.
- Data points with invalid values are ignored.
- The data point of a particular index <n> is ignored if the corresponding Angle<n> and/or Scale<n> definition is missing.
- The semicolon ';' can be used for comments. All characters in a line following a semicolon are ignored.
- The instructions for data points in the table can be ordered as desired.



Sample table and resulting scaling function $scale(\phi)$. The sample table contains four data points.

- Indices for data point pairs in the table can be selected as desired within the range [1...50] (the table is then automatically sorted by ascending angles).
- If the table contains no valid data point, then the command `load_varpolydelay` has no effect (return value 1 or 13).
- The angle $\phi = 0^\circ$ means that two successive vectors are parallel and are marked in the same direction.
If the table contains no explicit data for $\phi = 0^\circ$ (equality is checked to within $\pm 0.01^\circ$), then a data point for $\phi = 0^\circ$ with the scaling factor `scale(0°) = 0` is added.
- The angle $\phi = 180^\circ$ means that two successive vectors are marked in the opposite direction.
If the table contains no explicit data for $\phi = 180^\circ$ (equality is checked to within $\pm 0.01^\circ$), then a data point for $\phi = 180^\circ$ with the largest scaling factor found in the table for `scale(180°)` is added.

After initialization of the RTC5 (with `load_program_file`), the internal (default) table for the variable polygon delay ($1 - \cos(\phi)$, see [figure 36](#)) is used. Alternatively, this can also be achieved with Name = 0 in `load_varpolydelay`.

7.2.3 Notes on Optimizing the Delays

The delays have to be set with the commands `set_scanner_delays` and `set_laser_delays`. The delays have to be appropriate for the defined jump speed and the marking speed. If the delays are not optimized, the quality of the scanning results are under some circumstances reduced and scanning time is extended. The figures on [page 124](#) through [page 126](#) show the various effects of non-optimized delays on the lettering "RTC".

Notes

- Note that the laser delays must be specified in units of 0.5 μ s, whereas the scanner delays (jump delay, mark delay and polygon delay) must be specified in units of 10 μ s.

Recommended Sequence

The LaserOn delay and the LaserOff delay should be optimized first, followed by the delays for scanner control, i.e. the jump delay, the mark delay and the polygon delay. When optimizing the laser delays, it is useful to set the jump delay and the mark delay to long values.

The lengths of the LaserOn delay and the LaserOff delay have no influence on the total scanning time if positive values are chosen (see also the following section).

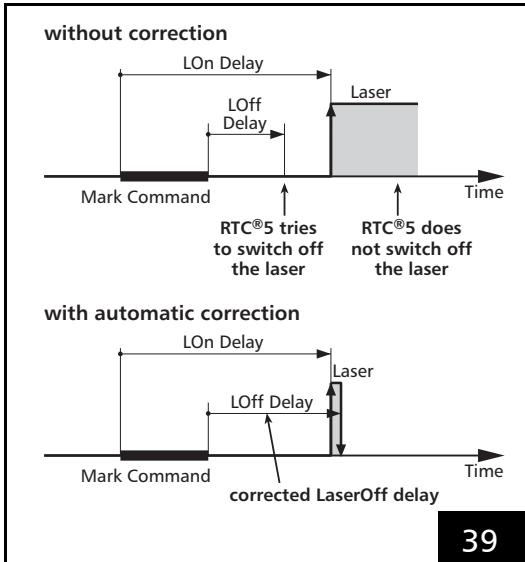
Automatic Delay Adjustments

All delays can be set for any value within the corresponding allowed range. With predecessor boards, however, some delays would result in laser control errors (when laser-on and laser-off overlap).

Therefore, the RTC5 checks each vector's defined delays and – if necessary – performs an appropriate adjustment of the delays for the respective vector:

(1) Adjustment of the LaserOff Delay

With predecessor boards, if the specified LaserOff delay is shorter than the LaserOn delay ($L_{OffD} < L_{OnD}$), then very short mark vectors could cause the LaserOff delay to finish before the LaserOn delay has finished. These boards would then attempt to switch the laser off before it had actually been switched on, and then switch the laser on without later switching it back off (see [figure 39](#)).



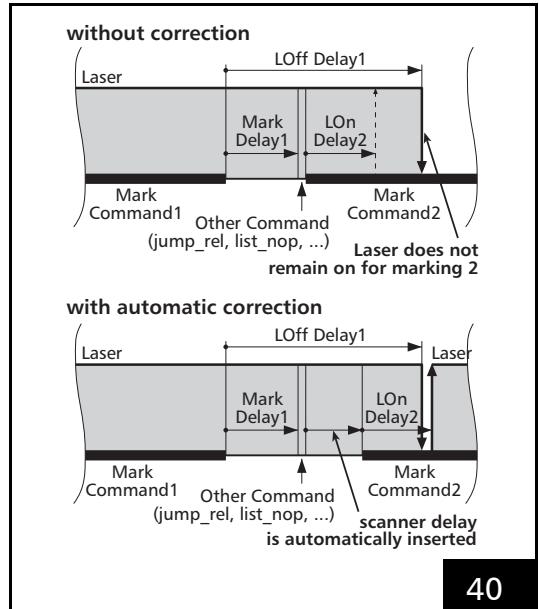
39

Automatic adjustment of LaserOff delay

One way of avoiding this problem is to generally select a LaserOff delay that is *longer* than the LaserOn delay ($\text{LOffD} > \text{LOnD}$, as with the predecessor boards). On the other hand, the RTC5 now checks each mark vector and – if necessary – automatically extends the LaserOff delay for the respective mark vector so that LaserOff occurs only after LaserOn. Laser delays can thus be set independently of each other. Then each mark vector is marked for at least 0.5 μs .

(2) Adjustment of the Scanner Delays

Under some circumstances with predecessor boards, a short delay could result in laser control errors if a non-mark command (e.g. `jump_rel(0, 0)` or `list_nop`) comes between two mark commands. Here, the laser would be switched off during the second mark command, but unexpectedly not switched on again if the LaserOff delay is longer than the sum of the MarkDelay (and any jump delay) and the LaserOn delay (see figure 40).



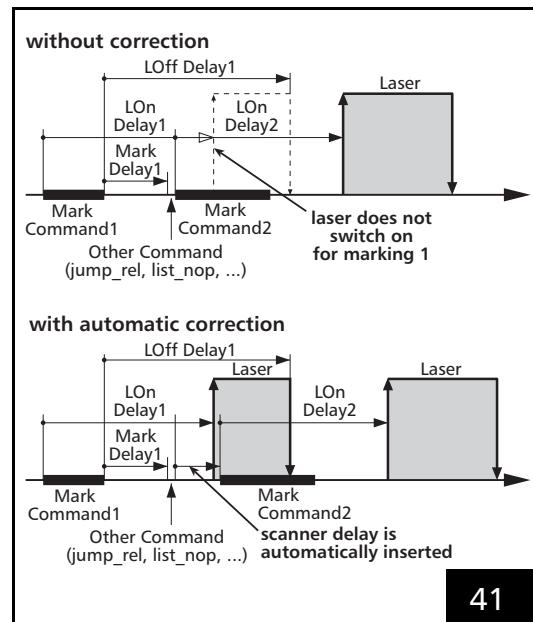
40

Automatic adjustment of the scanner delays for small mark delays

One way of avoiding this problem is to generally select a mark delay that is larger than the difference between the LaserOff delay and the LaserOn delay ($\text{markD} > \text{LOffD} - \text{LOnD}$, as with the predecessor boards). But here, a minimum mark delay size is required that otherwise would not be necessary where mark commands directly follow another; this thus unnecessarily increases the total marking time. On the other hand, the RTC5 now checks each mark command's laser delays and automatically extends the scanner delay so that a preceding LaserOff delay first finishes before another Laser-On follows. Thus, marking time is only increased when necessary. The scanner delays can now be optimized independently of the laser delays.

The same applies for the edgelevel of the variable polygon delay as well as for a too-short polygon delay if a polygon chain is continued by a list change but no `auto_change` or `start_loop` is used and the second list is started immediately after the first list finishes processing. For predecessor boards in the latter case, the laser might remain switched-off for the rest of the polyline.

For very large LaserOn delays extending across a mark command as well as a subsequent non-mark command and up to the next mark command, appropriate scanner delays are inserted to prevent overlap of a not-yet-expired LaserOn delay with the start of the second marking command. Otherwise, the currently executing LaserOn delay would have gotten overwritten and the laser would not have even switched on for the first marking command, but instead only for the second marking command and only after expiration of the LaserOff delay (see figure 41).

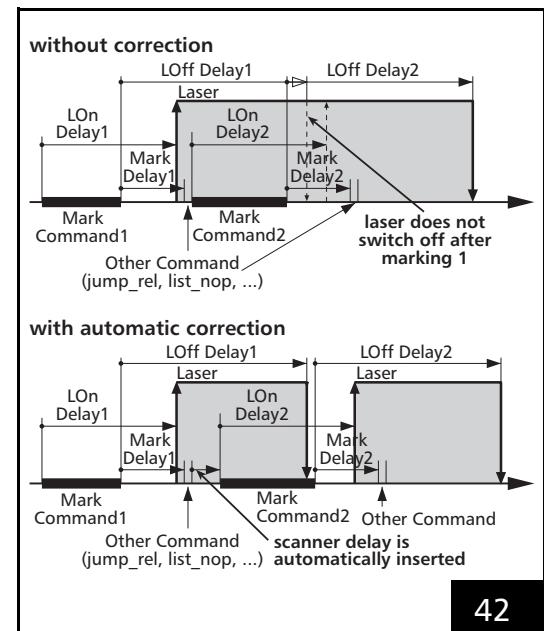


41

Automatic adjustment of the scanner delays for large LaserOn delays

Similarly, overlaps would occur for very large LaserOff delays set for the beginning of a non-mark command and extending across the next mark command up to yet another non-mark command. Here, the laser would not switch off between the two mark commands and instead only switch off after the second non-mark command and only after expiration of the LaserOn delay (see figure 42). To prevent this situation, a check is performed at the start of a mark command to determine if the laser would have already switched off prior to the end of the command. If not, then the mark command gets postponed by an appropriate scanner delay.

On the other hand, laser switch-off during polyline with sharp corners (edgelevel parameter) only occurs if the laser was already on during this polyline (i.e. when all previously initiated LaserOn delays have already expired and no LaserOff delays remains active).



42

Automatic adjustment of the scanner delays for large LaserOff delays

Notes

- The above description applies to firmware files (`RTC5RBF.rbf`) where version > 509 (see [get_RTC_version](#)). Discontinue using earlier firmware versions, because in the above example the laser would get switched off for the first mark command, but not for the second one. Likewise, the laser would get switched off for the first - but not the second - non-mark command and sometimes not for [set_end_of_list](#).
- Thanks to these previously described automatic adjustments, the RTC5 always fulfills the scanner delay conditions ($\text{markD} > \text{LOffD} - \text{LOnD}$, $\text{edgelevel} > \text{LOffD} - \text{LOnD}$, $\text{polygonD} + \text{LOnD} > \text{LOffD}$) at run-time that had to be explicitly observed with the RTC4.
- Automatic adjustment of the scanner delays might result in a total marking time longer than that expected from just adding-up the predefined vector lengths and delays.

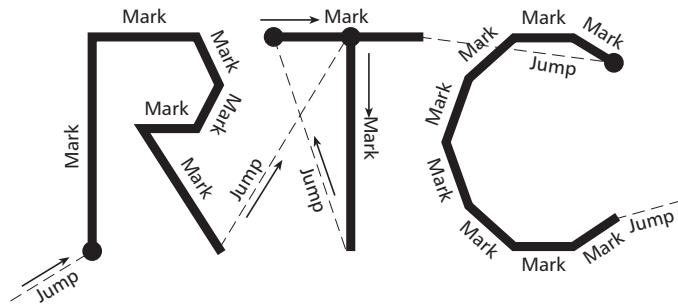
Optimizing the Delays

The following figures show the various effects of non-optimized delays on the lettering "RTC".

LaserOn delay too short

At the beginning of a mark vector the laser is switched on, even though the mirrors have not yet reached the necessary angular velocity.

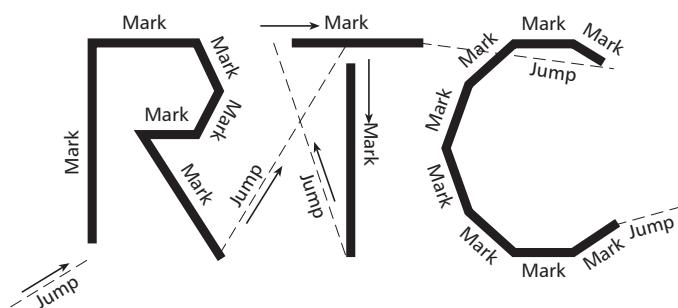
Burn-in effects at the start points of the respective vectors result.



LaserOn delay too long

The laser is turned on too late at the beginning of a mark vector.

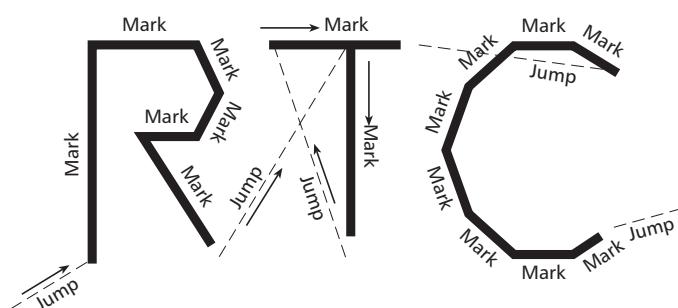
The first part of the vector is not marked.



LaserOff delay too short

The laser is turned off after the last mark command of a line or polyline, although the mirrors have not yet reached the end position of the vectors.

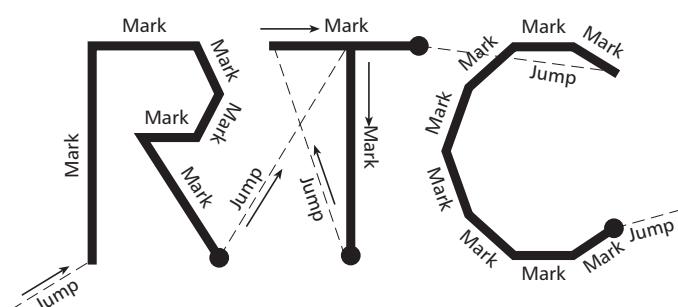
The respective vectors are not marked completely.



LaserOff delay too long

The laser is turned off too late after the last mark command of a line or polyline. The laser is still on, even though the mirrors have already stopped or move only very slowly.

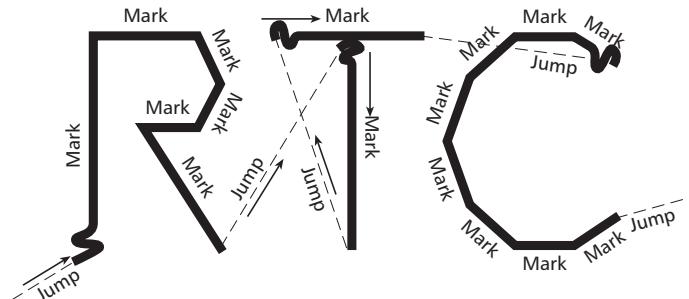
The results are burn-in effects at the end points of the respective vectors.



Jump delay too short

After a jump, the first mark vector has already started although the scanners have not yet settled.

A running-in oscillation (overshoot) is visible.



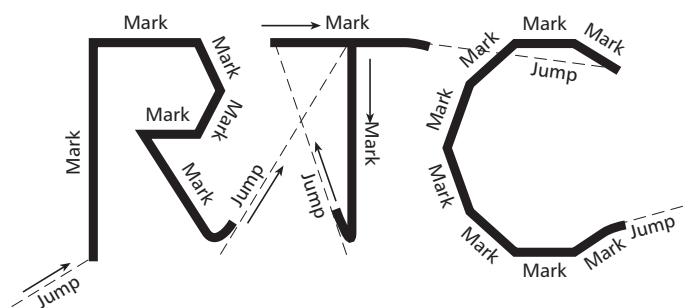
Jump delay too long

There are no visible effects if the jump delay is too long. However, the scanning time is extended.

Mark delay too short

Though the mirrors have not yet reached the end position of the last vector of a line or polyline, the command for the succeeding jump vector is already executing.

The end of the mark vector is turned towards the direction of the jump vector.



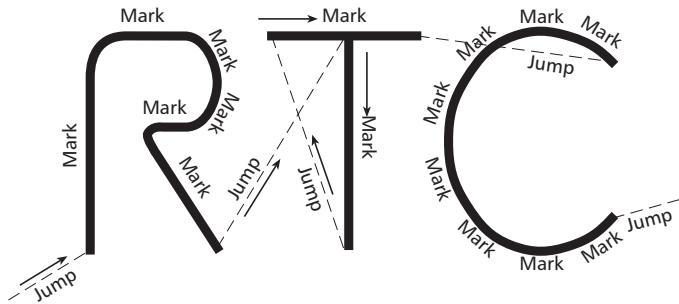
Mark delay too long

There are no visible effects if the mark delay is too long, but the scanning time is increased.

Polygon delay too short

The subsequent mark command in a polyline is already executing, although the mirrors have not yet reached the end position of the preceding mark vector.

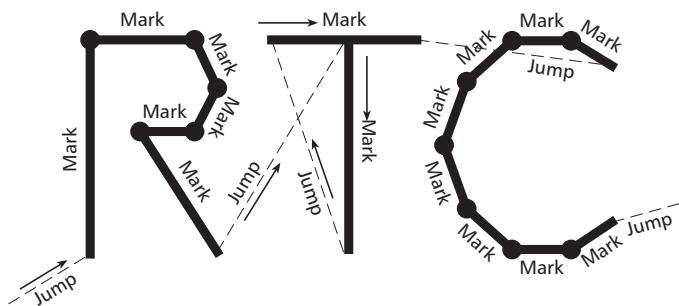
The corners of the polyline are rounded off.



Polygon delay too long

If the polygon delay is too long, the mirrors are moving too slowly or are even stopping between subsequent mark commands.

Since the laser is not turned off between these vectors, burn-in effects occur.



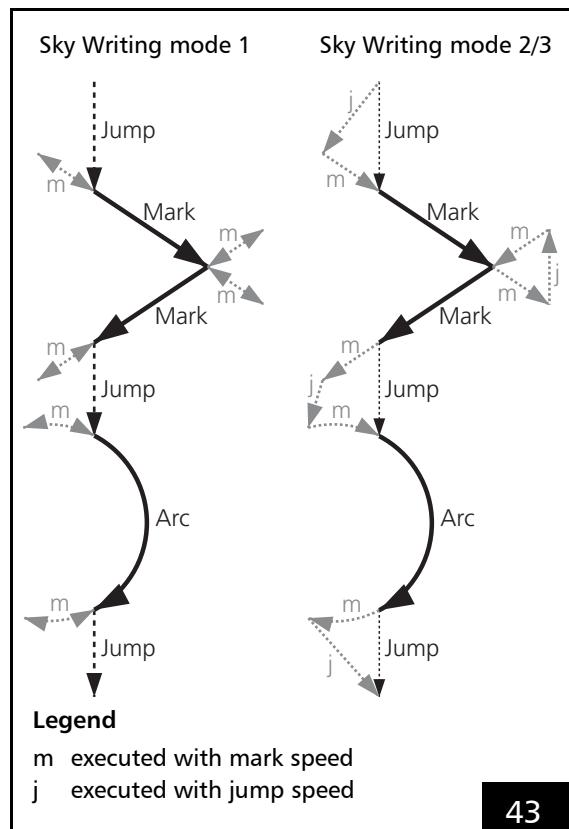
In the variable polygon delay mode, a maximum length ("edgelevel") can be defined.
See [page 119](#) for details.

7.2.4 Sky Writing

The RTC5 provides Sky Writing for applications with elevated accuracy requirements. With Sky Writing, every mark vector is precisely executed at a constant marking speed over the entire vector length.

For mark commands with Sky Writing activated, the RTC5 automatically performs non-marking Sky Writing scan motions (see [figure 43](#)) before and after the to-be-executed vectors or arcs. By the Sky Writing command's parameters you can specify the durations of these run-in and run-out motions as well as synchronization between scan motions and laser control.

Sky Writing can be activated in modes 1, 2 or 3.



Mode 1

In Sky Writing mode 1, the RTC5 performs the following Sky Writing motions for each marking vector (or arc) – regardless of previous or subsequent commands:

- In the run-in phase, the vector (or arc) is preceded by a “forerun” movement performed by the galvanometer scanners at marking speed (see [figure 43](#)): the scanners are driven a short distance parallel to the vector (or along the arc extension), initially from the startpoint in the opposing direction, then back to the startpoint.
- After the vector (or arc) is processed at marking speed, it is then appended in the run-out phase with a short deceleration and retrace movement of the scanners (at marking speed).

Sky Writing mode 1 can be activated (and deactivated) by the commands `set_sky_writing_para`, `set_sky_writing` or the corresponding list commands.

Except for para commands, Sky Writing mode 1 activation affects all mark, arc and ellipse commands (even time-based and 3D commands). In contrast, during execution of para commands (e.g. with activated vector-defined laser control, see [page 166](#)) and during execution of micro vector commands (see [page 221](#)), Sky Writing mode 1 is *not* taken into account (but is also not deactivated).

Mode 2

In Sky Writing mode 2, the RTC5 calculates *time-optimized* Sky Writing motions. Here, too, each to-be-executed vector (or arc) gets preceded and appended with a run-in motion and a run-out motion in extension of the vector (or arc) at mark speed. Within a SkyWriting2 marking sequence, however, neither scanner forerun motions (in the run-in phase) nor retrace motions (in the run-out phase) occur. Instead, the RTC5 executes Sky Writing jumps (at the currently defined jump speed) from jump vector startpoints to run-in startpoints, from run-out endpoints to run-in startpoints, and from run-out endpoints to jump vector endpoints (see [figure 43](#)).

You can only activate (and deactivate) Sky Writing mode 2 after successful activation of Sky Writing mode 1 (see above) by subsequently calling **`set_sky_writing_mode`** or the corresponding list command.

Sky Writing mode 2 activation affects the same commands as Sky Writing mode 1 (except ellipse commands, see above). *Time-optimized* Sky Writing, however, can only occur within a sequence of non-parameterized mark, arc and jump commands (may also contain timed or 3D commands). If such a sequence gets interrupted by some other “non-Sky-Writing2-capable” list command (e.g. a para, ellipse or any short list command), then the RTC5 suspends Sky Writing mode 2 and complete the preceding command in Sky Writing mode 1 (with a scanner retrace motion if required). This suspension of Sky Writing mode 2 does not deactivate it (that can only occur by **`set_sky_writing`** or **`set_sky_writing_mode_list`**): Subsequent “SkyWriting2 capable” commands again execute in Sky Writing mode 2 (however, the first command of such a subsequent “SkyWriting2 capable” sequence execute in Sky Writing mode 1, i.e. possibly with a scanner forerun motion).

With Sky Writing mode 2 activated, ellipse commands are always fully executed in Sky Writing mode 1.

Mode 3

The time cost of Sky Writing motions for vectors and arcs having only small directional changes within a polyline is probably disproportionately high for the gained accuracy. To optimize Sky Writing even in this scenario (for reducing total process time), the RTC5 provides Sky Writing Mode 3, which allows specification of a switching limit (`Limit`) by

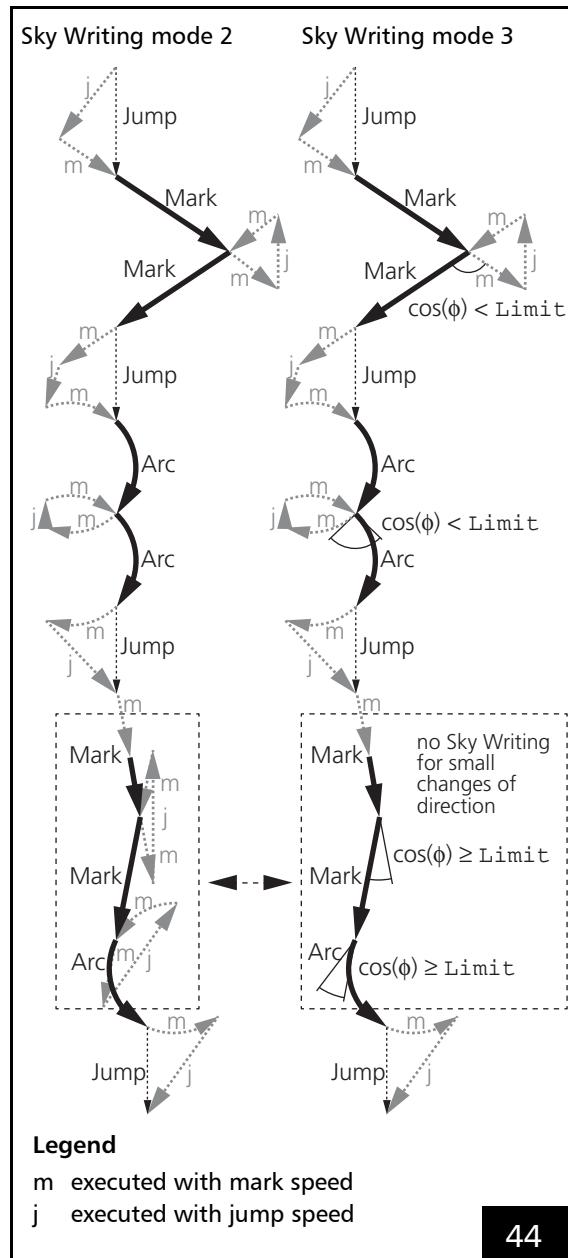
`set_sky_writing_limit` or

`set_sky_writing_limit_list`. After activation of Sky Writing Mode 3 (by **`set_sky_writing_mode`** or **`set_sky_writing_mode_list`**), Sky Writing motions similar to Sky Writing Mode 2 (only) occur for large angular changes between consecutive vectors or arcs of a polyline ($\cos(\phi) < \text{Limit}$). In contrast, smaller angular changes ($\cos(\phi) \geq \text{Limit}$) result in a possibly variable polygonal delay (see [page 118](#)) instead of Sky Writing motion. See also [figure 44](#).

Notes:

- In case a polyline ends with a short mark vector (shorter than $\text{mark speed} \times \text{Timelag}$) - and a polygon delay has been executed but not a Sky Writing motion - then the length of the short vector (caused by the tracking error) may possibly not achieve the precision expected with Sky Writing.

Sky Writing Mode 3 produces a Sky Writing motion before and after every polyline and also if the polyline is interrupted by non-Sky-Writing2-capable” list commands (see the description for Sky Writing Mode 2).



Sky Writing motions (grey) for Sky Writing modes 2 and 3

Synchronization

Figure 45 shows the timing diagram for scan-head and laser control in Sky Writing mode 1. The timing diagram for Sky Writing mode 2 and 3 is similar, but here the scanner perform no reverse motion in the run-in and run-out phases.

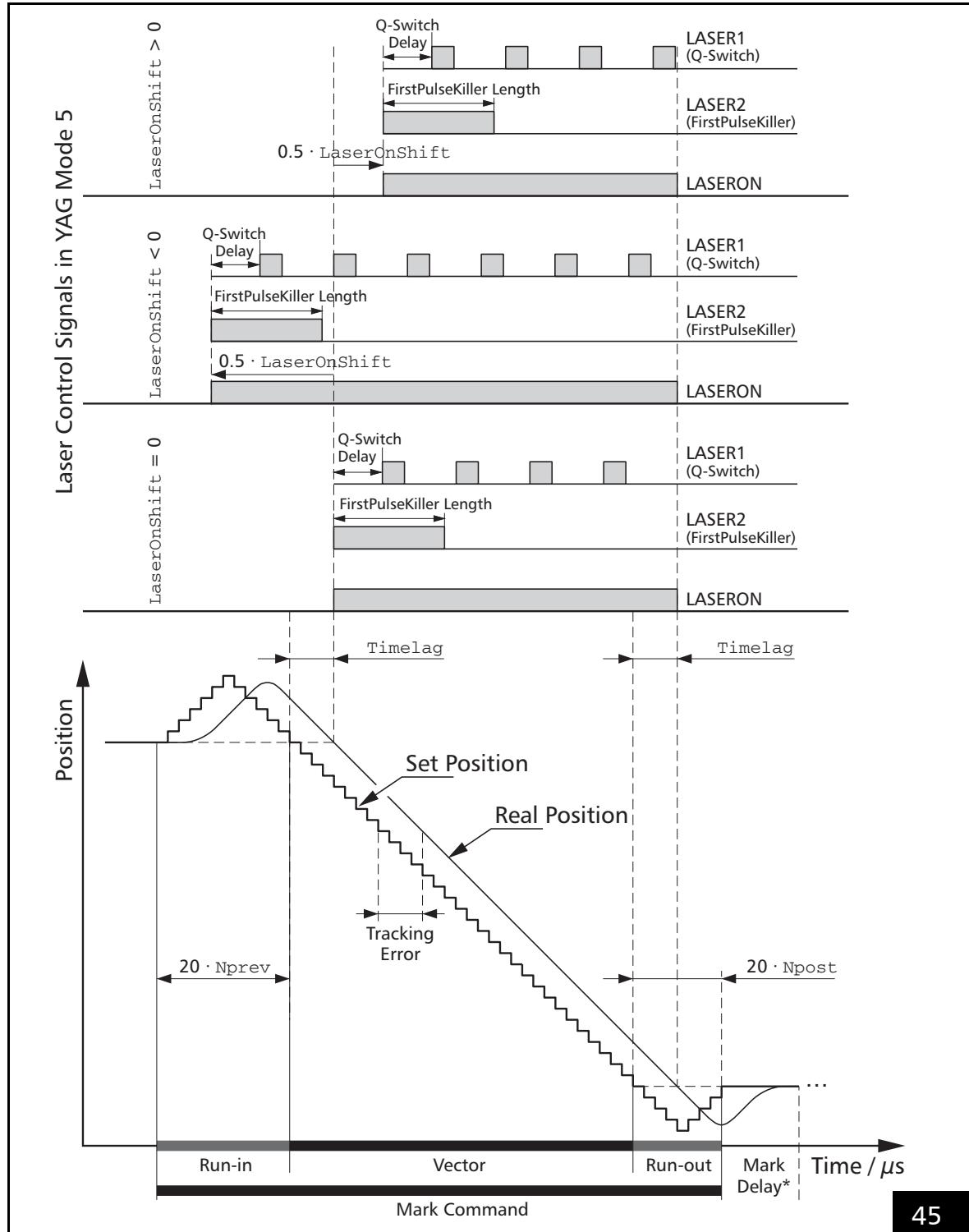
The Timelag, Nprev, Npost and LaserOnShift parameters are specifiable for the commands **set_sky_writing_para** and **set_sky_writing** and the corresponding list commands:

- The Nprev parameter is a whole number in units of 10 µs and defines the duration of the run-in:
 - Run-in duration / µs = $20 \times N_{prev}$ (mode 1)
 - Run-in duration / µs = $10 \times N_{prev}$ (mode 2, 3)
- The Npost parameter is a whole number in units of 10 µs and defines the duration of the run-out:
 - Run-out duration / µs = $20 \times N_{post}$ (mode 1)
 - Run-out duration / µs = $10 \times N_{post}$ (mode 2, 3)
- The parameters Timelag (whole number in units of 1 µs) and LaserOnShift (whole number in units of 0.5 µs) define the delay of the “laser active” laser control signals’ switch-on and switch-off time points relative to the set starting position and set ending position:
 - Delay of switch-on time point relative to the set starting position / µs
= Timelag + $0.5 \times \text{LaserOnShift}$
 - Delay of switch-off time point relative to the set ending position / µs
= Timelag



For `LaserOnShift` = 0, the "laser active" laser control signals are switched on/off with a delay of `Timelag` relative to the set starting position and to the set ending position. By setting the parameter `Timelag` to the actual tracking error (and the parameters `Nprev` and `Npost` to sufficiently large values), you ensure that the "laser active" laser control signals switch on/off precisely at the endpoints of the desired vector, that the tracking error becomes constant even before the vector's startpoint is reached, and that the vector executes right up to its endpoint with constant marking speed and therefore also with a constant tracking error.

The Parameter `LaserOnShift` allows adjustment of the "laser active" laser control signals' switch-on time point. Positive values delay switching on of the laser control signals. Negative values result in an earlier switch-on of the laser control signals, but no earlier than the beginning of the run-in phase (an additional delay such as a negative `LaserOn` delay – see **set_laser_delays** – is not possible). Negative values can be used (for example in the YAG modes, where a `FirstPulseKiller` signal and a positive Q-Switch delay may be previously set) for shifting the "laser active" laser control signals' switch-on time point so far forward (even to within the run-in phase) that the first laser pulse coincides with the target start position.



Scan-head and laser control in Sky Writing mode 1 (with parameters Timelag, Nprev, Npost and LaserOnShift)
 (the laser control signals LASER1 and LASER2 in YAG mode 5 are exemplarily shown)
 (* a one-time mark delay is only added in some circumstances after deactivation of Sky Writing 1 mode; see [page 550](#))

Notes

- By `set_sky_writing` and `set_sky_writing_mode_list`, you can switch back and forth between Sky Writing modes 1 and 2 or 3. Temporarily switching off is also possible, as long as `Timelag > 0`.
 - When searching for appropriate Sky Writing parameters, initially the `set_sky_writing` command and Sky Writing mode 1 should be used: initially parameter `Timelag` should be adjusted (with `LaserOnShift = 0`) such, that the marking vectors' *ends* are exactly marked and only afterwards the parameter `LaserOnShift` should be adjusted (keeping constant the parameter `Timelag`), so that the marking vectors' *beginnings* are exactly marked, too.
- In a second step (if necessary) `set_sky_writing_para` can be used to optimize the parameters `Nprev` and `Npost`: `Nprev` and `Npost` may be decreased (relating to the default settings of `set_sky_writing`, see below) to minimize marking times. Alternatively, `Nprev` may be increased to enable a correspondingly large negative `LaserOnShift`.
- The parameter `Timelag` should be set to the tracking error of the galvanometer scanners.
 - For the `set_sky_writing` and `set_sky_writing_list` commands, `Nprev` and `Npost` are predefined. Here, the following applies:
 - `Nprev = approx. 0.15 × Timelag`
 - `Npost = approx. 0.1 × Timelag`

This results in the following run-in and run-out durations:

- Run-in duration / μs
= approx. $3 \times \text{Timelag}$ (mode 1)
- Run-in duration / μs
= approx. $1.5 \times \text{Timelag}$ (mode 2, 3)
- Run-out duration / μs
= approx. $2 \times \text{Timelag}$ (mode 1)
- Run-out duration / μs
= approx. $1 \times \text{Timelag}$ (mode 2, 3)

These are safe values for a suitable `Timelag` parameter.

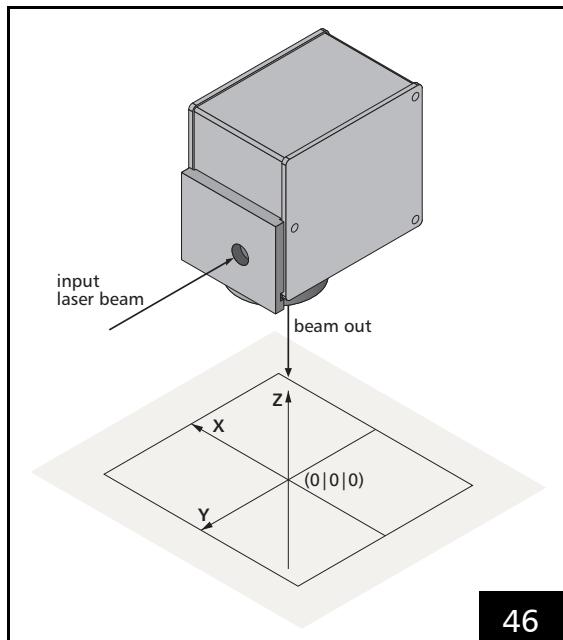
- With `set_sky_writing_para` or `set_sky_writing_para_list`, you can also specify shorter run-in and run-out phases to reduce marking times. Shorter run-in and run-out phases are particularly beneficial when lines are marked in only one direction without interruption and jump and marking speeds are set equally. Take note, though, that in Sky Writing mode no automatic adjustment of laser and scanner delays (as described on [page 121](#)) is performed:
 - If the next marking command's run-in phase begins when the preceding command's `LaserOn` delay has not yet expired, then the laser does not switch on for the preceding command (see [figure 41](#)).
 - If the next marking command's run-out phase begins when the preceding command's `LaserOff` delay has not yet expired, then the laser does not switch off for the preceding command (see [figure 42](#)).
 - In Sky Writing mode 1, a positive `LaserOnShift` time (in μs) should exceed the smallest occurring marking time by no more than $(20 \times Npost - \text{Timelag})$ – and in Sky Writing mode 2 and 3 by no more than $(10 \times Npost - \text{Timelag})$. Otherwise, the laser is not switched on for this marking (this applies as of Version OUT 535; with older versions, a very short marking command might result in the laser switching on only after the command has been executed and then remaining switched on until explicitly switched off by one of the following commands).

- Pulse lengths and frequencies (in YAG modes additionally the Q-Switch delay and the FirstPulseKiller signal parameters) previously defined for the “laser active” laser control signals are kept unchanged in the Sky Writing mode. On the other hand, previously defined LaserOn, LaserOff, mark, polygon or variable polygon delays are *not* taken into account in the Sky Writing mode. They are fully functional again after deactivation of Sky Writing mode. Deactivation of Sky Writing mode results in some circumstances in addition of a mark delay defined prior to activation (see [page 131](#)).
- In Sky Writing mode 1, the run-in and run-out phases take place fully within the respective marking command. As a result, execution of the marking command is extended by a time period (in 10 µs) of $(2 \times N_{\text{prev}} + 2 \times N_{\text{post}})$. In Sky Writing mode 2 and 3 execution is extended by a time period (in 10 µs) of at least $(N_{\text{prev}} + N_{\text{post}})$.
- In Sky Writing mode 1, jump delays are performed normally. The jump delay value (in 10 µs, see [set_scanner_delays](#)) can be reduced by approx. $(2 \times N_{\text{prev}})$ or even to 0. In Sky Writing mode 2 and 3, the jump delay is automatically (internally, dynamically) reduced by up to N_{prev} .
- Time-based mark and arc commands can also be performed in Sky Writing mode (see [page 223](#)). Here, however, a shorter marking period is coupled with a higher marking speed and therefore with longer starting and ending distances and acceleration phase in the run-in and run-out phases. Therefore, N_{prev} and N_{post} can be separately adjusted with respect to the specified marking duration (with [set_sky_writing](#) the `Timelag` parameter might have to be adjusted with respect to the specified marking duration).
- During execution of para commands (e.g. with activated vector-defined laser control, see [page 166](#)) and during execution of micro vector commands (see [page 221](#)), the Sky Writing mode is not taken into account (but also not deactivated).
- With Sky Writing mode 2 or 3 activated, the two following RTC5 functionalities of the 2D jump commands [jump_abs](#) and [jump_rel](#) get deactivated:
 - Tuning auto-switching in jump mode (see [page 176](#))
 - Coordinate transformations with `at_once = 2` (see [page 185](#))

7.3 Scan Head Control

7.3.1 Reference System

Figure 46 shows the reference system for the image field which is used by the RTC5. The Y-axis points in the *reverse* direction of the input laser beam, the Z-axis points in the *reverse* direction of the output laser beam. X-axis, Y-axis and Z-axis form a right-handed reference system. The origin of the reference system, i.e. the point (0|0|0), is in the center of the image field.



46

Reference system for the RTC5 coordinates

7.3.2 Image Field Size and Calibration

The size of the usable image field is determined by the maximum scan angle and the focal length of the objective or the working distance (i.e. the distance between the input laser beam axis and the image field).

The X and Y vector coordinates of a vector must be specified as signed 20-bit numbers (i.e. as numbers between -524288 and +524287)⁽¹⁾, however the Z coordinates in a 3D system as signed 16-bit numbers (i.e. as numbers between -32768 and +32767).

The ratio of a point coordinate in *bits*⁽²⁾ and the actual position of the point in *millimeters* is defined by the calibration factor *K*.

Let a_0 denote the side length of the image field given by the maximum scan angle. The theoretical calibration factor is then $K_0 = 2^{20}/a_0$ [bits per mm⁽²⁾]. SCANLAB provides a rounded value for the calibration factor *K*. This value is slightly larger than, but close to, the theoretical value. The actual calibration factor *K* can be read out from the used correction table by the commands `get_table_para` or `get_head_para`.

Given the calibration factor *K*, the side length *a* of the usable image field can be calculated:

$$a = 2^{20} / K$$

Notes

- The calibration factor is the same for the X and Y direction. For 3D scan systems:

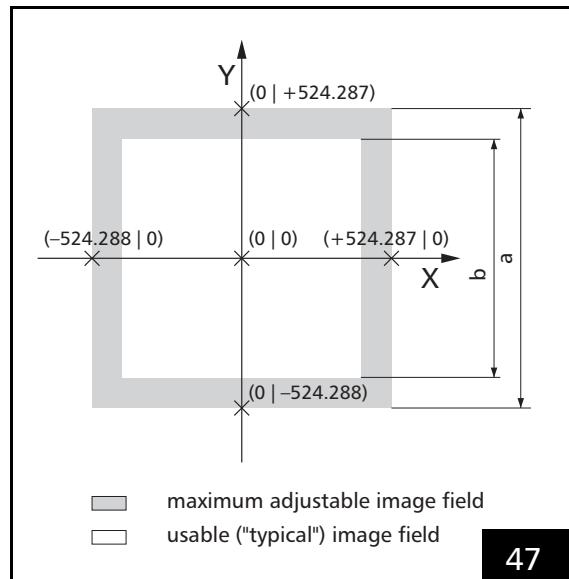
$$K_z = K_{xy} / 16.$$

(1) Actually, X and Y coordinates can be specified as signed 24-bit numbers (i.e. as numbers between -8388608 and +8388607). However, the complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to the value range [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table (see also page 135 and page 141).

(2) The expression "bits" is here synonymous with "digital control value" (see footnote on page 103).

Typical Image Field

In general, the size of the usable (or “typical”) image field – dependent on the objective and the optical configuration of the scan system – is smaller than the maximum adjustable image field.



RTC4 Compatibility Mode

With the RTC5 (in RTC5 mode), the image field coordinates for the X and Y axes (but not the Z axis) and all related parameters (e.g. jump speed or wobble amplitude) are specified as 20-bit values; in RTC4 compatibility mode they are specified as 16-bit values (just like with the RTC4).

In RTC4 compatibility mode, the RTC5 multiplies the specified values by 16 (the allowed range of values is correspondingly reduced). Therefore, in RTC4 compatibility mode a 16-fold smaller calibration factor has to be used for X and Y coordinate values than in RTC5 mode. See also section “Increased Parameter Resolution”, page 33.

7.3.3 Virtual Image Field

X and Y coordinates can be specified as signed 24-bit numbers (i.e. as numbers between -8388608 and $+8388607$) for all vector and arc commands as well as timed vector and arc commands (but not `goto_xy` and `goto_xyz`). The current coordinates are clipped to the value range of the real image field $[-524288 \dots 524287]$ during runtime – after a coordinate transformation (if applicable), after Processing-on-the-fly corrections (if applicable), and directly prior to use of the correction table (see also page 141).

A virtual 24-bit image field is therefore available for Processing-on-the-fly applications. Vector and arc commands can also be loaded for objects up to 16 times larger than the real image field (in the Processing-on-the-fly direction). For details see page 210.

In addition, the extended value range of the virtual image field can be used for utilizing the complete real 20-bit image field even if a coordinate transformation (as rotation, shrinkage or shift, see page 183) is activated. If necessary, appropriate coordinate values within the virtual image field can be defined, which are subsequently transformed to coordinate values within the real image field by the set coordinate transformations (with the RTC5’s predecessor boards, if such coordinate transformations are set, some edge points of the real image field are inaccessible).

Size of the usable image field

The scan head has a usable image field. If the laser focus moves outside this field, some vignetting of the laser beam can occur. The interior of the scan head can be damaged due to excessive absorption of laser power. Refer to your scan head operating manual’s section on objectives.

- ▶ Compare the calculated side length a of the maximum adjustable image field with the side length b of the usable image field given in the technical specifications of your scan head manual (see figure 47).
- ▶ If the laser focus shall be restricted to points within the usable image field, the absolute values of the X and Y coordinates (in bits) must be smaller than the maximum value M , where M is the calibration factor K multiplied by half the side length of the usable image field:

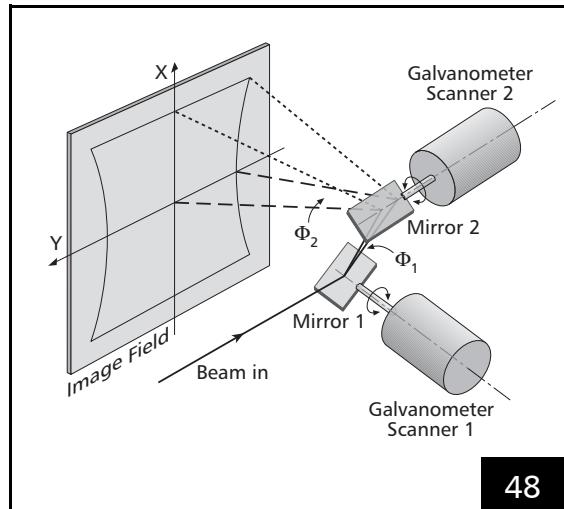
$$M = K \times b/2$$

7.3.4 Image Field Correction and Correction Tables

Field Distortion

The deflection of a laser beam with a two-mirror system results in three effects:

- (1) The arrangement of the mirrors leads to a certain distortion of the image field – see **figure 48**. This distortion arises from the fact that the distance between mirror 1 and the image field depends on the size of the scan angles of mirror 1 and mirror 2. A larger scan angle leads to a longer distance.
- (2) The distance in the image field is not proportional to the scan angle itself, but to the tangent of the scan angle. Therefore, the marking speed of the laser focus in the image field is not proportional to the angular velocity of the corresponding scanner.
- (3) If an ordinary lens is used for focusing the laser beam, the focus lies on a sphere. In a flat image field, a varying spot size results.



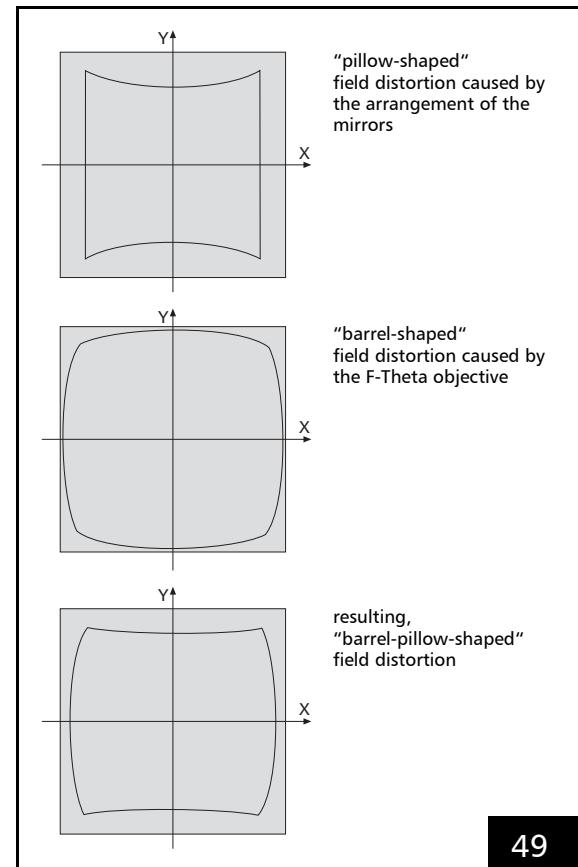
48

Field distortion in a two-mirror deflection system

By focusing the deflected laser beam with an F-Theta objective, the two last-named effects can be avoided:

- A direct proportionality between scan angle and scanned distance in the image field is obtained.
- Additionally, the focus lies on a flat surface.

The F-Theta objective, however, causes a barrel-shaped distortion of the image field. This distortion is added to the pillow-shaped distortion caused by the arrangement of the mirrors, resulting in a so-called "barrel-pillow-shaped" distortion of the image field (see **figure 49**).



49

Field distortion caused by the arrangement of the mirrors and by the F-Theta objective



Field Correction Algorithm

The RTC5 Board provides a correction algorithm to compensate for the above mentioned field distortion. The algorithm is based on a correction table.

An orthogonal grid of 257 x 257 points is superimposed on the ideal square image field. The adjusted X and Y coordinates for a corrected output of these grid points are stored in a correction table. To move the focus to any point within the image field, the RTC5 calculates the correct coordinates by interpolating from the grid points in the correction table. The result is transmitted to the scan head. The correction algorithm is executed for every single microstep, see also [chapter 7.1.2 "Microsteps", page 107](#).

By default, SCANLAB creates an individual correction table for every delivered system. It is stored in a correction file named (*.CT5) in the RTC5 software package. See also the command [load_correction_file](#).

The calculation of the correction table is based solely on general system data (such as mirror geometry, calibration and the objective's specifications).

Standard correction files therefore reflect neither the unique properties possessed by the customer's individual system, nor alignment errors.

For those customers requiring more accurate correction tables tailored to the unique properties of their particular scan systems, SCANLAB offers the correXion program line. These generate RTC5 correction files based on data derived from actual test measurements performed by the customer (for further information, refer to the corresponding manual or contact SCANLAB).

Activating Image Field Correction

To activate image field correction, the appropriate correction tables must be loaded (by [load_correction_file](#)) from the corresponding correction files into RTC5 memory at the beginning of a user program and subsequently assigned (by [select_cor_table](#) or [select_cor_table_list](#)) to the scan head connectors used by the user program (see the following sections).

2D correction tables activate an image field correction for X and Y coordinates, 3D correction tables (if the 3D option is enabled, see below) additionally for Z coordinates.

2D and 3D Correction Files

SCANLAB calculates 2D and 3D correction tables and supplies them in 2D and 3D correction files. RTC5 correction files use the extension *.ct5 and the following naming scheme:

- D2_xxx.ct5 for 2D correction files
- D3_yyy.ct5 for 3D correction files

Here, **xxx** and **yyy** are numbers. Each correction file is calculated for a specific optical configuration. The configuration is specified in the accompanying Readme.txt file and in parameters of the correction file's header (see also ["Correction File Header", page 140](#)).

Loading Correction Tables

Up to four correction tables can be loaded from the corresponding correction files into RTC5 memory by [load_correction_file](#) (see [page 192](#)).

If the 3D option is *not* enabled, then only 2D correction tables can be loaded. They can be loaded from 2D or 3D correction files (3D correction files also contain 2D correction tables; if desired, the 3D data sections are simply ignored). In this way, 3D correction files are also usable for 2D scan systems.

If the 3D option is *enabled*, then 3D correction tables can also be loaded (even in a mix with 2D correction tables). A 3D correction table can also be loaded from a 2D correction file. The 2D correction table is thereby automatically expanded to incorporate a linear Z correction. The actually suitable Z correction can subsequently be loaded by the [load_z_table](#) command (see [page 196](#)).

Assigning Loaded Correction Tables

Loaded correction tables are assigned to the two scan head control ports by `select_cor_table` or `select_cor_table_list`.

If neither the "second scan head control" option nor the 3D option is enabled, then a 2D correction table (one of the possibly four loaded correction tables) can be assigned (only) to the primary scan head connector.

If the "second scan head control" option (but not the 3D option) is enabled, then a 2D correction table can be assigned to the two scan head control ports each. Then the secondary scan head connector outputs the same signal types as the primary scan head connector. Selectively, the same or two different correction tables can be assigned to the two scan head control ports.

If the 3D option (but not the "second scan head control" option) is enabled, then a 2D or 3D correction table can be assigned (only) to the primary scan head connector. If a 2D correction table is assigned, then the primary scan head connector outputs signals for an XY scan system. If a 3D correction table is assigned, then the RTC5 additionally outputs signals for the third axis (Z axis) by both channels of the secondary scan head connector.

If *both* options ("second scan head control" and 3D option) are enabled, two (different) 2D correction tables can then be assigned to the two scan head control ports. But it is not possible to simultaneously assign two 3D correction tables. Nor is it possible to simultaneously assign a 2D and a 3D correction table. For controlling a 3D system, exactly one of the two connectors must be assigned a 3D correction table. Signals can then be outputted by this scan head connector to an XY scan head – and by both channels of the other scan head connector to the third axis (Z axis). When assigning the correction table, users specifies which signals (XY or Z) are to be outputted by which connector.

Notes

- If no correction table has been loaded into the RTC5's memory, then the RTC5 transmits unintended values to the scan system. If no 2D or 3D correction file for the specific optical configuration is available, then at least a 1to1 (2D or 3D) correction table should be loaded (see "["1to1 Correction Tables", page 139](#)"). Correction files should be loaded at the beginning of a user program, before or after loading the program file (by `load_program_file`)⁽¹⁾. They should absolutely be loaded and assigned prior to first-time starting of a list or issuance of a control command (e.g. `goto_xy`) that sets the scan system's galvanometer scanners in motion.
- If only one of the two scan head connectors has been assigned a 2D correction table (e.g. initialization by `load_program_file` only assigns a correction table to the primary scan head), then the other scan head connector transmits no signals.
- As of version DLL 521, OUT 521, `load_correction_file` automatically calls `select_cor_table` after loading the correction table. This results in both assignment of a correction table and triggering of a jump with `jump_speed` (no *hard* jump) to the corrected galvanometer scanner position.
 - However, if `load_correction_file` is called prior to loading a program file by `load_program_file`, then the automatic call of `select_cor_table` has no effect⁽¹⁾. And though `load_program_file` assigns a correction table – as with `select_cor_table(1,0)` – it does not trigger scanner motion to the corrected position. Therefore, immediately after the `load_program_file` command, the laser focus is initially centered in the image field (0|0) – even if a correction table #1 had been previously loaded by `load_correction_file`. The desired image field correction becomes active only after a subsequent `select_cor_table`, `select_cor_table_list`, jump or marking command.

(1) `load_program_file` deletes correction tables number 3 and 4.

- If **load_correction_file** is called *after* **load_program_file**, then the call of **select_cor_table** executes with the default parameter values (`HeadA = 1, HeadB = 0`) or – if **select_cor_table** or **select_cor_table_list** was called in the meantime – with the parameters used there. If you require a different assignment, then you must call **select_cor_table** or **select_cor_table_list** (again) using the appropriate parameter values. As of version DLL 527, OUT 529, the command only returns to the user program, after the **select_cor_table**-induced jump to the corrected galvanometer position has completed.
- RTC5 correction files differ from those of prior RTC products (with the file extension *.ctb) in terms of size, structure and content. If ct5 and ctb correction files have the same file name (except for the different extensions), then they were calculated for the same optical configuration. For converting older correction files, see “[Converting Correction Files](#)”, page 141.

Notes on Versions Older than DLL 521, OUT 521

- Here **load_correction_file** does not call **select_cor_table**. Here, the command **select_cor_table** or **select_cor_table_list** should be called directly after loading a correction table (or after a subsequent **load_program_file** command) to ensure that the current position, in accordance with the most recently loaded correction table, is attained immediately by a smooth transition (with `jump_speed`) instead of by a hard jump initiated by the subsequent jump or marking command. If you do not explicitly call **select_cor_table** or **select_cor_table_list**, then some boards might also output unexpected initialization-related galvanometer scanner positions.
- With such older versions, **select_cor_table** or **select_cor_table_list** must also absolutely be called after loading if **load_correction_file** is used to replace an already loaded and assigned 2D correction table by a 3D correction table (or a 3D table by a 2D table or by another 3D table). Otherwise the newly loaded correction table is not fully effective. The same applies when the correction table is loaded *after* **load_program_file** (though initialization assigns a table as by `select_cor_table(1, 0)`, it does not trigger scanner motion to the corrected output position). On the other hand, calling

select_cor_table or **select_cor_table_list** is not obligatory, if a 2D correction table is replaced with another 2D correction table. But even in this case, **select_cor_table** or **select_cor_table_list** should be called to avoid hard jumps.

1to1 Correction Tables

Unlike previous RTC products, the RTC5 does not need to have 1to1 correction tables loaded from 1to1 correction files. Instead, they can be directly generated with **load_correction_file** and `Name = 0`. For `Dim = 3` (with activated 3D option), the specified correction table is replaced with a 3D 1to1 table, otherwise with a 2D 1to1 table (see command description).

A 1to1 correction file (`Cor_1to1.ct5`) is nevertheless supplied with the RTC5, because some applications require a filename and do not accept 0.

Inverse Tables

The RTC5’s correction file format permits storage of a table for transforming returned scanner-axis position data into the input coordinate system. These are called “inverse tables” (for usage, see “[Position Monitoring](#)”, page 174). SCANLAB-generated ct5 correction files automatically contain an inverse table, but files converted from older formats might not always have one (see “[Converting Correction Files](#)”, page 141). If no valid inverse table exists, then a 1to1 correction table is used instead. Here, precise inverse calculation of uncorrected position data is not possible. The correction file’s header contains entries regarding the inverse table’s validity and calculation method (see parameter 11, [page 140](#)).

Correction File Header

The RTC5 correction file's header contains 16 parameters describing the file name, the contained data and the optical data of the corresponding optical configuration. Some of the listed parameters were previously only obtainable from the supplied ReadMe file with the RTC4/RTC3/RTC2 and needed to be manually incorporated into the user program. For the RTC5, the parameters included in the correction files can be read out by **get_table_para** (from the currently loaded correction tables) or by **get_head_para** (from the assigned correction tables) and thus directly incorporated into a user program. The file header contains the following parameters (numbering corresponds with that of the commands **get_table_para** and **get_head_para**):

(0) Type of correction tables

- = 0.0: 2D correction table
- = 1.0: 3D correction table

(1) Calibration factor K_{xy} [bit/mm] (see [page 134](#))

This parameter is generally a integer-multiple of 16. Because of the RTC5's 16x higher resolution, a ct5 file's calibration factor is 16 times larger than that of a corresponding ctb correction file.

(2) Focal Length / Working Distance [mm]

- for a configuration with a scan objective, this is the objective's effective focal length [mm]
- for a configuration without a scan objective, this is the working distance A [mm] (distance from the optical axis of the incident laser beam at the first deflection mirror to the image plane).

(3) Stretch factor for the X direction

for 3D vector commands this results in a Z-coordinate-dependent modification of the image field

(4) Stretch factor for the Y direction

See parameter 3.

(5) Coefficient A of the parabolic function for Z-axis control (offset part, ± 26 Bit)

(6) Coefficient B of the parabolic function for Z-axis control (linear part, ± 11 Bit)

(7) Coefficient C of the parabolic function for Z-axis control (square part, ± 4 Bit)

(8) Number of the correction file

i.e. the filename number of the correction file supplied by SCANLAB (e.g. 145 for D2_145.ct5 or D3_145.ct5).

(9) Differentiation between correction with or without an F-Theta objective

The following applies:

Parameter = $10 * P_{Obj} + P_{Typ}$ with

- $P_{Obj} = 0$: Correction without F-Theta objective
- $P_{Obj} = 1$: Correction with F-Theta objective
- if correction with F-Theta objective:
 - $P_{Typ} = 0.0$: without distortion data
 - $P_{Typ} = 1.0$: with F-Theta's F-stop progression condition
 - $P_{Typ} = 2.0$: with image height table

(10) Indicator for the source of the correction file

The following applies:

Parameter = $1000 * P_{Orig} + P_{Ver}$ with

- $P_{Orig} = 10000$: Original file
(as originally calculated)
- $P_{Orig} = 20000$: converted from ctb file
- $P_{Orig} = 30000$: reconstructed from txt file
- If a correction file is modified by the correXion5 software or by another correction program, then P_{Orig} is incremented by 1 each time
- P_{Ver} = Version number of the program used to create the correction file

(11) Information about inverse tables.

The following applies:

Parameter = $P_{Exist} + 2 * P_{Calc}$ with

- $P_{Exist} = 1.0$: valid inverse table is present
- $P_{Exist} = 0.0$: no valid inverse table present
- if valid inverse table is present:
 - $P_{Calc} = 0$: inverse table calculated ab initio
 - $P_{Calc} = 1$: inverse table numerically calculated

(12) Scan system's angle calibration

Mechanical angle deflection in [$\pm ^\circ$] for 96% of maximum deflection

(13) Indicator for the scan head geometry used for the calculation (for internal use only), e.g.:

- = -1.0: unknown geometry (e.g. for a table converted from a ctb file)
- = 0.0: standard geometry

(14) Indicator for whether an additional protective glass was taken into account

The following applies:

Parameter = $1000000 * P_T + 1000 * P_I$ with

- P_T = protective glass thickness in mm (max. 2 decimal places)
- P_I = refraction index (max. 3 decimal places)

Example: The value 3521450.0 corresponds to a protective glass thickness of 3.52 mm and a refraction index of 1.450.

- (15) Indicator for whether the image field size was clipped in the correction file
 – = 0.0: field size not clipped
 – = 2.0: field size clipped

Notes

- A ct5 file created by converting another ctb file does not contain all parameter data (see below). The same applies to a 1to1 correction table.
- Parameters 3-7 are only relevant for 3D vector commands and are set to 0 in 2D correction tables.
- If the correct calibration factors are used (see [page 194](#)), then the same ABC coefficients (parameters 5-7) can be used on the same 3-axis scan system with RTC4 Boards (ctb files) and RTC5 Boards (ct5 files).

Converting Correction Files

The delivered product includes the program `CorrectionFileConverter.exe` for converting RTC4 correction files (ctb) into RTC5 correction files (ct5) and vice versa. Program usage is described in the accompanying guide.

When converting a ctb file into a ct5 file, the latter's file header receives an origin indicator (parameter 10, $P_{\text{Orig}} = 20000$). Such conversions do not produce fully complete RTC5 correction files. After conversion, the file header lacks a variety of parameters, which can be subsequently added by the `CorrectionFileConverter.exe` program. Moreover, conversion of numerically calculated inverted correction tables sometimes results in incomplete scan field coverage or even no inverse correction table at all (see also parameter 11, [page 140](#)).

In contrast, converting a ct5 file into a ctb file results in a fully complete ctb correction file. The ctb file's 2D tables then do not contain the parameters from the ct5 file header, whereas the 3D tables contain only some of them.

7.3.5 Output Values to the Scan System

Calculations

X and Y coordinate values specified in vector and arc commands are converted by the RTC5 (while taking into account the Z axis coordinates if necessary) into output values for the scan system's galvanometer scanners. Specified Z coordinate values are (while taking into account the X and Y coordinate values) initially converted into a focal length value and finally into an output value for the scan system's Z axis.

During calculation of output values based on the specified coordinate values, the following corrections (in the listed order, if previously configured) are taken into account after successful microvectorization:

- If a wobbel motion was enabled by `set_wobbel` or `set_wobbel_mode`, then the X and Y coordinates are transformed in accordance with the current wobbel vector (see [page 189](#))⁽¹⁾.
- If a Processing-on-the-fly correction was enabled by `set_fly_x`, `set_fly_y`, `set_fly_rot` or `set_fly_x_pos`, `set_fly_y_pos`, `set_fly_rot_pos`, then it is applied to the X and Y coordinates (see [page 199](#))⁽²⁾.
- If a coordinate transformation was defined for aligning the scan system to the image field (see `set_matrix`, `set_offset`, `set_scale`, `set_angle` and corresponding list commands), then it is applied to the X and Y coordinates (see [page 183](#)).
- Additionally, `set_offset_xyz` or `set_offset_xyz_list` as well as `set_defocus` or `set_defocus_list` can be used for defining an offset to the Z coordinate or the calculated focal length, which then appropriately affect the Z output value (see [page 193](#)).
- Coordinate values exceeding the real image field range (into the virtual image field range, see [page 135](#)) is clipped to the edge of the 20-bit real image field range.

(1) only for marking commands (mark and arc), not for jump commands or `goto_xy` and `goto_xyz`

(2) only for marking commands (mark and arc) and jump commands, not for `goto_xy` and `goto_xyz`

- Correction tables assigned by `select_cor_table` or `select_cor_table_list` result in 2D or 3D image field correction (see [page 136](#)).
- If automatic self-calibration was enabled by `auto_cal` (see [page 224](#)), then a compensating "Gain" and "Offset" is taken into account when calculating output values (gain and offset correction can also be specified by `set_hi`, see [page 227](#)).

Overflowing output values are, if necessary, clipped to the edge of the maximum possible range of values.

Value Ranges

For all axes of the scan system, the output values are (unlike with the RTC4) 20-bit signed numbers (i.e. digital values between –524288 and +524287).

Therefore, the RTC5 converts (by multiplying by 16) image field coordinates for the Z axis specified as 16-bit signed numbers (between –32768 and +32767) and image field coordinates for the X and Y axes specified as 16-bit signed numbers in RTC4 compatibility mode into 20-bit output values. However – even in RTC4 compatibility mode – the RTC5 does *not* change 20-bit values returned by the scan system (directly or by the XY2-100 converter).

If the scan system is controlled in conjunction with an XY2-100 converter, then the converter converts (by eliminating the four least significant bits) the RTC5's 20-bit output values into 16-bit unsigned numbers (i.e. into digital values between 0 and 65535) for all axes of the scan system (such control values are outputted by the RTC4). 16-bit values returned by the scan system are converted into 20-bit values (by extension with four null bits).

Precalculation and Diagnosis

By using the command `get_galvo_controls` the output values for specified coordinates and setting parameters in the present configuration (correction file, coordinate transformations) can be precalculated without having a galvanometer scanner movement.

Clock Overruns

The 10 µs clock period might not always suffice for calculating all data required by the computation-intensive jump, mark and arc commands if several of the available command options are utilized simultaneously – e.g. simultaneous control of two scan systems, wobble motion, Processing-on-the-fly for two axes (correction by McBSP/SPI interface), automatic laser control, para vectors, data recording, variable polygon delay (scanner delay possibly 0), short list commands (e.g. for polylines), certain control commands during list execution.

This overrun situation can be internally detected and counted. You can appropriately test your user program by using `get_overrun` to count the number of overruns. Such overruns result in one or several peripheral ports not being accessible during the current 10 µs clock period, possibly including output to the scan head (where galvanometer scanner motion could pause for 10 µs).

7.3.6 Status Monitoring and Diagnostics

For status monitoring and diagnostic purposes, the commands **get_value** or **get_values** can be used to read a variety of signals:

- a 20-bit status word returned by the scan system
 - by the XY2-100 converter when applicable ("Status Information Returned from the Scan System", page 143)
- the current value of the LASERON signal
- the current cartesian output values (hence the sample values common to both scan head connectors: any wobble or Processing-on-the-fly corrections are taken into account, but not image field corrections or coordinate transformations)
- the (specific to each scan head connector) control values which take into account
 - any coordinate transformations defined by **set_matrix**, **set_offset**, **set_scale** or **set_angle** (or by the corresponding list commands),
 - any Z coordinate offset defined by **set_offset_xyz** or **set_offset_xyz_list**,
 - any focal length offset defined by **set_defocus** or **set_defocus_list** and
 - any loaded correction table.

The command **set_trigger/set_trigger4** can be used on each of these signals to record their values over time – and with a selectable sample period. The recorded values are stored on the RTC5. The command **get_waveform** can be used to transfer the recorded values to the PC.

The current status of a measurement session started with **set_trigger/set_trigger4** can be obtained by calling the command **measurement_status**.

The returned values are always 20 bits wide. When necessary, even in RTC4 compatibility mode, these must be converted by users into 16-bit values (by dividing by 16). The XY2-100 status word, too, is returned by **get_value**, **get_values** or **set_trigger/set_trigger4** as a 20-bit value (but as a 16-bit value by **get_head_status**, see below).

Status Information Returned from the Scan System

The scan system returns the following signals (on the return channel) to the RTC5 every 10 µs by the SL2-100 protocol:

- a 20-bit status word:
 - Most scan systems return the XY2-100 status word⁽¹⁾. iDRIVE scan systems (intelliSCAN, intelliSCAN_{de}, intelliDRILL, intellicube, intelliWELD, varioSCAN_{de}) also allow other data signals for each axis to be transmitted separately to the RTC5 for evaluation instead (see chapter 8.1, page 172).
 - Scan systems without SL2-100 interface return a 16-bit status word. This value is multiplied by 16 and transferred to the RTC5 as 20-bit value by the XY2-100 converter.
 - The 20-bit status word can be queried by **get_value**, **get_values** or **set_trigger/set_trigger4** (see below).
- 6 additional status bits:
 - Scan systems with SL2-100 interfaces – additionally to the 20-bit status word – transmit the 6 variable bits of the XY2-100 status word.
 - For scan systems without SL2-100 interfaces the XY2-100 converter – additionally to the 20-bit status word – transmits to the RTC5 bits #6-11 of the 16-bit status word outputted by the scan system. If the scan system outputs the XY2-100 status word (the iDRIVE scan systems should be correspondingly configured), then the 6 variable bits of the XY2-100 status word is transmitted for such systems, too.
 - The 6 additional status bits are queried by **get_head_status** and returned as a (16-bit) XY2-100 status word (see command description).
- a user data bit
 - By this user bit, iDRIVE scan systems with SL2-100 interfaces (operated without an XY2-100 converter) can supplementally return slowly changing status information (e.g. current galvanometer temperature) along with the above-mentioned "normal" status return. This status information can be queried by **read_user_data** (see also page 173).

(1) The status signals of the XY2-100 status word can also be used for laser-signal auto-suppression (see page 146).



7.4 Laser Control

The RTC5 provides several laser control signals with programmable pulse length and frequency by the LASERON, LASER1 and LASER2 ports of the 15-pin D-SUB-laser connector (see [page 47](#)) or of the EXTENSION 2 socket connector (see [page 52](#)). These signals can be used for controlling various types of lasers.

By the command `set_laser_mode`, users can choose one of seven different laser control modes:

- The CO₂ mode (laser mode 0) is provided for controlling CO₂ lasers.
- The four YAG modes (laser modes 1, 2, 3 and 5) are provided for controlling Nd:YAG and related lasers.
- Laser mode 4 and laser mode 6 generate a continuously-running control signal.

Depending on the selected mode, different laser control signals are available for "laser active" operation and "laser standby" operation. The various laser modes are described in the following sections.

All laser control signals are TTL signals. They can be set to either active-high or active-low by the command `set_laser_control` – see also [page 47](#). The command `get_startstop_info` (Bit#13) can be used to query whether the signals are currently set to active-high or active-low.

The maximum current load for all signals is 20 mA.

7.4.1 Enabling, Activating and Switching Laser Control Signals

After a hardware reset and also after initialization with `load_program_file`, all laser control signals are disabled. All laser output ports (LASERON, LASER1 and LASER2) are in the (high impedance) tristate mode. TTL states (LOW or HIGH) are only available, if the signals are enabled by the `set_laser_control` command (and the desired TTL level is thereby specified).

After `set_laser_control` is called:

- Laser control signals for "laser active" operation are provided at all output ports as soon as these signals have been activated and switched on or
- All output ports provide standby signals, if they have been activated and if no laser control signals for "laser active" operation are switched on or
- The outputs are in their basic TTL states (e.g. LOW if a high signal level was specified) if no laser control signals (neither for "laser active" nor for "laser standby" operation) have been activated

After `set_laser_control` has been called, the laser output ports (LASERON, LASER1 and LASER2) enter the (high impedance) tristate mode only after a new hardware reset or if the laser control signals are subsequently disabled by a command (see `disable_laser`, `pause_list` and `set_laser_control`(bit#2 = 1)).

Before activation of the laser control signals, the command `set_laser_mode` should be called for selecting the laser control mode (otherwise – by default – the CO₂ mode is selected).

For first-time activation of the laser control signals (both for "laser active" and for "laser standby" operation), the `set_laser_control` command must be used to enable the laser output ports (LASERON, LASER1 and LASER2).

"Laser Active" Signals

The signals for "laser active" operation – after `set_laser_control` – can be activated by `set_laser_pulses`, `set_laser_pulses_ctrl` or `set_laser_timing`. Additionally they can be enabled by `set_laser_control` or `enable_laser` and disabled by `set_laser_control` or `disable_laser`⁽¹⁾. The command `get_startstop_info` (Bit #9) queries the current status of the laser control signals (enabled or disabled).

Even if the laser control signals for "laser active" operation have been enabled and activated as described above, they only appear at the output ports if they are *switched on* by further commands. They are (automatically) switched on when mark or arc commands are executed (see [page 103](#)); they can also be switched on for a fixed scanner position by the command `laser_on_list` for a specified time interval⁽²⁾ or switched on (for an indefinite period of time) by `laser_signal_on` or `laser_signal_on_list` and switched off by `laser_signal_off` or `laser_signal_off_list`.

The laser control signals for "laser active" operation are automatically switched off by the RTC5, when a marking command (mark or arc) is followed by a non-marking command, when a list is stopped (by `set_end_of_list` or `stop_execution`) or a list is temporarily interrupted (after `set_wait`, `pause_list` or `stop_list`). In the latter case, the signals are switched on again, when the list is continued (after `release_wait` or `restart_list`). The laser control signals are also automatically switched off by a `list_nop` command. This command is automatically inserted by the RTC5 under some circumstances, e.g. as a replacement for list commands with an invalid parameter.

"Laser Standby" Signals

After first-time activation of the laser control signals by `set_laser_control`, standby signals can be activated and deactivated by `set_standby` or `set_standby_list`.

Standby signals are permanently outputted after activation without further commands if and only if no signals for "laser active" operation are switched on. If applicable, the commands which switch on and switch off the signals for "laser active" operation (see previous section), correspondingly switch from standby to "laser active" signals (or vice versa). If the "laser active" signals are deactivated (e.g. by `PulseLength = 0`), then no switching occurs and the standby signals are permanently outputted even during execution of such commands.

If the standby signals are deactivated, then for "laser standby" operation all outputs are set to their basic TTL states (e.g. LOW if a high signal level was specified by `set_laser_control`).

If activated standby signals shall be deactivated when a list is halted by `pause_list` (here the RTC5 automatically switches off only the laser control signals for "laser active" operation), then this must be explicitly induced by calling the command

`set_standby`(`PulseLength = 0`). If you do not know the currently set standby parameters at the moment you call the control command `pause_list` command (the standby parameters could have been changed within a list) these can be read out by the control command `get_standby`.

- (1) If the signals are disabled, the laser output ports (LASERON, LASER1 and LASER2) are in the (high impedance) tristate mode (the signals are automatically disabled by `pause_list`, too, and reenabled by `restart_list`).
- (2) By `laser_on_pulses_list`, you can also switch on the LASERON signal for a specified number of external signal pulses.



Laser-Signal Auto-Suppression Triggered by Scan-System-Errors

set_laser_control (bits#16 – 27) can be used to define that the laser control signals shall be automatically suppressed when one or more scan-system status signals indicate an error. It can also be used to select, *which* status signals to be used for this laser-signal auto-suppression: PowerOK, TempOK and/or PosAck status signals of heads A and/or B and axes X and/or Y.

If in case of an error at least one of the selected status signals is 0, then

- output of the laser control signals (enabled by **set_laser_control** or **enable_laser**) are automatically interrupted and are only continued if all selected status signals are simultaneously 1 (laser control signals disabled by **set_laser_control**, **disable_laser** or **pause_list** remain disabled regardless of the status signals' current value),
- internal error bits are (cumulatively) set, which can be read-out by **get_marking_info** (as of version DLL 527, OUT 529).
- if accordingly set by **set_laser_control**(bit#28 = 1), a /STOP signal is automatically generated (list stops, laser control signals get permanently switched off).

Laser-Signal Auto-Suppression Triggered by Galvanometer Scanner Position Exceedances

range_checking can be used to define that the laser control signals shall be automatically suppressed when a galvanometer scanner exceeds a given position limit.

7.4.2 Configuring the Laser Connector (LASER)

To configure the laser signal types to be put out on pins 1, 2 and 9 at the LASER connector, you can use the control command **config_laser_signals** (as of version DLL 535, OUT 535, RBF 524) and the list command **config_laser_signals_list** (as of version DLL 537, OUT 537).

If you employ a variety of lasers or laser operational modes, then these commands might eliminate the need to configure at the hardware level (i.e. using various cables and switches).

Whereas the default setting (for normal vector marking) outputs the LaserOn signal as a laser start signal on the LASERON channel, you could, for example, configure the LASER2 signal as a gate signal outputted on the LASERON channel in pixel mode with pulse picking.

For other operational modes, you could also, for example, configure the FirstPulseKiller signal as the laser start signal on the LASERON channel. This way, LASER1 signals can be outputted even before a delayed switch-on of the laser (which is not possible in the default setting, see **config_laser_signals/config_laser_signals_list** command description).

The following description of the various laser modes applies to the default setting for laser signal output.

7.4.3 CO₂ Mode

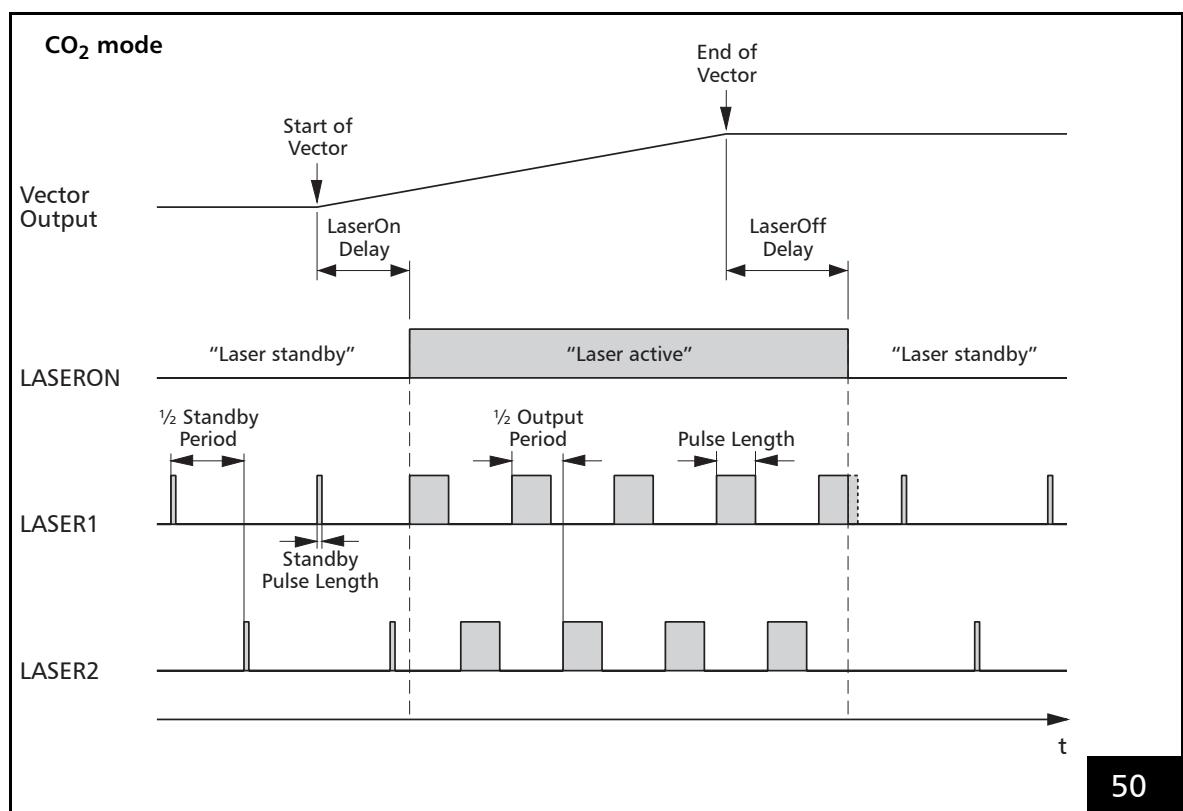
The command `set_laser_mode(0)` selects the CO₂ laser mode (laser mode 0). The laser control timing diagram [figure 50](#) shows the corresponding signals.

For “laser active” operation (if applicable after a LaserOn delay and if enabled by `set_laser_control` or `enable_laser`)

- The LASERON signal is switched on and
- Two alternating modulation signals are provided at the LASER1 and LASER2 outputs with pulse lengths and periods that can be defined by `set_laser_pulses`, `set_laser_pulses_ctrl` or `set_laser_timing`.

For “laser standby” operation (if applicable after a LaserOff delay)

- The LASERON signal is switched off and
- Alternating standby pulses are provided at the LASER1 and LASER2 outputs with pulse lengths and periods that can be defined by `set_standby` or `set_standby_list`.



Laser control timing diagram for CO₂ mode (with active-high laser control signals)



Notes

- The LASER1 and LASER2 signals are activated by setting the pulse length to a nonzero value and are deactivated by setting the pulse length to zero (default setting).
- The signals LASER1 and LASER2 have a constant relative phase shift of 180° (half an output period). LASER2 can be used for the control of a second laser tube. With `set_laser_control` (bit#1=1), both signals can be exchanged with each other. To control laser power, the pulse length of the LASER1 and LASER2 signals can be varied. Both signals share the same pulse lengths and output frequencies.
- Note that *half* of the output period must be specified for the LASER1 and LASER2 signals, i.e. the shift between the two laser signals.
- The commands `set_laser_pulses` and `set_laser_timing` are *list* commands. They can be used within a list, for instance, for changing the laser power at any time between two list commands.
- The actual output period and the pulse lengths of laser signals LASER1 and LASER2 (but not the laser delays) are dependent on the time base. For RTC5 mode, this is fixed at 64 MHz (1 bit equals 1/64 µs). For RTC4 compatibility mode (for “laser active” operation), the time base can be set by `set_laser_timing` (but not by `set_laser_pulses` or `set_laser_pulses_ctrl`) for either 1 MHz (1 bit equals 1 µs) or 8 MHz (1 bit equals 1/8 µs). For RTC4 compatibility mode, SCANLAB generally recommends setting the time base to 8 MHz (by `set_laser_timing(...,...,...,1)`). A time base of 1 MHz should only be chosen if necessary. The time base for standby signals is always 1/64 µs (in RTC5 mode) or 1/8 µs (in RTC4 compatibility mode).
- The command `set_laser_control` (bit#0) can be used to specify whether a (for “laser active” operation) started modulation pulse (LASER1 or LASER2) should execute to completion or be cut off when it has not yet fully executed when the LASERON signal is switched off (see figure 50). This cannot be specified for standby signals; if applicable, these are cut off.

7.4.4 YAG Modes

With `set_laser_mode`([1, 2, 3 or 5]) one can choose between four different YAG laser control modes. The laser control timing diagram [figure 51](#) shows the corresponding signals.

In all four YAG modes, for “laser active” operation (if applicable after a LaserOn delay and if enabled by `set_laser_control` or `enable_laser`)

- The LASERON signal is switched on,
- A Q-Switch signal with variable pulse length and frequency is provided at the LASER1 output and
- A programmable FirstPulseKiller signal is provided at the LASER2 output.

For “laser standby” operation (if applicable after a LaserOff delay)

- The LASERON signal is switched off and
- Standby pulses are provided at the LASER1 output (but not at the LASER2 output) with pulse lengths and periods that can be defined by `set_standby` or `set_standby_list`.

The LASER1 signal is activated by setting the pulse length to a nonzero value and is deactivated by setting the pulse length to zero (default setting).

Q-Switch Signal

The Q-Switch signal is available for control of the laser’s Q switch. The Q-Switch period and pulse length are set with the commands `set_laser_pulses`, `set_laser_pulses_ctrl` or `set_laser_timing`.

Notes

- Note that *half* of the Q-Switch output period must be specified.
- The actual Q-Switch output period and the pulse length (but not the laser delays) are dependent on the time base.
For RTC5 mode, this is fixed at 64 MHz (1 bit equals 1/64 µs).
For RTC4 compatibility mode, the time base can be set by `set_laser_timing` (but not by `set_laser_pulses` or `set_laser_pulses_ctrl`) for either 1 MHz (1 bit equals 1 µs) or 8 MHz (1 bit equals 1/8 µs). For RTC4 compatibility mode, SCANLAB generally recommends setting the time base to 8 MHz (by `set_laser_timing(.....,1)`). A time base of 1 MHz should only be chosen if necessary.
- The command `set_laser_control` (Bit #0) can be used to specify whether a Q-Switch pulse (LASER1) should execute to completion or be cut off when it has not yet fully executed when the LASERON signal is switched off (see [figure 51](#)). This cannot be specified for standby signals; if applicable, these are cut off.

FirstPulseKiller Signal

The FirstPulseKiller signal is available for reduction of the laser pulse power at the beginning of a pulse train. The initial pulses of a pulse train often have a higher energy than the following pulses. These intensity variations are to be avoided in most laser marking applications. Therefore the laser should be equipped with a FirstPulseKiller device that reduces the power of the first laser pulses.

The RTC5 provides a control signal for the FirstPulseKiller (TTL level). The FirstPulseKiller signal is started together with the LASERON signal. While the FirstPulseKiller signal is active, the energy of the pulses should be reduced adequately.

The length of the FirstPulseKiller signal is set with the command `set_firstpulse_killer` or `set_firstpulse_killer_list`.

Differences Between the YAG Modes

The four YAG laser modes only differ in the relative start time of the first Q-Switch pulse with reference to the FirstPulseKiller signal (see also the timing diagrams in figure 51).

- In **YAG mode 1** the first Q-Switch pulse starts at the *beginning* of the FirstPulseKiller signal.
- In **YAG mode 2** the first Q-Switch pulse starts at the *end* of the FirstPulseKiller signal. This mode is provided for certain YAG laser types.
- In **YAG mode 3** the first Q-Switch pulse starts 10 µs after the beginning of the FirstPulseKiller signal.
- In **YAG mode 5** the time interval can be specified by `set_qswitch_delay` or `set_qswitch_delay_list`.
YAG modes 1-3 are special cases of YAG mode 5 with:
 - Q-Switch delay = 0 (YAG mode 1)
 - Q-Switch delay = Length of the FirstPulseKiller signal (YAG mode 2)
 - Q-Switch delay = 10 µs (YAG mode 3)

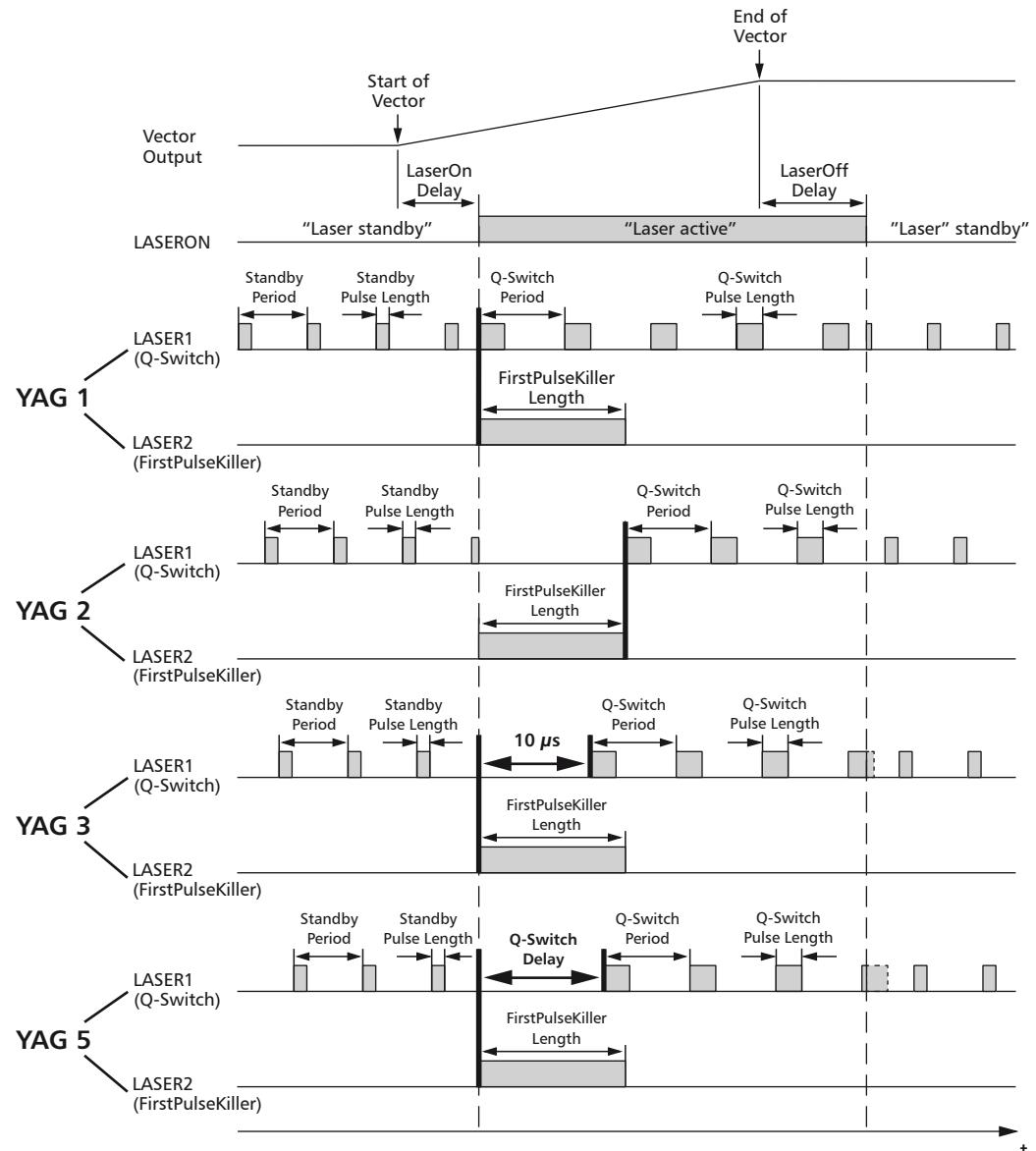
The `set_laser_mode` command, however, merely preselects the Q-Switch delays. In YAG modes 1-3, too, the Q-Switch delay can therefore be subsequently changed by `set_qswitch_delay` or `set_qswitch_delay_list`. For YAG mode 2, each `set_firstpulse_killer` or `set_firstpulse_killer_list` command results in a corresponding adjustment of the Q-Switch delay, too.

Lamp Current (Laser Power)

To control the lamp current of a YAG laser, the 12-bit analog output ports, ANALOG OUT1 or ANALOG OUT2 can be used. They are available by the 15-pin D-SUB laser connector – see figure 10. To set the output signal, use the control command `write_da_x` or the short list command `write_da_x_list`.

Alternatively, the lamp current can be controlled digitally by the 8-bit digital output port on the EXTENSION 2 socket connector (figure 17). The commands for setting the 8-bit port are `write_8bit_port` and `write_8bit_port_list` (see also chapter 4.4.3 "EXTENSION 2 Socket Connector", page 52).

When the lamp current is changed, continued execution of the list can, if required, be halted by the `long_delay` command until a stable laser power level has been achieved.

YAG modes 1-3 and 5


Laser control timing diagram for YAG modes 1, 2, 3 and 5 (with active-high laser control signals)
 (the standby signals are not synchronized with the laser-active signals and can have arbitrary phase alignments)

7.4.5 Softstart Mode

For some applications it is important to control the laser intensity at the beginning of a marking process. SCANLAB provides a convenient solution in the form of the softstart mode, which can be used for all laser modes (for "laser active" operation).

In the softstart mode, various control values (`Level(0)` ... `Level(Number - 1)`, `Number ≤ 1024`) for as many as the first 1024 pulses (at the beginning of a marking process) can be defined. Either pulse length values for the laser signal at the LASER1 output (see [figure 10](#)) or analog voltage levels for variable laser power can be defined. Analog voltage softstart values can be outputted either at the ANALOG OUT1 or ANALOG OUT2 output ports (see [figure 10](#)).

Initialization of the softstart mode is performed by `set_softstart_mode` or `set_softstart_mode_list`. Here, the type of softstart value is defined. If analog voltage values are to be outputted, then the analog output port can also be defined. Afterwards the individual softstart control values can be defined by `set_softstart_level` or `set_softstart_level_list` commands.

After the laser control signals are switched on, the defined softstart values (max. 1024) are outputted at the selected output port simultaneously with the laser pulses of the LASER1 signal – always with the leading edge of the laser pulse. The `set_softstart_mode`/`set_softstart_mode_list` command allows specification of a time period (`Delay`) for which the laser must have been switched off before softstart is activated at the next switch-on.

Notes

- With a large enough value for the `Delay` time, one can avoid the softstart mode being also activated after very short jump commands.
- As soon as the LASERON signal (the laser) remains switched off longer than `Delay`, the value `Level(0)` is assigned to the output port, provided the softstart mode has not been meanwhile deactivated.
- When the LASERON signal is switched on, then the values `Level(0)` ... (`Number - 1`) are automatically assigned to the output port simultaneously with the first laser pulses.
- Analog softstart values are outputted simultaneously with the laser pulses of the LASER1 signal. It might be beneficial under some circumstances to acquire these analog softstart values triggered by a signal shifted back 180° related to the LASER1 signal. For this purpose, the CO₂ mode provides a LASER2 signal at the LASER2 output. And in the YAG version's, the signal at the LASER1 output can be phase-shifted back 180° by `set_laser_control` (Bit #1 = 1). For the CO₂ mode, this is equivalent to exchanging LASER1 with LASER2.
- The laser pulse frequency should not exceed a value of approx. 308 kHz (this corresponds to a `set_laser_pulses-HalfPeriod` of 104), because the changing values cannot be transmitted faster. However, the laser pulse frequency cannot be automatically checked for this case.
- If Softstart values are to be outputted as analog voltage levels, then also note that digital-to-analog conversion of laser frequencies above around 100 kHz (i.e. for a `set_laser_pulses-HalfPeriod` < approx. 320) cannot always be fully completed. For such laser frequencies, users must carefully verify that sufficient capability is available.
- If Softstart mode is inactive, then values for the analog outputs can be set by `write_da_x` or `write_da_x_list`.



- The Softstart mode is also available in Laser Mode 4 (see [page 154](#)) and Laser Mode 6 (see [page 155](#)). In these modes, the LASER1 output only outputs standby pulses and therefore only analog voltage levels can be variably defined ([`set_softstart_mode`](#) Mode = 1, 2, 11 or 12). Here, too, the defined softstart values are outputted simultaneously with the pulses of an internally-generated (but not outputted) LASER1 signal. The standby pulses cannot be phase-shifted in laser mode 4 and laser mode 6 and are not synchronized with the internal LASER1 pulses. If the period durations of the standby and LASER1 pulses are set (by [`set_laser_pulses`](#) and [`set_standby`](#)) to be equal, then the analog voltage softstart values are outputted in parallel with the standby pulses, though they might have a (random) constant phase shift with respect to the standby pulses.
- The Softstart mode cannot be used simultaneously with the “freely definable wobble shapes” wobble mode, see [`set_wobble_vector`](#).

7.4.6 Laser Mode 4

The command `set_laser_mode(4)` selects the laser mode 4. The laser control timing diagram [figure 52](#) shows the corresponding signals.

In this mode, a continuously-running modulation signal with variable pulse length and frequency is provided at the LASER1 output both for "laser active" and for "laser standby" operation, if it is activated by `set_standby` or `set_standby_list`.

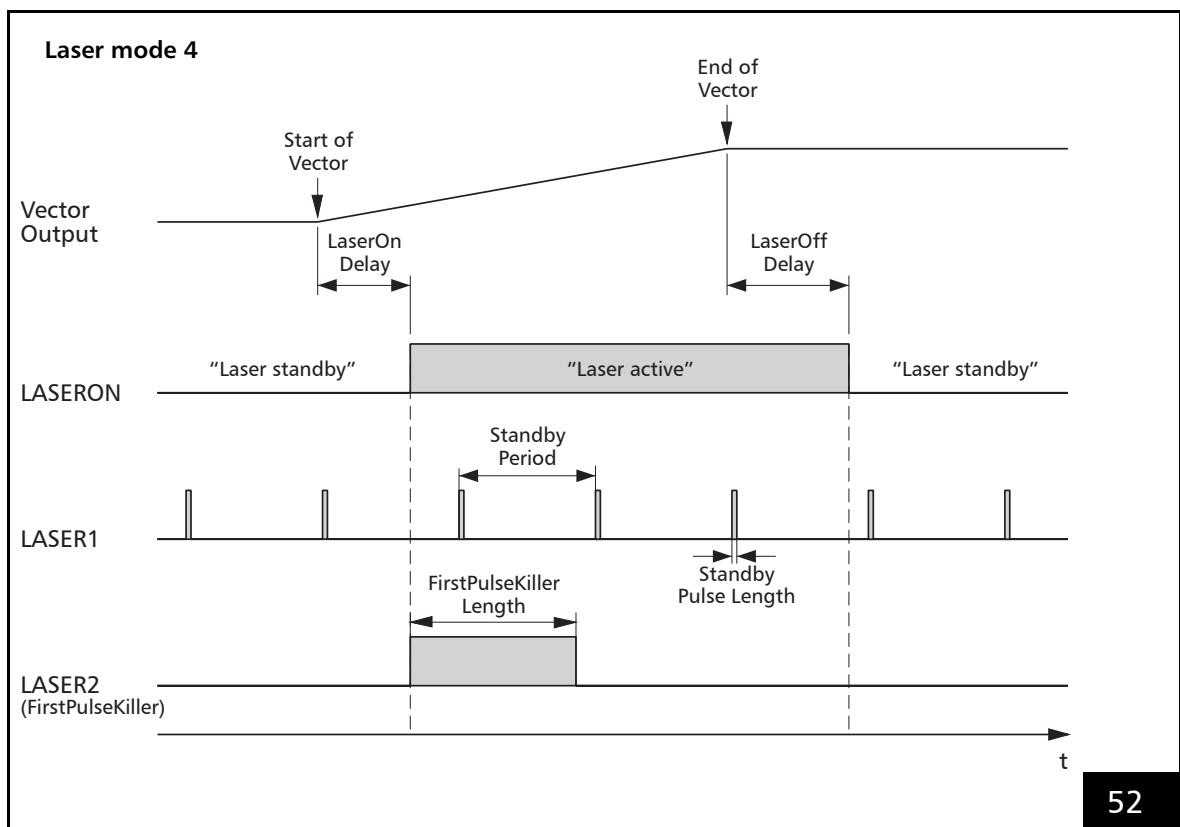
Additionally, for "laser active" operation (if applicable after a LaserOn delay and if enabled by `set_laser_control` or `enable_laser`)

- The LASERON signal is switched on and
- A programmable FirstPulseKiller signal is provided at the LASER2 output.

For "laser standby" operation, additionally the LASERON signal and (if applicable) the LASER2 signal are switched off (if applicable after a LaserOff delay).

Notes

- The pulse length and the output period of the LASER1 modulation signal are set with the command `set_standby` or `set_standby_list`. The LASER1 signal is activated by setting the pulse length to a nonzero value and is deactivated by setting the pulse length to zero (default setting).
- Note that *half* of the output period must be specified.
- The FirstPulseKiller signal is started together with the LASERON signal. The length of the FirstPulseKiller signal is set with the command `set_firstpulse_killer` or `set_firstpulse_killer_list`.
- In laser mode 4 the time base for the signals LASER1 and LASER2 is always 1/64 µs (in RTC5 mode) or 1/8 µs (in RTC4 compatibility mode).
- Laser mode 4 is used for some fiber lasers.



Laser control timing diagram for laser mode 4 (with active-high laser control signals)

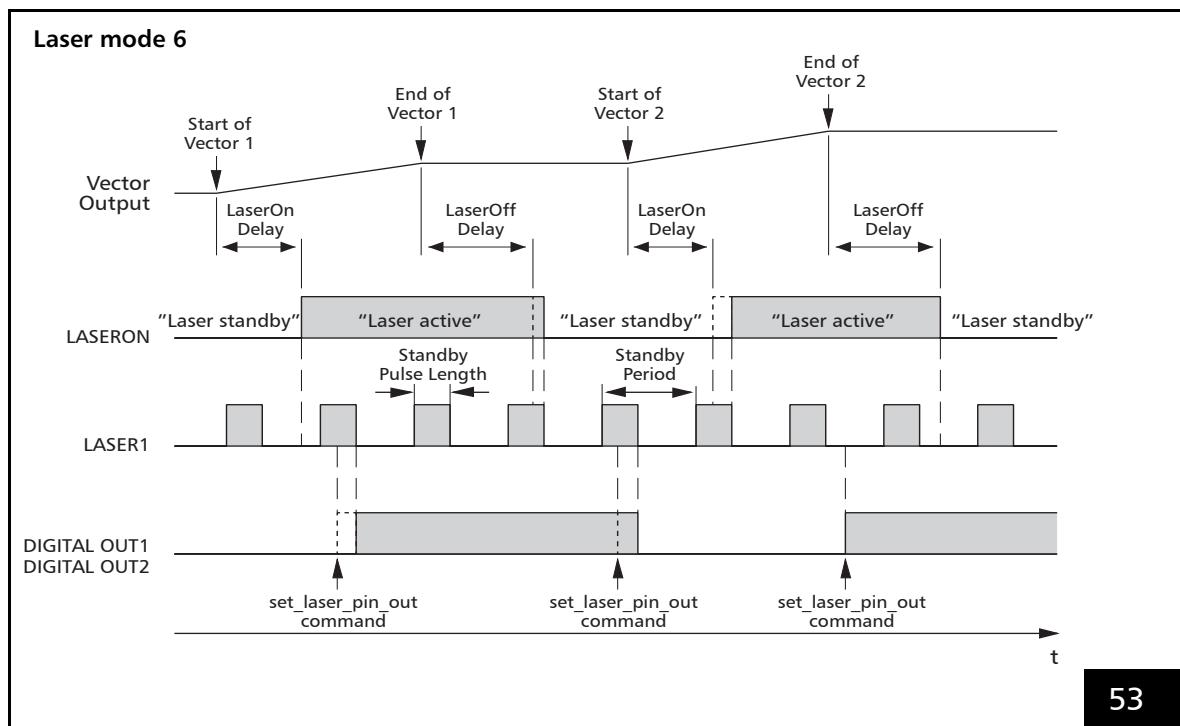
7.4.7 Laser Mode 6

The command `set_laser_mode(6)` selects the laser mode 6. This laser mode is provided for lasers, which do not allow changing the laser power during execution of a laser pulse. The laser control timing diagram [figure 53](#) shows the corresponding signals.

In this mode, a continuously-running modulation signal with variable pulse length and frequency is provided at the LASER1 output both for "laser active" and for "laser standby" operation, if it is activated by `set_standby` or `set_standby_list`.

Additionally, for "laser active" operation the LASERON signal is switched on (if applicable after a LaserOn delay and if enabled by `set_laser_control` or `enable_laser`) synchronized with the LASER1 signal (see below).

For "laser standby" operation, the LASERON signal is switched off (if applicable after a LaserOff delay) synchronized with the LASER1 signal (see below).



Laser control timing diagram for laser mode 6 (with active-high laser control signals)



Notes

- Here a LASER2 signal is neither provided for “laser active” nor for “laser standby” operation.
- Any change of the LASERON signal is prohibited while the LASER1 signal is “on”: the change is executed (with delay) only when the LASER1 signal is “off” again (see [figure 53](#)). For instance – if the LASER1 signal is set active-high – that means: a change of the LASERON signal is not executed while the LASER1 signal is high but is executed (with delay) when the LASER1 signal is low again.
Likewise, in laser mode 6 any change of the output value at the 2-bit digital output (DIGITAL OUT1 and DIGITAL OUT2) by [**set_laser_pin_out**](#) or [**set_laser_pin_out_list**](#) is executed with delay, if the LASER1 signal is “on” at the moment of change. This allows controlling external equipment synchronously with the laser pulses.
- The pulse length and the output period of the LASER1 modulation signal are set with the command [**set_standby**](#) or [**set_standby_list**](#). The LASER1 signal is activated by setting the pulse length to a nonzero value and is deactivated by setting the pulse length to zero (default setting).
- Note that *half* of the output period must be specified.
- In laser mode 6 the time base for the LASER1 signal is always 1/64 µs (in RTC5 mode) or 1/8 µs (in RTC4 compatibility mode).
- To mark points, you can use [**laser_on_pulses_list**](#) to switch on the LASERON signal for a specified number of external pulses (or a specified time interval). The external pulses must be supplied at the LASER connector’s DIGITAL IN1 digital input (see [page 48](#)). With [**para_laser_on_pulses_list**](#), a selected signal parameter is linearly varied to the supplied value during point marking.

7.4.8 Pulse Picking Laser Mode

The commands `set_pulse_picking` or `set_pulse_picking_list` select the pulse picking laser mode. The laser control timing diagram in figure 54 shows the corresponding signals.

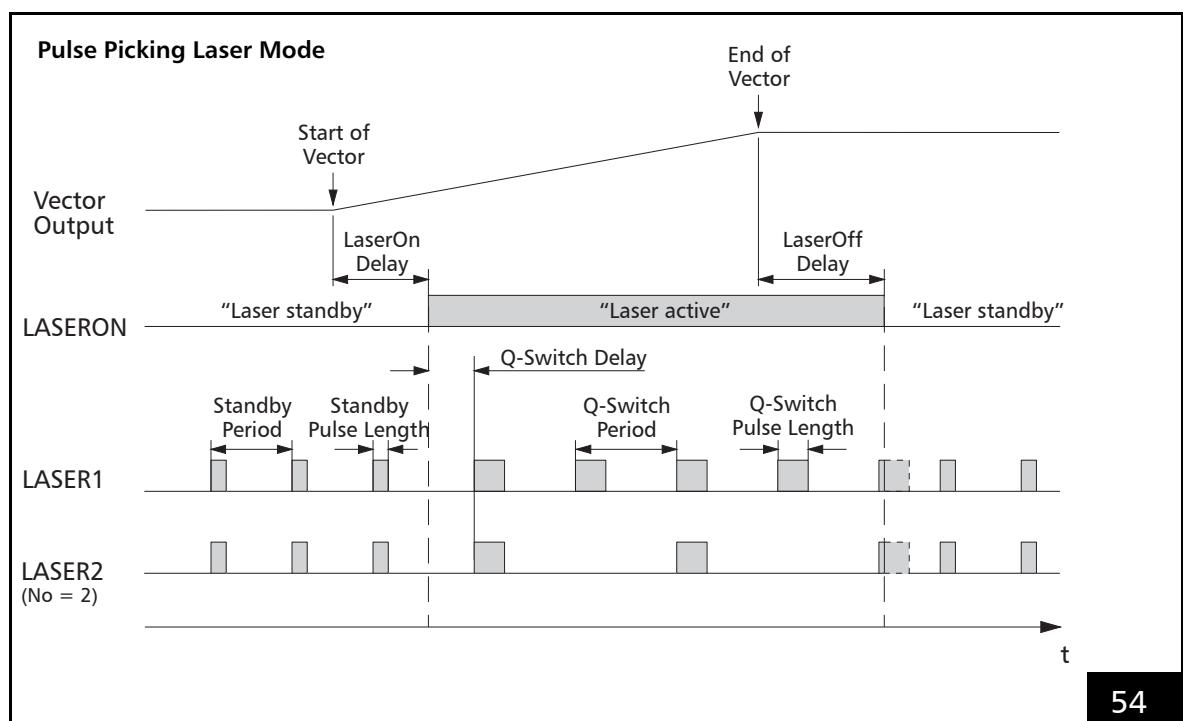
For "laser active" operation (if applicable after a LaserOn delay and if enabled by `set_laser_control` or `enable_laser`):

- The LASERON signal is switched on
- A modulation signal is provided at the LASER1 output with pulse lengths and periods that can be defined by `set_laser_pulses`, `set_laser_pulses_ctrl` or `set_laser_timing`
- The LASER2 output emits the same signals as the LASER1 output (in phase), but only every No^{th} pulse in accordance with the parameter No in `set_pulse_picking` or `set_pulse_picking_list`.

For "laser standby" operation (if applicable after a LaserOff delay):

- The LASERON signal is switched off
- Standby pulses are provided in phase at the LASER1 and LASER2 outputs with pulse lengths and periods that can be defined by `set_standby` or `set_standby_list`.

You can activate constant pulse length mode (a special case of the pulse picking laser mode) by the command `set_laser_control`(bit #7 = 1). In this mode, the "laser active" LASER2 signal is outputted with a fixed pulse length that is independent of the LASER1 signal (the desired LASER2 pulse length is specified by the `set_pulse_picking_length` command). This allows you to also use the LASER2 signal as a gate signal for short LASER1 pulses.



Laser control timing diagram for pulse picking laser mode (with active-high laser control signals and $No = 2$)



Notes

- As in YAG mode 5, the time interval between LaserOn and the first Q-Switch pulse (LASER1/LASER2) can be specified by **set_qswitch_delay** or **set_qswitch_delay_list**. The LASER2 signal synchronously starts with the LASER1 signal.
- The LASER1/LASER2 signals are activated by setting the pulse length to a nonzero value and is deactivated by setting the pulse length to zero (default setting).
- Note that *half* of the output period must be specified for the LASER1/LASER2 signals.
- The commands **set_laser_pulses** and **set_laser_timing** are *list* commands. They can be used within a list, for instance, for changing the laser power at any time between two list commands.
- The actual output period and the pulse lengths of laser signals LASER1 and LASER2 (but not the laser delays) are dependent on the time base. For RTC5 mode, this is fixed at 64 MHz (1 bit equals 1/64 µs). For RTC4 compatibility mode (for "laser active" operation), the time base can be set by **set_laser_timing** (but not by **set_laser_pulses** or **set_laser_pulses_ctrl**) for either 1 MHz (1 bit equals 1 µs) or 8 MHz (1 bit equals 1/8 µs). For RTC4 compatibility mode, SCANLAB generally recommends setting the time base to 8 MHz (by **set_laser_timing(....,...,1)**). A time base of 1 MHz should only be chosen if necessary. The time base for standby signals is always 1/64 µs (in RTC5 mode) or 1/8 µs (in RTC4 compatibility mode).

- The command **set_laser_control** (bit#0) can be used to specify whether a (for "laser active" operation) started modulation pulse (LASER1 or LASER2) should execute to completion or be cut off when it has not yet fully executed when the LASERON signal is switched off (see **figure 54**). This cannot be specified for standby signals; if applicable, these are cut off.
- The LASER1 signals may be set (by **set_laser_pulses** etc.) prior to or after setting the pulse-picking laser mode.
- **set_pulse_picking** and **set_pulse_picking_list** overwrite any laser mode setting previously set by **set_laser_mode**. A subsequent laser mode setting (by **set_laser_mode**) switches off the pulse-picking laser mode.

7.4.9 Automatic Laser Control

Automatic laser control commands can be used to achieve automatic readjustment of “laser active” laser control signals – and thus of the laser power – even during execution of vector and arc commands.

On the one hand, this allows compensation of laser impact variations (and thus variations in the processing results) arising from (e.g.) changes in power density, spot size or processing speed, even for constant predefined laser power. In marking processes, for example, this compensation results in more uniform marking strength and more uniform line widths during hatching.

On the other hand, the laser impact along a vector can also be deliberately readjusted. e.g. to achieve the desired weld seam shape in laser welding applications.

The automatic laser control commands supplement commands described in the preceding sections of this chapter, such as `set_laser_pulses`, `write_da_x_list` or `write_8bit_port_list`, which only define a *constant* laser power for subsequent mark and arc commands.

For automatic laser control, a position-dependent, speed-dependent or vector-defined correction of laser control signals can be activated (these three correction possibilities can also be freely combined with each other). Here, the selected laser control signals are readjusted in accordance with scanning movements of the scan system. As an alternative to (galvanometer scanner-) speed-dependent laser control, encoder-speed-dependent correction (combinable with position-dependent and vector-defined laser control) can be enabled.

- With *position-dependent* laser control (see [page 161](#)), position-dependent (radial) variations in laser impact can be compensated during execution of vector and arc commands. Such variations may occur, for example, from optical system aberrations (e.g. objective edge diminution, i.e. power losses at the objective edge). Depending on the optical system, variations may also occur for a non-perpendicular incidence of the laser beam at the processed material (i.e. angles of incidence larger than zero) for increasing distances to the image field center – which alters the reflection and the laser spot. Position-dependent laser control allows definition of an automatic radial correction (dependent on the distance to the image field center).

- *Speed-dependent* laser control (see [page 165](#)) allows compensation of speed-dependent variations in laser impact during execution of vector and arc commands. Such variations can occur if the speed changes during a mark command or across multiple mark commands – for example:
 - resulting from acceleration or deceleration scanner motions at the beginning or end of a vector or from directional changes within a polyline
 - resulting from automatic readjustment to the 10 µs clock of the stepwidth and effective marking speed (short-vector jitter)
 - with timed mark commands resulting from the automatic readjustment of the marking speed in accordance with the predefined marking duration (timed mark commands allow avoidance of effects, such as fluttered edges at the vector’s end, that might occur with normal mark commands)
- *Vector-defined* laser control (see [page 166](#)) allows linear readjustment of a laser signal parameter along a mark or jump vector. This allows, for example, to compensate laser impact variations along a vector on uneven material surfaces (resulting from an angle of incidence that varies along the vector)
- *Encoder-speed-dependent* laser control (see [page 168](#)), allows the laser impact to be controlled based on the current encoder speed (counter pulses in the most recent 10 µs interval) of an encoder counter, e.g. corresponding to the current conveyance speed of a transport system (and therefore to the speed of the workpiece).

Selectively, automatic laser control adjusts one of the following signal parameters:

- the 12-bit output value at the ANALOG OUT1 or ANALOG OUT2 analog output ports of the 15-pin D-SUB LASER connector (see also [page 48](#))
- the output value at the 16-bit digital TTL output of the EXTENSION 1 socket connector (see also [page 51](#))
- the output value at the 8-bit digital output port of the EXTENSION 2 socket connector (see also [page 52](#))
- the pulse length (`PulseLength`) or output period (`HalfPeriod`) of the laser signals LASER1 and LASER2
- Only for vector-defined laser control: focus shift for subsequent parameterized vector output (requires enabling the 3D option as well as loading and assigning a 3D correction file)

For position-dependent and speed-dependent or encoder-speed-dependent laser control, only a common signal parameter selection can be defined. Thus, these corrections are applied to the same signal parameter. In contrast, a different signal parameter can be selected for vector-dependent laser control.

The automatically readjusted final output values for laser control can be logged by `set_trigger/set_trigger4` (Signal1/2=24 and 31...38). This allows, for example, monitoring the effect of varying speed during acceleration or deceleration phases of the scanner motion.

Notes

- A varying laser impact resulting from an explicit change of the marking speed between to marking commands (by calling `set_mark_speed`) is *not* compensated by automatic laser control.
- If automatic laser control readjusts the 12-bit output values at the ANALOG OUT1 or ANALOG OUT2 output ports or pulse lengths (`PulseLength`) or output periods (`HalfPeriod`) of the laser signals LASER1 and LASER2, then automatic laser control should *not* be used in conjunction with the pixel output or softstart modes (these functions cause mutual disruption). In contrast, automatic readjustment of values output at the 16-bit or 8-bit digital output ports can be used simultaneously and unrestrictedly with the pixel output or softstart modes.
- Users are responsible for determining the automatic laser control parameter settings and tables necessary for their applications. This can require considerable effort, but it is recommended especially if exceptionally high precision is needed.
- Automatic laser control cannot be combined with variable laser control by the McBSP/SPI interface (see `set_multi_mcbsp_in`).



Position-Dependent Laser Control

To initialize position-dependent laser control, **set_auto_laser_control** must be used for specifying which signal parameter is readjusted (by the `Ctrl` parameter), and the **load_position_control** command used for loading a scaling function.

For subsequent mark and arc commands (also timed mark commands), the RTC5 then performs an automatic position-dependent correction of the selected signal parameter (i.e. a correction dependent on the radial distance between the current output position and the image field center). The selected signal parameter is multiplied by the position-dependent (radial) correction factor defined by the loaded scaling function. The scaling factors are relative to a 100% value defined by the **set_auto_laser_control** command (`Value` parameter).

Because position-dependent changes in laser impact are system dependent, the to-be-loaded scaling function must be defined by users in a text file (see "[Notes on Loading a Scaling Function](#)", page 162).

Position-dependent compensation is calculated based on the current cartesian control values `Sample<X . . Y>` (see also [set_trigger/set_trigger4](#)). Where applicable, these encompass wobble and Processing-on-the-fly corrections, but not image field corrections, coordinate transformations or gain and offset corrections.

General Notes

- **set_auto_laser_control** (`Ctrl` parameter) allows initialization of not only position-dependent, but also speed-dependent or encoder-speed-dependent laser control (see [page 165](#) and [page 168](#)). The signal parameter selection and the 100% value defined by **set_auto_laser_control** (`Ctrl` and `Value` parameters) apply to both corrections and the correction factors for position- and speed-dependent or encoder-speed-dependent laser control are then multiplicably combined into a common factor.

If only position-dependent laser control is required, then speed-dependent or encoder-speed-dependent laser control can be deactivated by **set_auto_laser_control** (parameter `Mode=0`).

In contrast, if only speed-dependent or encoder-speed-dependent laser control is required, then position-dependent correction can be effectively deactivated by using **load_position_control**

(`Name = 0`) to load a scaling function with a constant value of 1.0 for the complete positioning area (`Scale(Position)=1.0`, same as the initial state after **load_program_file**, see also [page 162](#)).

- If, in addition, vector-defined laser control is activated for the same signal parameter (`Ctrl` parameter), then the 100% value (`Value` parameter) might be changed by the vector-defined laser control (see [page 167](#)).
- The total correction factor (if applicable, of position- and speed-dependent or encoder-speed-dependent laser control) cannot exceed 4.0 and is clipped to this value if necessary.
- Because laser power is not necessarily fully proportional to the output values of the selected signal parameter, users can also use **load_auto_laser_control** to load a nonlinearity curve that defines this relationship. The total correction factor would then (after possible clipping, see above) be modified multiplicatively by a nonlinearity factor in accordance with this nonlinearity curve. The nonlinearity curve must be determined by the user for his specific situation and then defined in a text file (see "[Notes on Loading and Determining Nonlinearity Curves](#)", [page 163](#)).
- Additionally, the selected signal parameter cannot exceed the maximum allowed range nor the range specified by **set_auto_laser_control** (`MinValue` and `MaxValue` parameters). If necessary, the output value (after possible modification by a specified nonlinearity factor) is appropriately clipped.



- After successful initialization of position- and/or speed-dependent or encoder-speed-dependent laser control by **set_auto_laser_control**, then – every time the laser is switched off at the end of marking or when position- and/or speed-dependent or encoder-speed-dependent laser control is set to another **Ctrl** parameter by **set_auto_laser_control** or deactivated by **set_auto_laser_control** (**Ctrl** = 0) – the selected signal parameter is set to the following value:
 - the 12-bit output value at the ANALOG OUT1 or ANALOG OUT2 analog output port, the output value at the 8-bit digital output port or the output value at the 16-bit digital output is set to the value previously defined by **set_port_default** or **set_laser_off_default** (or – if no default value was defined – to the maximum allowed value)
 - the pulse length (**PulseLength**) or output period (**HalfPeriod**) of laser signals LASER1 and LASER2 are set to the 100% value defined by **set_auto_laser_control** (**Value** parameter) (an additionally activated vector-defined laser control has no effect here).
- The position-dependent laser control's to-be-controlled signal parameter, the 100% value and its corresponding limit values can be specified not only with **set_auto_laser_control**, but also by **set_auto_laser_params** or **set_auto_laser_params_list**.

Notes on Loading a Scaling Function

The command **load_position_control** loads a table for the scaling function **Scale(Position)** from an ASCII text file. The text file can contain one or more tables.

Each table can contain up to 50 data points (**Position | Scale(Position)**) for various positions. The RTC5 determines the scaling function **Scale(Position)** from this data by linear interpolation.

The following rules apply for these tables:

- Each table must start with the instruction (**Caption**)


```
[PositionCtrlTable<No>]
```

 where **<No>** must be replaced by a nonnegative integer which denotes the table number.
- If the table contains multiple


```
[PositionCtrlTable<No>]
```

 entries with the same **<No>**, then only the instructions after the first entry is used; instructions that follow further entries are ignored. Only instructions up to the next '[' character (that is not preceded by a semicolon) are used.
- Each data point (**Position | Scale(Position)**) is described by two instructions:


```
Position<n> = <Value>
Scale<n> = <Value>
```

 where **<n>** must be replaced by an integer ($1 \leq <n> \leq 50$) which denotes the number of the data point. The values **<Value>** can be specified as (unsigned) floating point numbers. Use the period (.) as the decimal separator.
- If the table contains multiple data points with the same **Index <n>**, then the most recently read one is used and the previous ones ignored.
- If the table contains multiple data points with the same position value **Position**, then the data point with the largest **Index <n>** is used and the others ignored. Equality is checked to within ± 0.01 .
- The position value is specified radially as the distance between the to-be-marked point and the coordinate midpoint ($= (x^2 + y^2)^{1/2}$) as a percent value (percent of half the image-field side length). Example: ($X_{\max}|0$) corresponds to 100%, ($X_{\max}|Y_{\max}$) corresponds to $2^{1/2} \times 100\%$ (with $X_{\max}, Y_{\max} = 2^{19}$ or 2^{15} in RTC4 compatibility mode).

- For `<Value>`, the following ranges apply:
 $0.0 \leq Position \leq 150.0$ and
 $0.0 \leq Scale(Position) \leq 4.0$.
(Though the RTC5 uses the scaling function only for the range $0.0 \leq Position \leq 141.42$, the interpolation also takes into account values in the range $141.42 \leq Position \leq 150.0$)
- Each instruction must be in a separate line.
- Spaces and tabs in a line (e.g. between '=' and `<Value>`) are ignored.
- Empty lines are ignored.
- Data points with invalid values are ignored.
- The data point of a particular index `<n>` is ignored if the corresponding `Position<n>` and/or `Scale<n>` definition is missing.
- The semicolon ';' can be used for comments. All characters in a line following a semicolon are ignored.
- The instructions for data points in the table can be ordered as desired.
- Indices for data point pairs in the table can be selected as desired within the range [1...50] (the table is then automatically sorted by ascending position values).
- If the table contains no valid data point, the command `load_position_control` has no effect (return value 1 or 13).
- If there is no entry for `Position = 0.0`, then an entry with `Scale = Min(Scale<i>)` is inserted (the smallest allowed value defined in the table is used for lower positions values). Likewise for `Position = 150.0` with `Max(Scale<i>)`.
- If the selected text file only contains a single valid data point with `Scale<n> = S`, then (for the entire position range) the scaling function `Scale(Position) = S` is loaded with the effect that no position-dependent correction is executed. A 100% value set with `set_auto_laser_control` (Value parameter) is multiplied by `S`. For `S = 1.0`, position-dependent correction is therefore fully switched off. Alternatively, this can also be achieved with `Name = 0` in `load_position_control`.

During initialization of the RTC5 (with `load_program_file`), the scaling function `Scale(Position) = 1.0` is loaded for the full position range, with the effect that no position-dependent correction is executed.

Notes on Loading and Determining Nonlinearity Curves

The command `load_auto_laser_control` loads a table for the nonlinearity curve `Scale(Percent)` from an ASCII text file. The text file can contain one or more tables.

Each table can contain up to 50 data points (`Percent | Scale(Percent)`) for various percent values. The RTC5 determines the nonlinearity curve `Scale(Percent)` from this data by linear interpolation.

For the table, the following rules apply:

- Each table must start with the instruction (Caption)
`[AutoLaserCtrlTable<No>]`
where `<No>` must be replaced by a nonnegative integer which denotes the table number.
- If the table contains multiple `[AutoLaserCtrlTable<No>]` entries with the same `<No>`, then only the instructions after the first entry is used; instructions that follow further entries are ignored. Only instructions up to the next '[' character (that is not preceded by a semicolon) are used.
- Each data point (`Percent | Scale(Percent)`) is described by two instructions:
`Percent<n> = <Value>`
`Scale<n> = <Value>`
where `<n>` must be replaced by an integer ($1 \leq <n> \leq 50$) which denotes the number of the data point. The values `<Value>` can be specified as (unsigned) floating point numbers. Use the period (.) as the decimal separator.
- If the table contains multiple data points with the same Index `<n>`, then the most recently read one is used and the previous ones ignored.
- If the table contains multiple data points with the same percent value `Percent`, then the data point with the largest Index `<n>` is used and the others ignored. Equality is checked to within $\pm 0.01^\circ$.

- The percent value refers to the 100% value (that can additionally be further changed through vector-defined laser control, see [page 166](#)) set by **set_auto_laser_control** (Value parameter).

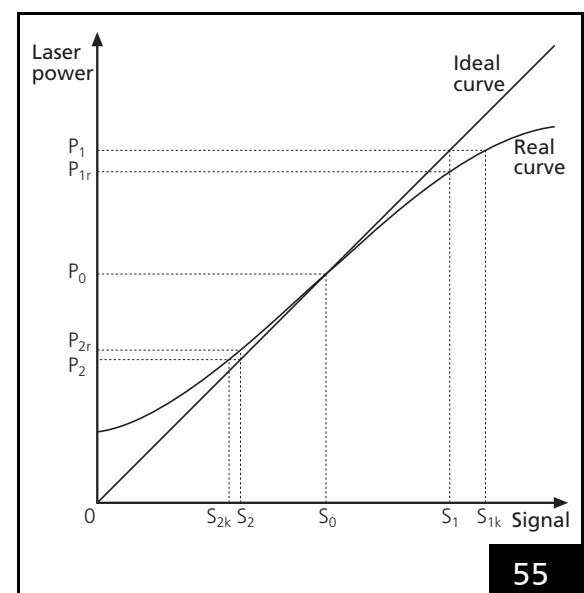
In the following example, a nonlinearity factor of 1.2 is set for a 1.5x multiple of the 100% value:

Percent<n> = 150
Scale<n> = 1.2

- For <Value>, the following ranges apply
 $0.0 \leq \text{Percent} \leq 400.0$ and
 $0.0 \leq \text{Scale}(\text{Percent}) \leq 4.0$.
- Each instruction must be in a separate line.
- Spaces and tabs in a line (e.g. between '=' and <Value>) are ignored.
- Empty lines are ignored.
- Data points with invalid values are ignored.
- The data point of a particular index <n> is ignored if the corresponding Percent<n> and/or Scale<n> definition is missing.
- The semicolon ';' can be used for comments. All characters in a line following a semicolon are ignored.
- The instructions for data points in the table can be ordered as desired.
- Indices for data point pairs in the table can be selected as desired within the range [1...50] (the table is then automatically sorted by ascending percent values).
- If the table contains no valid data point, then the command **load_auto_laser_control** has no effect (return value 1 or 13).
- If there is no entry for **Percent = 0.0**, then an entry with **Scale = Min(Scale<i>)** is inserted (the smallest allowed value defined in the table is used for lower percent values). Likewise for **Percent = 400.0** with **Max(Scale<i>)**.

During initialization of the RTC5 (with **load_program_file**), the function **Scale(Percent) = 1.0** is loaded for the full percent range (no nonlinearity). Alternatively, this can also be achieved with **Name = 0** in **load_auto_laser_control**.

The example diagram in [figure 55](#) illustrates how the nonlinearity curve can be determined.



Laser power progression – example of determining a nonlinearity curve

The straight line in the diagram describes ideal linear behavior in the relationship between laser power and the laser control signal parameter (here, the term laser power also represents the pulse frequency = $0.5/\text{LaserHalfPeriod}$). The curved line is an example of a realistic relationship: here, the laser power rises/falls more slowly than the signal parameter.

S_0 is the signal parameter value defined as the 100% value and P_0 is the associated laser power. At point $(S_0 | P_0)$, the two curves intersect (this corresponds to the data point **Percent0 = 100, Scale0 = 1.0**).

An increase (decrease) of the signal parameter to S_1 (S_2) results in an ideal laser power P_1 (P_2) and a real laser power P_{1r} (P_{2r}). For the actually desired laser power P_1 (P_2), a corrective signal parameter value S_{1k} (S_{2k}) is needed. In this example, the following two value pairs would then be entered as data points for the nonlinearity curve:

$$\text{Percent1} = S_1/S_0 \times 100 = P_1/P_0 \times 100$$

$$\text{Scale1} = S_{1k}/S_1$$

$$\text{Percent2} = S_2/S_0 \times 100 = P_2/P_0 \times 100$$

$$\text{Scale2} = S_{2k}/S_2$$

Speed-Dependent Laser Control

For initializing speed-dependent laser control, the command **set_auto_laser_control** must be called for specifying which signal parameter is readjusted (by the `Ctrl` parameter) and which quantity v_a shall be used as the calculation base for speed-dependent corrections (`Mode` parameter).

For subsequent mark and arc commands (also timed mark commands), the RTC5 then performs an automatic speed-dependent correction of the selected signal parameter (i.e. a correction dependent on the current speed): The 100% value defined by **set_auto_laser_control** (Value parameter) is multiplied by the correction factor v_a/v_s (exception: for the output period `HalfPeriod`, the correction factor is v_s/v_a). Here, v_s is the marking speed defined with **set_mark_speed**, and v_a depends on the definition by **set_auto_laser_control**:

- For `Mode` = 1, v_a is the actual set speed derived from the microvector's step size (in set coordinates).
- For `Mode` = 2 (only selectable for iDRIVE scan systems), v_a is the current speed returned by the scan system, derived as per $(v_x^2 + v_y^2)^{1/2}$

Notes

- For **set_auto_laser_control** `Mode` = 1:
With mark or arc commands, the actual set speed can deviate from the marking speed defined by **set_mark_speed** because the marking time must be an integer-multiple of 10 µs (the command is executed by a corresponding number of microsteps) and the step size (and therefore effective marking speed = actual set speed) is automatically adjusted thereto. Particularly for short lines, the actual set speed is therefore noticeably variable. Such length-dependent speed variations – without speed-dependent laser control – can, for example, cause hatching line widths to vary.

For timed mark commands, the set speed is generally determined in accordance with the specified vector length and marking time, and therefore usually differs from the marking speed defined by **set_mark_speed**.

Speed variations within a marking command cannot be compensated in `Mode` = 1. Anyway, most standard scan systems cannot supply the necessary information to the RTC5.

- For **set_auto_laser_control** `Mode` = 2, correction is always performed in accordance with the current speed so that automatic laser control can also compensate speed variations during marking commands (e.g. during acceleration or deceleration phases). However, transfer times to/from the scan head delays evaluation of the current speed by a few clock cycles.

This functionality is only practical for scan heads equipped with special firmware which compensates those transfer time delays. If you want to use this functionality, but do not yet have such a specially-equipped scan head, then contact SCANLAB.

- With `set_auto_laser_control` Mode = 6 the current galvanometer scanner speed (Mode = 2) can be combined with the present encoder speed (Mode = 5). But this requires the Processing-on-the-fly correction to be enabled and to be activated by `set_fly_x`, `set_fly_y` or `set_fly_2d`. The current marking speed serves as reference speed (as with Mode = 2). The `set_encoder_speed` command (which is used for Mode = 5) is not taken into account in Mode = 6 (see also section "Encoder-Speed-Dependent Laser Control", page 168).
- Variable laser impact resulting from explicit changes in marking speed between two mark commands by `set_mark_speed` cannot be compensated through speed-dependent laser control.
- Observe all notes on position-dependent laser control in the section "General Notes", page 161: the correction factor for speed-dependent laser control is, if necessary, multiplied by a position-dependent laser-control factor and a nonlinearity factor before being applied to the 100% value.
- If a negative LaserOn delay has been set to preheat the material before marking (see page 111), then speed-dependent laser control might in some circumstances actually be counterproductive, because this would attenuate laser power in the acceleration phase at the beginning of a marking operation. But this can be counteracted with the `set_auto_laser_control` parameters `MinValue` and `MaxValue`.
- The speed-dependent laser control's to-be-controlled signal parameter, the 100% value and its corresponding limit values can be specified not only with `set_auto_laser_control`, but also by `set_auto_laser_params` or `set_auto_laser_params_list`.

Vector-Defined Laser Control

Vector-defined laser control makes possible to linearly vary a signal parameter along a mark or jump vector.

To initialize vector-defined laser control, `set_vector_control` must be called to define the signal parameter to be varied (by the `Ctrl` parameter) and to set the initial value of this signal parameter (`Value` parameter).

For subsequent para-mark and para-jump commands (i.e. parameterized mark and jump commands), the RTC5 then linearly varies the selected signal parameter along with the varying of coordinates. The start value for the first para command is the initial value defined with `set_vector_control` (`Value` parameter)⁽¹⁾. The signal parameter is microvectorized like a normal coordinate and linearly varied until the defined end value (defined by the `P` parameter). The end value is the start value for a subsequent para command.

The following para-mark and para-jump commands are available:

- `para_mark_abs`, `timed_para_mark_abs`
- `para_mark_rel`, `timed_para_mark_rel`
- `para_jump_abs`, `timed_para_jump_abs`
- `para_jump_rel`, `timed_para_jump_rel`

If the 3D option is enabled, 3D para commands can also be executed:

- `para_mark_abs_3d`, `timed_para_mark_abs_3d`
- `para_mark_rel_3d`, `timed_para_mark_rel_3d`
- `para_jump_abs_3d`, `timed_para_jump_abs_3d`
- `para_jump_rel_3d`, `timed_para_jump_rel_3d`

The para command `para_laser_on_pulses_list` can be used for marking points.

(1) With `Ctrl` = 7 (focus shift), a hard jump of the Z coordinate to the initial value might occur.

Notes

- If a para command's desired start value differs from the previous para-command's end value, then the corresponding signal-parameter change between both commands can be achieved by **para_jump_rel** with a jump "to the same place" or by a renewed call of **set_vector_control** with a new initial parameter value.
If, between two para-commands, a list command (e.g. **set_laser_pulses**, **write_da_x_list** or **write_8bit_port_list**) is called, then the signal parameter also temporarily changes. Even so, the start value for the second para-command is still the end value of the first para-command. Control commands such as **set_vector_control** or commands that explicitly change the **Ctrl** parameter's set value (e.g. **write_da_x** or **write_8bit_port**) should be avoided during processing of a list of para-commands.
- Vector-defined laser control has *direct* effects only with para-commands (normal mark and jump commands do not change the currently set value of the **Ctrl** parameter), but has indirect effects under some circumstances with normal mark and arc commands or timed mark commands (see also "**General Notes**", page 161). If, in addition to vector-defined laser control (**Ctrl** = 1...6), a position- and/or speed-dependent or encoder-speed-dependent laser control of the same signal parameter has simultaneously been activated, then the current signal

parameter value for vector-defined laser control becomes the basis for the 100% value of position- and/or speed-dependent or encoder-speed-dependent laser control:

- During initialization of vector-defined laser control by **set_vector_control**, the 100% value is set to the initial value defined by **set_vector_control** (**Value** parameter).
- During execution of subsequent para-commands, the 100% value is continuously varied in accordance with the current signal parameter value.
- During deactivation of vector-defined laser control by **set_vector_control** (parameter **Ctrl**=0), the 100% value is reset to the initialized value (**Value** parameter of **set_auto_laser_control**, **set_auto_laser_params** or **set_auto_laser_params_list**).
- For para commands, a nonlinearity curve loaded by **load_auto_laser_control** only effects signal parameters selected by **set_vector_control** if position- and/or speed-dependent or encoder-speed-dependent laser control has been activated for the same signal parameter (**Ctrl** = 1...6).
- During execution of para commands, the Sky writing mode (see **page 127**) is *not* taken into account (but also not deactivated).
- After vector-defined laser control is activated by **set_vector_control** (**Ctrl** = 7), the focus shift for 2D or 3D para commands changes, if the 3D option is enabled and a 3D correction file has been loaded and assigned.



Encoder-Speed-Dependent Laser Control

Encoder-speed-dependent laser control allows the most recently read encoder speed (counter pulse rate) of an encoder counter (or of both encoder counters) to be used for controlling a laser signal parameter. This lets you, for instance, adjust the laser's impact on moving objects in accordance with the transport system's current conveyance speed.

For initializing encoder-speed-dependent laser control, the command `set_auto_laser_control`, `set_auto_laser_params` or `set_auto_laser_params_list` must be called for specifying (by the `Ctrl` parameter) which signal parameter is readjusted according to the current encoder speed (counter pulse rate) (parameter `Mode` = 5).

By `set_encoder_speed_ctrl` or `set_encoder_speed`, you can choose which of the two encoders should be used for measuring the speed (or if both encoders should be used to determine a vectorial encoder velocity). Here, you can also specify a target encoder speed and a smoothing factor for a two-stage low-pass filter (see command description).

For subsequent mark and arc commands (also timed mark commands), the RTC5 then performs an automatic correction of the selected signal parameter dependent on the current encoder speed: The 100% value defined by `set_auto_laser_control`, `set_auto_laser_params` or `set_auto_laser_params_list` (Value parameter) is multiplied by the correction factor v_a/v_s (exception: for the output period `HalfPeriod`, the correction factor is v_s/v_a). Here, v_s is the target encoder speed defined with `set_encoder_speed` and v_a is the current encoder speed.

Notes

- Encoder-speed-dependent laser control can be combined with position-dependent and vector-defined laser control, and as of DLL 540 with speed-dependent laser control as well.
- Observe all notes on position-dependent laser control in the section "[General Notes](#)", page 161: the correction factor for encoder-speed-dependent laser control is, if necessary, multiplied by a position-dependent laser-control factor and a nonlinearity factor before being applied to the 100% value.

7.4.10 Output Synchronization

The RTC5 provides the so-called (galvanometer) "output synchronization" functionality for synchronizing the scan system's scanning motions (during all marking commands) to the laser pulses of a free-running laser.

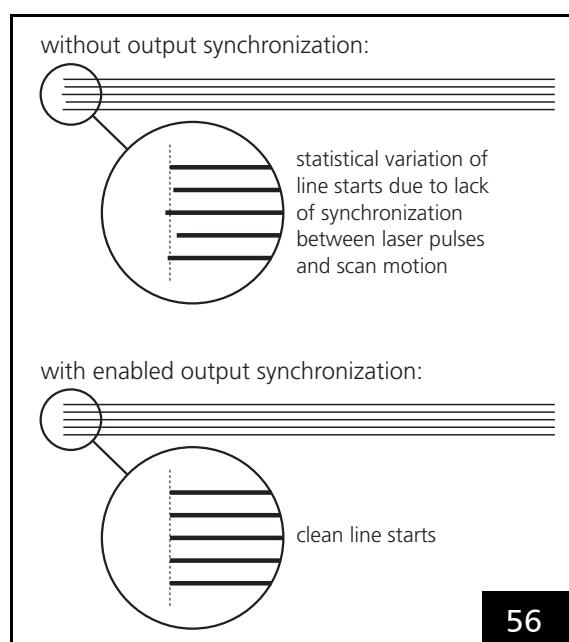
The laser pulse signal (the synchronization master clock) must be supplied at the LASER connector's DIGITAL IN1 digital input (see [page 48](#)).

Output synchronization is enabled or disabled by bit#6 of [set_laser_control](#).

When enabled, output synchronization affects all marking commands (mark, arc and ellipse). At each marking command's start, the RTC5 determines the time shift between that marking command's starting time and the first laser pulse after LaserOn⁽¹⁾ and appropriately corrects the scan system's output values. The RTC5 thus ensures that the galvanometer scanner's position upon the first laser pulse output after LaserOn does not depend on random phase shifts of the laser pulse. This way, jittery line images can be avoided (see [figure 56](#)).

Notes (Valid as of Version RBF 527)

- The supplied laser pulse signal must have a frequency between 50 kHz and 6.4 MHz and its pulse length must exceed 0.080 µs.
- The output period of the laser control signals must be set according to the laser pulse frequency in the user program (before enabling output synchronization) by [set_laser_pulses](#), [set_laser_pulses_ctrl](#) or [set_laser_timing](#). This also applies, if laser mode 4 (see [page 154](#)) standby signals are used as external input.
- For frequencies between 50 kHz and 100 kHz, the LaserOn delay must be set to 50 µs, otherwise the LaserOn delay must exceed 40 µs.
- If no laser pulse appears at the DIGITAL IN1 digital input within 20 µs of a marking command's start, then no output synchronization is performed on that command's output.
- Output synchronization can also be used in conjunction with Sky-Writing.
- Output synchronization cannot be used together with pixel mode.



56

Example of line image marked with a free-running laser

(1) the first laser pulse output following the LaserOn delay

7.5 Marking Dates, Times and Serial Numbers

Many applications need to mark the current time and date or product serial numbers. For this purpose, the RTC5 command set includes a set of supporting commands.

Before times, dates and serial numbers can be marked, the corresponding characters and text strings must be defined as indexed characters and indexed text strings. This is described “[Defining Indexed Text Strings for Times, Dates and Serial Numbers](#)” on page 88. Separate text strings can be defined for marking times/dates and serial numbers.

7.5.1 Marking the Date and Time

Before times and dates can be marked, the RTC5 (after each start-up) needs to synchronize with the PC’s time and obtain the current values for time and date:

- Calibration of the RTC5’s 24-hour time and date is achieved through comparison with the PC’s time by `time_update`. Afterward, the RTC5 internally maintains the date and time with a quartz-controlled 10 µs clock.
- The current time and date can be queried or stored with the command `time_fix`, `time_fix_f` or `time_fix_f_off`.

Subsequently, the time (hours, minutes, seconds) can be marked by `mark_time` or `mark_time_abs` and the date (year, month, day, day-of-the-week) by `mark_date` or `mark_date_abs`. These mark commands use the time and date supplied by `time_fix`, `time_fix_f` or `time_fix_f_off` and start output of the previously-defined indexed text strings for time and date.

For marking date and time with the RTC5, one can choose between Gregorian dates and Julian days as well as between the 12-hour and 24-hour time.

For marking dates of expiry (or something similar), you can also use `time_fix_f_off` to fix a forward date based on the current date and current time.

7.5.2 Marking Serial Numbers

Serial numbers containing up to 12 digits can be marked by the commands `mark_serial` or `mark_serial_abs`. The commands initiate output of command lists for the numerals 0...9 that were previously defined as indexed characters or text strings. You can also specify if and how leading zeros should be handled.

The RTC5 Board manages up to 4 serial-number-sets (each with its own serial number and increment size). After initialization with `load_program_file`, serial-number-set 0 is selected. When using multiple serial-number-sets, you need to first select the desired set by `select_serial_set` or `select_serial_set_list` (see notes below).

The control commands `set_serial` and `set_serial_step` and the list command `set_serial_step_list` let you specify a starting serial number (max. 10 digits) and an increment size for each serial-number-set. After initialization with `load_program_file`, all starting serial numbers are set to 0 and all increment sizes to 1.

With each call of `mark_serial` (or `mark_serial_abs`), the current serial number of the currently selected set is BCD-encoded and (even before execution of the BCD-encoded serial number marking) the serial number is incremented by the specified increment size. If a serial number is to be omitted, then `mark_serial` can also execute a blank marking (digits = 0), which increments the serial number by 1 (*not* by the specified increment size).



Notes

- If a serial-number-set is to be marked by `mark_serial` or `mark_serial_abs`, then you can only select that set by the list command `select_serial_set_list`. `mark_serial`, `mark_serial_abs` and `set_serial_step_list` are always applied to the serial-number-set most recently selected by `select_serial_set_list` (or to serial-number-set 0 after `load_program_file`).
- You can use the control command `get_list_serial` to query the number of the serial-number-set most recently selected by `select_serial_set_list` as well as the current serial number of that set (i.e. typically the serial number most recently marked by `mark_serial` or `mark_serial_abs`). This also lets you determine (among other things) whether the current number was or was not incremented after an uncontinued aborted list (if necessary, the number can be corrected by `set_serial` or `set_serial_step`, but only within a 10-digit range).
- The control command `select_serial_set` lets you select (independently of selection by `select_serial_set_list`) a serial-number-set for the control commands `set_serial_step` and `set_serial` (e.g. for assigning parameters for a serial-number-set while marking another serial-number-set. Note that the RTC5 does not suppress modifying parameters of the serial-number-set currently being marked). The control commands `set_serial_step` and `set_serial` always apply to the serial-number-set most recently selected by `select_serial_set` (or to serial-number-set 0 after `load_program_file`).
- `get_serial` returns the current serial number of the serial-number-set selected by `select_serial_set` (if multiple serial-number-sets exist, then the returned serial number is not necessarily the most recently marked serial number).
- Older software versions (DLL 536, OUT 536 and earlier) only allowed one serial-number-set. Because version DLL 537, OUT 537 and higher select serial-number-set 0 after `load_program_file`, you can continue using already-existing user programs (even those without the new commands `select_serial_set` and `select_serial_set_list`).
- The control command `set_max_counts` allows specification of the maximum number of external list starts and thus the maximum number of externally started markings. The number of external list starts can be obtained with the control command `get_counts`. When a single serial-number-set is used, these commands let you indirectly set the maximum serial number and query the current serial number. But when multiple serial-number-sets are used, these commands do not differentiate between the various serial-number-sets.



8 Advanced Functions for Scan Head and Laser Control

8.1 iDRIVE Functions

SCANLAB's iDRIVE scan systems (intelliSCAN, intelliSCAN_{de}, intelliDRILL, intellicube, intelliWELD, varioSCAN_{de}) utilize the company's iDRIVE technology. This new servo and control approach exploits the advantages of fully digital servo electronics to deliver significantly expanded functionality. An enhanced transfer protocol between the servo electronics and the controller facilitates support of all the new features (see also the following section).

As a result, scan systems with iDRIVE functionality allow to configure a number of settings. For example, users can

- select the data type to be transmitted from scan system to the controller,
- select an appropriate dynamic configuration (tuning),
- set a desired PosAcknowledge threshold value,
- change the scan system's effective calibration,
- configure the scan system's start behavior or
- perform a fault diagnosis or verify intact data transfer capability.

With an RTC5, most iDRIVE functions can be executed by **control_command**. During this command's execution, no position data is transmitted from the RTC5 to the scan system. As a result, the galvanometer scanners' movements might briefly be interrupted. However, for scan systems with SL2-100 interfaces (operated without an XY2-100 converter) specific iDRIVE functions can be executed by the command **send_user_data** simultaneously to normal communication, such that neither the transmission of position data nor the movements of the galvanometer scanners are interrupted.

All iDRIVE functions are described in the following sections.

8.1.1 Transfer Protocol

Data transfer between RTC5 and scan systems follows the SL2-100 protocol, which supports the full functionality of iDRIVE technology. With iDRIVE scan systems this protocol allows, for example, status signals of the X and Y axes to be separately and simultaneously evaluated. For a 3D scan system, Z-axis status signals can be simultaneously queried by the channel of the scan head connector to which the Z-axis is connected.

SCANLAB's XY2-100 converter can be used for iDRIVE scan systems equipped with a conventional XY2-100 or XY2-100 Enhanced interface. It converts the RTC5's SL2-100 control signals into XY2-100-conformant signals and the scan system's XY2-100 status signals into SL2-100-conformant signals (see [page 43](#)). This conversion supports the full functionality of iDRIVE technology on previous XY2-100-based scan systems.

SL2-100-equipped scan systems can only be controlled by an RTC5 without a converter.

8.1.2 Configuring Status Return Behavior

Selecting Data Signals

The digital servo architecture of iDRIVE scan systems allows a wide variety of data signals to be returned to the control board. Each axis is assigned its own status channel which transmits data to the controller board every 10 µs: The STATUS channel is provided for the X axis (galvanometer scanner 2) and the STATUS1 channel for the Y axis (galvanometer scanner 1). This opens up possibilities such as monitoring the galvanometer scanners' actual values during execution of an application or carrying out comprehensive troubleshooting in case of operational malfunction.

The **control_command** command allows selection of which data the scan head should return to the control board. The available data signals are described in detail in the manual of the respective scan system and in the **control_command** section of the command reference. The selected data sources are transmitted until another source is selected.

In addition to the above-mentioned "normal" status information, iDRIVE scan systems with SL2-100 interfaces (operated without an XY2-100 converter) can also simultaneously return additional slowly-changing status information (e.g. current temperatures of the galvanometer scanners). The command **send_user_data** allows specification of data that the scan system should return. With **send_user_data**, data transmission to the scan system takes place by a user data bit across multiple transmission cycles parallel to normal communication. In contrast to **control_command**, neither the transmission of position data nor the movements of the galvanometer scanners are interrupted (see also page 172). Likewise, the scan system's return of additional status information takes place by a user data bit. Here, neither the above-mentioned "normal" status returns nor the associated functions (e.g. **set_trigger/set_trigger4** or speed-dependent laser control) are adversely affected.

After every power-up or reset (starting with approx. five seconds after the power-up or reset), the scan system transmits (on all receive channels) the XY2-100 status word (and no information i.e. 0 by the user data bit).

Querying Data

Data received by the RTC5 (with the exception of user data bits) can be read asynchronously at any time by the commands **get_value**, **get_values** or **get_head_status** or synchronously by the command **set_trigger/set_trigger4** (see "**Status Monitoring and Diagnostics**", page 143 and the corresponding command references). Note that switching of the data source is followed by a short (serial transmission-related) delay before the first data is transmitted. Therefore, after switching data sources waiting times of approx. 50 µs can occur before reading the data.

The value ranges of transmitted data and the possible status states are described in the command reference of the **control_command** command.

Notes

- The **get_head_status** command queries the XY2-100 status word, which the scan system always returns in parallel with other status information, as per the SL2-100 protocol. In contrast, if the scan system is controlled by an XY2-100 converter, the XY2-100 status word is only returned when this data type is specifically selected (otherwise, **get_head_status** would return unusable data).

The scan system's status information returned by the user data bit (across multiple transmission cycles) can be queried by the RTC5 (in intervals of approx. 2 ms) with **read_user_data**. The value ranges of returned signals and the possible status states are described in the command reference of the **send_user_data** command and the user manual of the respective scan system. Due to the bitwise serial nature of data transmission and a possible synchronization time, a waiting period of at least 4-6 ms should be inserted between **send_user_data** and **read_user_data** to ensure reliable receipt of the requested type of new data.

8.1.3 Position Monitoring

For some applications, it is important to monitor scan system positioning even when processing, and, if necessary, with documentability.

For this purpose, the actual positions of the scan axes (hence the galvanometer scanners and/or the scan system's Z axis) must be selected to be returned from the scan system by the command **control_command**. The returned actual position values can be queried by **get_values** or recorded by **set_trigger/set_trigger4**⁽¹⁾.

If returned actual positions of the scan axes need to be compared with cartesian coordinate values (X, Y, Z) (which are the call parameters for RTC5 commands), then it must be additionally taken into account that these coordinate values generally – depending on the selected settings – undergo various corrections and transformations (e.g. an image field correction, a coordinate transformation and/or a Processing-on-the-fly correction) before they are outputted as control values to the scan system (see also [page 141](#)). To facilitate such comparability of the scan system's returned actual position values with original cartesian coordinate values (X, Y, Z), the RTC5 command set provides commands for backward transformation of position values.

For runtime reasons, the backward transformation needs to subsequently be performed by the PC rather than on the RTC5 Board itself. For this, all correction and transformation settings currently assigned to the scan system can be transferred from the RTC5 Board to the PC by the **upload_transform** command. Afterwards, an individual XY value pair or an individual Z value can be backward transformed by **transform**. With **get_transform** (see also **get_waveform**), you can backward transform an entire series of XY value pairs or Z values previously recorded by **set_trigger/set_trigger4**.

Notes

- Wobbel transformations, Processing-on-the-fly corrections and eventual clipping during forward transformation can never be backward transformed (see following table).

Forward transformation	Backward transformation possible
Wobbel motion	no
Processing-on-the-fly correction	no
Coordinate transformations (total matrix and offset) to the X and Y coordinates	yes
Offset to the Z coordinate	yes
Clipping to the edge of the positionable (20-bit real) image field range	no
Image field correction	yes
Gain and offset correction of automatic self-calibration	yes
Clipping to the edge of the maximum possible range of control values.	no

Likewise, backward transformation is not possible if a noninvertible transformation matrix was defined during forward transformation (note: clipping to ± 50 for each individual matrix coefficient while taking into account a factor defined by **set_scale**; see [page 183](#)). The non-invertability is already reported by the **upload_transform** command.

If forward transformation included a clipping to the edges of the positionable image field or to the edges of the maximum possible range of control values, then backward transformation (ideally) calculates the cartesian coordinates of these edge values instead of the original values.

(1) However, transfer times to/from the scan head cause the most recently returned actual position to lag the most recently outputted control signal by a few clock cycles.



- By **get_values**, four freely specifiable signals can be queried simultaneously (example: galvanometer 2's actual position by StatusAX, galvanometer 1's actual position by StatusAY, the actual Z-axis position by StatusBX, plus an additional desired signal such as LaserOn). In contrast, the **get_value** command (without the 's' in the command name) is not useful for monitoring XY positioning because it is only meant for querying a single signal and multiple calls unavoidably lead to XYZ values across different points of time.
- With **set_trigger**, you can simultaneously record two desired signals by two measurement channels (with **set_trigger4** four signals by four measurement channels).
- With **transform** and **get_transform**, you can use the parameter `Code` to specify that values queried by **get_values** or **set_trigger** are to be assigned to X, Y and Z for backward transformation.
- **transform** and **get_transform** also allow specification of which partial transformations should be performed.
- Values queried by **get_values**, or arbitrary synthetic values (e.g. measurement points of a scanned test pattern) can actually be backward transformed with **transform**. During backward transformation of synthetic values beyond the forward transformation's achievable image field (e.g. when test pattern measurements do not use SCANLAB's reference system, see [page 134](#)), values sometimes are only calculated by extrapolation, due to possible range violations or other errors.
- If the user program binarily stores both the recorded values and the transferred transformation data (see **get_waveform**), then subsequent backward transformation by **transform** (not **get_transform**) can also be executed offline, hence without needing to further access a RTC5 Board.
- If **control_command** is used to specify positioning error rather than actual position as the to be returned data type by the scan system, then it is not possible to directly compare the originally defined pattern with the scanned pattern. But you can check if the scan system correctly processed the RTC5 Board's output values. This is particularly useful if backward transformation of actual values is not (fully) possible or when it cannot be determined if deviations between backward-transformed actual positions and originally defined coordinate values are due to scan system error or clipping during forward transformation.

8.1.4 Configuring Dynamics Settings (Tuning)

SCANLAB can optimize the dynamics setting of scan systems (tuning) to accommodate differing requirements of diverse applications regarding the laser positioning dynamics – e.g. to execute vectors or circular arcs at a constant processing speed (vector tuning) or to execute jumps of minimized duration (jump tuning).

SCANLAB's iDRIVE scan systems can also be optionally equipped with multiple tunings (i.e. with several parameter sets or algorithms), thus allowing user selection of appropriate tunings matched to each of their various applications. You can select a tuning by **control_command**. For systems equipped with one or several jump tunings, you can also activate jump mode (and hereby tuning autoswitching) for 2D jumps (see [page 176](#)).

The default start behavior is for the scan system to start with its first tuning (number 0) upon power-up or after a reset.

8.1.5 Jump Mode

For applications such as drilling holes with defined spacing (whereby laser processing is actually point-by-point rather than along lines and curves), you can optimize process times by activating the so-called jump mode. For this, the scan system must be equipped with a jump tuning (see also [section "Requirements and Activation", page 177](#)).

Functional Principle

In the default setting (after [load_program_file](#)), both jump commands and marking commands execute in vector mode:

- The jump length gets subdivided into individually executable microvectors in accordance with the current jump speed. If the scan system is only equipped with a jump tuning, then the microvectors execute using this tuning.
- A subsequent list command is preceded by a jump delay specified by [set_scanner_delays](#).

In contrast, when jump mode is enabled and activated by the [set_jump_mode](#) or [set_jump_mode_list](#) command, every 2D jump (see below) executes as follows:

- The 2D jump's entire jump length executes as a so-called hard jump by a timed jump with a 10 µs execution time, i.e. the target position gets instantaneously outputted without microvectorization.
- The jump executes with a jump tuning (the scan system must be equipped with one). [set_jump_mode](#) can be used to designate which jump tuning to use. If a different tuning was set before the jump, then the RTC5 automatically switches at the beginning of the jump to the tuning specified by [set_jump_mode](#).
- At the end of the 2D jump, the RTC5 automatically switches to a vector tuning (if the scan system is equipped with one and if a corresponding setting was made by [set_jump_mode](#)).

- At the end of the 2D jump, a jump-length-dependent jump delay occurs. This jump delay can be specified for the corresponding jump length by **load_jump_table_offset** or **set_jump_table** (see also "Jump-Length-Dependent Jump Delays", page 178). Here, an external jump delay specified by **set_scanner_delays** is *not* taken into account.

Notes

- Jump mode works exclusively on
 - **jump_abs**, **jump_rel**, **goto_xy** (*not* on the corresponding 3D, para or timed commands)
 - home jumps and home returns (see **home_position**)
- If a 2D jump occurs where the jump length limit (`Length` parameter) specified by **set_jump_mode** is *not* reached or exceeded on at least one of the two axes, then the jump executes in vector mode even if jump mode was enabled and activated. This allows exploitation of the fact that *short* jumps can in some circumstances execute faster by vector tuning than with jump tuning. But if no vector tuning is installed or none specified, then you should set the `Length` parameter to 0.
- Each switch between different tunings (servos) requires an additional 10 µs clock period. For applications such as pure drilling, this can be avoided by not specifying a vector tuning to switch back to when you call **set_jump_mode**.
- When you deactivate or disable jump mode (by **set_jump_mode** or **set_jump_mode_list**), then subsequent jumps again execute in vector mode (microvectorized and without further servo autoswitching). Here, the vector tuning is used that was most recently set at the end of jump mode, unless deactivation was followed by selection of a different tuning by **control_command**. Moreover, the most recently set jump speed is again used and jumps are followed by the jump delay specified by **set_scanner_delays**.

Requirements and Activation

The following are required for enabling and activating jump mode:

- At least one of the two scan head connectors must have been assigned a correction table.
- At least one of the two scan head connectors must be connected to an intelliSCAN, intellicube, intelliWELD or intelliDRILL scan system.
- The software version of the attached scan system(s) must be 2078 or higher.
- The attached scan system must be equipped with at least one jump tuning. In contrast, a vector tuning is not absolutely required.
- The tuning numbers specified by **set_jump_mode** must match those stored on the board.
- The tunings specified by **set_jump_mode** must be of the proper type – vector tuning or jump tuning – (the tuning type is stored as info in the scan system's firmware) and must be suitable for rapid switching.

Before jump mode can be activated by **set_jump_mode_list**, it must have been successfully enabled at least once by **set_jump_mode** (see command description). The **set_jump_mode** control command (but not the **set_jump_mode_list** list command) performs an appropriate check if jump mode was not already enabled.



Jump-Length-Dependent Jump Delays

When executing a hard jump, it takes the scanner some time to reach the specified position. The RTC5 takes this delay (also called step response) into account by appending a jump delay at the end of the jump that appropriately postpones further program execution (point-by-point laser processing does not need to take other scanner delays into account and you can generally set laser delays to 0).

The specific step response behavior of the respective scan system (step response time vs. jump length) can be stored on the RTC5 in a user-specific jump delay table. With jump mode enabled, the RTC5 uses the specified jump delay table to determine the appropriate jump delay value for each 2D jump in accordance with the jump's longer edge (i.e. either the x or y component of the jump).

You can determine the step response behavior experimentally and then load it onto the board as a table of values using the `load_jump_table_offset` command. Alternatively, the jump delay table can also be automatically determined by `load_jump_table_offset` (parameter Name = 0). Additionally, the currently loaded jump delay table can be retrieved as a binary table by `get_jump_table` and reloaded onto the board by `set_jump_table`.

The step response time (at least for longer jumps) typically scales with the squareroot of the jump length, and `load_program_file` accordingly initializes the internal jump table – with an end value of 10.24 ms for a jump length of 2^{20} bits.

Experimental Determination of Jump Delay Values

The scan system's user manual typically specifies the step response times for each jump tuning at selected jump lengths.

To experimentally determine the step response behavior, you need to have the scan system perform jumps of various lengths and query the resulting position values by the status channel for analysis.

After you activate jump mode, perform the jumps by using `jump_abs` or `jump_rel` commands (and the desired jump tuning). The scan system should have been previously set to return the actual-position data type by `control_command`. You can then record the latter by `set_trigger/set_trigger4` and retrieve it by `get_waveform`.

The determined jump delay values must be supplied in an ASCII text file. If the step response behaviors of both axes differ, then the higher of the two axes' jump delay values should be supplied.

Notes on Loading Determined Jump Delay Values

The **load_jump_table_offset** command loads a table with jump lengths and jump delay values from an ASCII text file. The text file can contain one or several tables.

Each table can contain up to 50 data points (*Length | Delay(Length)*) for various jump lengths. The RTC5 applies linear interpolation to this data to create the complete (internal) jump delay table *Delay(Length)*.

The following rules apply to tables:

- Each table must begin with the statement (Caption)
`[JumpTable<No>]`
 where *<No>* must be replaced by a nonnegative integer which denotes the table number.
- If the table contains multiple `[JumpTable<No>]` entries with the same *<No>*, then only the instructions after the first entry is used; instructions that follow further entries are ignored. Only instructions up to the next '[' character (that is not preceded by a semicolon) are used.
- Each data point (*Length | Delay(Length)*) is described by two instructions:
`Length<n> = <LengthValue>`
`Delay<n> = <DelayValue>`
 where *<n>* must be replaced by an integer ($1 \leq <n> \leq 50$) which denotes the number of the data point. The *<Value>* numbers can be supplied as (unsigned) floating point numbers. Use the period (.) as the decimal separator.
- If the table contains multiple data points with the same index *<n>*, then the most recently read one is used and the previous ones ignored.
- If the table contains multiple data points with the same jump length value *Length*, then the data point with the largest index *<n>* is used and the others ignored. Equality is checked to within ± 0.01 .
- For *<Value>* the following ranges apply:
 $0.0 \leq Length \leq 1048576.0$
 $0.0 \leq Delay(Length) \leq 65535.0$
 Delay values are supplied in units of 10 µs, jump lengths in bits.
- Each instruction must be in a separate line.

- Space characters and tabs within a line (e.g. between '=' and *<Value>*) are ignored.
- Empty lines are ignored.
- Data points with invalid values are ignored.
- The data point of a particular index *<n>* is ignored if the corresponding *Length<n>* and/or *Delay<n>* definition is missing.
- The semicolon ';' can be used for comments. All characters in a line following a semicolon are ignored.
- The instructions for data points in the table can be ordered as desired.
- Indices for data point pairs in the table can be selected as desired within the range [1...50] (the table is then automatically sorted by ascending position values).
- If the table contains no valid data points, then the command **load_jump_table_offset** has no effect (return value 1 or 13).
- If there is no entry of *Length* = 0.0, then one with *Delay* = *Min(Delay<i>)* is inserted (the smallest valid value encountered is filled downward). The same applies to *Length* = 524288.0 with *Max(Delay<i>)*.
- If the specified text file contains only one valid data point with *Delay<n>* = D, then the jump delay table *Delay(Length)* = D (for the whole jump length range) is loaded.

During initialization of the RTC5 (by **load_program_file**), the internal jump delay table is initialized with a squareroot curve and with a jump delay end value of 1024 (i.e. 10.24 ms) for a jump length of 2^{20} bits.

Automatic Determination of the Jump Delay Table

You can initiate automatic determination of the jump delay table by **load_jump_table_offset** (parameter Name = 0) if you had previously enabled and activated jump mode successfully by **set_jump_mode** (Flag = 1).

For automatic determination, automatic laser control is deactivated if necessary, the data type returned by the scan system is set to target position and the tuning set to the jump tuning that had been defined by **set_jump_mode** (the original settings are, if necessary, restored when the command completes).

For automatic determination, several diagonal jumps of varying lengths are performed. For each jump, the target position returned by the scan system is recorded by **set_trigger/set_trigger4** and retrieved by **get_waveform**. The data is analyzed for the time-point at which the specified position tolerance (**PosAck** parameter) was last exceeded, i.e. when the target position persistently remained in the jump target positional range \pm **PosAck**. This value is then reserved as the jump delay value associated with the corresponding jump length.

Notes

- Before automatic determination, you must absolutely switch off the laser.
- Data recording requires execution of a list with a total of six commands. The commands are automatically written to list memory and processed. The parameter **ListPos** indicates the position in list memory ("list 1" or "list 2") in which storage is to occur. This position should be such that any previously entered list commands can be harmlessly overwritten.

- For automatic determination, the longest jump is performed first, followed by increasingly shorter jumps (with a maximum of up to 16 different jump lengths). For the first (longest) jump length (jump across the entire image diagonal), the measurement period is specified by the parameter **MaxDelay** [10 μ s]. For subsequent (shorter) jumps, it is defined by each previously determined jump delay. **MaxDelay** should be chosen to be adequate but not significantly larger than the jump delay for the longest jump. A larger **MaxDelay** increases the total required execution time. With **MaxDelay** = 500, the total execution time for automatic determination is typically a few seconds.
- For statistical noise reduction, four identical jumps are performed for each jump length and the results averaged. Additionally, the values for each individual measurement are low-pass filtered (2-point smoothing). This permits selection of a position tolerance **PosAck** that can also be somewhat (but not substantially) under the expected noise level (but note that XY2-100 converters only return whole multiples of 16. Here, a noise level of $\pm 3 \times 16$ bits can be expected).
- If the **PosAck** range is not persistently reached within the measurement period, then **MaxDelay** becomes the determined jump delay.
- If the determined jump delay is smaller than **MinDelay**, then **MinDelay** becomes the determined jump delay and the measurement terminates. Then, shorter jumps are no longer performed and **MinDelay** is also the determined jump delay for these shorter jump lengths. A longer **MinDelay** reduces the total execution time for automatic determination.

- If an offset is specified for automatic determination (by the according `load_jump_table_offset` parameter), this offset is added to all automatically determined delay values before the overall jump delay table gets calculated by linear interpolation and loaded onto the board (in addition, the delay values are clipped to the value range 0...65535). The Offset can be used to compensate for measurement runtime latencies (e.g. caused by an XY2-100 converter, by tuning switching or by a runtime latency of the returned signal) when calculating the jump delay table. It can also be used to add a safety margin to the delay values to compensate for noise-induced random deviations.
- The `load_jump_table` command is identical to `load_jump_table_offset` with `Offset = 0`.
- For simultaneous control of two scan systems, you should determine the jump delay values for both systems and, after comparing, use the values of the slower system.
 - The resulting table can be retrieved in binary form by `get_jump_table` and reloaded onto the board by `set_jump_table`.

8.1.6 Configuring the PosAcknowledge Threshold Value

The `control_command` command can also be used to set the PosAcknowledge threshold value. The default start behavior is for the scan system to set the threshold value to 0.28% of the full position range after every power-up or reset. If other threshold values are desired, they must be separately set for each axis.

8.1.7 Configuring the Effective Calibration

The servo electronic can also be configured by the `control_command` to (down) scale the position values received from the RTC5 by a specific factor (1, 1/2, 1/4 or 1/8). The position signals (optionally) returned by the scan systems to the RTC5 remain unaffected, as do the pre-configured calibrations of SCANLAB's scan systems. However, the effective calibration can be thereby reduced to confine the scan area to a smaller angular range – with a higher angular resolution.

The default start behavior is for the scan system to start with a scale factor of 1 (i.e. with SCANLAB's pre-configured calibration) upon power-up or after a reset.

With the `control_command`, you can also query SCANLAB's pre-configured calibration and the currently selected scale factor.

8.1.8 Configuring the Start Behavior

In their default configuration the iDRI^E scan systems are pre-configured by SCANLAB so that

- the first tuning (number 0) is selected (see also [section "Configuring Dynamics Settings \(Tuning\)", page 176](#)),
- a PosAcknowledge threshold value of 0.28% of the full position range (i.e. 0.28% of 2^{16} bits) is set (see also [section "Configuring the PosAcknowledge Threshold Value", page 181](#)),
- a scale factor of 1 is set (see also [section "Configuring the Effective Calibration", page 181](#)).

The listed settings can be changed by the **control_command**. The changed settings are only temporary, however they can be additionally saved as starting settings for subsequent power-ups or resets (power supply switched off and switched on) with **control_command(..., ..., 0A00H)**.

As long as the start behavior is not changed as described, the scan head starts with the starting settings pre-configured by SCANLAB on every power-up or reset.

The status return behavior of the scan system can only be temporarily changed. The corresponding start behavior is fixed by SCANLAB: after every restart, the scan system transmits the XY2-100 status word (see also [section "Configuring Status Return Behavior", page 173](#)).

8.1.9 Fault Diagnosis and Functional Test

If a problem occurs, the versatile status return functions of the iDRI^E scan system can be used for scan system diagnosis, too. These functions allow to read for instance

- the current operating state
- the operating state at the moment of the most recently occurred operation interruption and
- an event code, indicating which particular event caused the scan head to enter a temporary or permanent error state.

To verify that data transfer capability between the RTC5 and a scan system is intact, **control_command** can transmit an 8-bit value – separately for each axis – to the scan system. Subsequently, a 20-bit value is returned on the corresponding status channel: If data transfer is error-free, then the upper 8 bits of the returned 20-bit value is identical with the originally sent 8-bit value, and the next lower 8 bits are identical with the complement of the sent 8-bit value.

These 20-bit values are returned until the **control_command** is used to select another return data type (see [section "Configuring Status Return Behavior", page 173](#)).

To facilitate – after a data transfer verification – restoration of the status return behavior in effect prior to the data transfer verification, the **control_command** allows the prior data type to be temporarily stored for later retrieval.

8.2 Coordinate Transformations

For precise set-up of the scan system relative to the image field (if the “second scan head control” option has been enabled, *two* scan heads can be adjusted relative to a *common* image field), a linear coordinate transformation can be defined (separately for the primary and secondary scan head connectors) for all X and Y output coordinates ($x|y$) defined by vector or arc commands:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = M \bullet \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}$$

The (2×2) total matrix M is thereby automatically calculated by the RTC5 as a product of a scaling matrix M_S , a rotation matrix M_R and a general transformation matrix M_T :

$$M = M_T \bullet M_R \bullet M_S$$

The coefficients of the three matrices (M_T , M_R , and M_S) and the offset values ($x_0|y_0$) can be individually defined for the primary and secondary scan head connector.

The offset ($x_0|y_0$) is set by `set_offset` or `set_offset_list`.

For 3-axis scan systems, `set_offset_xyz` or `set_offset_xyz_list` enables setting of an offset z_0 for the Z coordinate, too (z_0 has the opposite effect of `set_defocus` or `set_defocus_list`). The following applies:

$$z' = z + z_0$$

The coefficients of the scaling matrix M_S are set by `set_scale` or `set_scale_list` using a scaling factor k that is common to both axes:

$$M_S = \begin{bmatrix} k & 0 \\ 0 & k \end{bmatrix}$$

The coefficients of the rotation matrix M_R are set by `set_angle` or `set_angle_list` by specifying a rotation angle α (mathematical definition: positive angles produce counterclockwise rotation):

$$M_R = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix}$$

The coefficients $m_{11} \dots m_{22}$ of the general transformation matrix M_T are set by `set_matrix` or `set_matrix_list`:

$$M_T = \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix}$$

With the general transformation matrix M_T , the two above matrices (M_S and M_R , as special case) as well as further transformations for scaling, rotating, mirroring or skewing objects can be defined:

- Scaling by the factors k_X and k_Y :

$$M_T = \begin{bmatrix} k_X & 0 \\ 0 & k_Y \end{bmatrix}$$

- Rotation by the angle α :

$$M_T = \begin{bmatrix} \cos \alpha & -\sin \alpha \\ \sin \alpha & \cos \alpha \end{bmatrix}$$

For instance, the command

`set_matrix(1, 0.5, -0.866, 0.866, 0.5, 1)`
 defines – for the primary scan head connector – an immediate rotation by 60° (counterclockwise) around the center of the image field. This can also be achieved by the command `set_angle(1, 60)`.

- Mirroring around the Y axis (flipping in the X direction):

$$M_T = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$

- Mirroring around the X axis (flipping in the Y direction):

$$M_T = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

- Mirroring around the first dimension diagonal (exchanging the X and Y coordinates):

$$M_T = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

- Skewing in the X direction by the angle α (slanting):

$$M_T = \begin{bmatrix} 1 & -\sin \alpha \\ 0 & 1 \end{bmatrix}$$

Example: `set_matrix(1, 1, -0.25, 0, 1, 0)`

A general transformation defined by `set_matrix` or `set_matrix_list` can also represent a combination of various transformations (users can calculate the corresponding matrix M_T by multiplying the corresponding individual matrices in the correct order).

Notes

- The described coordinate transformations are primarily intended for small corrections when setting up the scan system relative to the image field. Separate settings for scaling and rotations thereby provide more handling flexibility in comparison to a single matrix setting.
- Initialization (by `load_program_file`) results in an offset of $(0|0|0)$ and in matrices M_S , M_R and M_T , each predefined as identity matrices.
- Each matrix or offset definition overwrites prior definitions.
- The RTC5 calculates the total matrix M independently of the order in which the offset and individual transformation matrices were defined.
- The value range of scaling factor k for the scaling matrix M_S is $[-16 \dots +16]$. The value range for the coefficients of the general transformation matrix M_T is $[-50 \dots +50]$. Also be sure that the value range $[-50 \dots +50]$ for individual coefficients of the total matrix M are not exceeded; otherwise calculation of the corrected coordinates might result, under some circumstances, in overflows.
- Rotations are exclusively around the centerpoint of the image field; mirroring is relative to the axes. It is the responsibility of the users to calculate and make available suitable offset translations.
- For each definition, the parameter `at_once` can be used to specify whether the new setting should have immediate effect on the current position (`at_once = 1` or `3`) or whether it should only be provisionally collected and intermediately stored (`at_once = 0` or `2`). For collection by list commands, the commands do not necessarily need to directly follow each other. The most recently issued `at_once` parameter value determines when a (collected) transformation takes effect.
- Before the (possibly collected) total transformation is applied to the current position, the “laser active” laser control signals with `at_once = 0` to `2` are switched off; but with `at_once = 3` the laser control signals remain unaffected.

- With `at_once = 1` or `3`, all settings collected until then (by control commands or list commands) are processed immediately and simultaneously. Then – even before the subsequent command is executed – the scan system's axes are moved from the current position to the corrected position at the (pre)defined jump speed (to avoid hard jumps). Consequently, this can require some clock cycles. Any scanner delays are not initialized. The INTERNAL-BUSY status is set while the jump to the corrected position is executed.
- With `at_once = 0`, a setting made by a control command (but not by a list command) automatically takes effect (together with all transformation settings that were collected and intermediately stored until then) upon execution of the next list command (essentially immediately if a list is currently processed; in this case, `at_once = 0` is handled like `at_once = 1`). Otherwise, all settings defined with `at_once = 0` (together with hitherto collected settings) remain without effect – even when a `goto_xy` or `goto_xyz` command is subsequently executed – as long as they are not activated by a subsequent coordinate transformation (e.g. by a list command with `at_once = 1` or `3` or a corresponding control command).

- With `at_once = 2`, the settings (collected by control or list commands) only become effective upon execution of the next `jump_abs`, `jump_rel`, `goto_xy` or `goto_xyz` command (but not other jump commands such as `jump_abs_3d` or `jump_rel_3d`). Correction of the current output position then occurs together with the specified coordinate jump instead of separately (unlike `at_once = 0, 1` or `3`). This eliminates unnecessary galvanometer scanner motions (incl. possible delays) and shortens runtime.

Example: The following command list produces a first jump to `(0, 0)`, followed by – if `at_once = 1` or `at_once = 3` – a second jump from `(0, 0)` to `(1000, 500)` and a third from `(1000, 500)` to `(0, 500)`. But if `at_once = 2`, then only a second jump occurs from `(0, 0)` to `(0, 500)`.

```
jump_abs(0, 0);
set_offset_list(1000, 500, at_once);
jump_abs(-1000, 0);
```

- If no correction table has been previously assigned to the corresponding scan head connector, then the new settings for the coordinate transformations are only stored on the RTC5 (even when `at_once = 1` or `3`) and only takes effect when a correction table is assigned.
- If the “second scan head control” option has not been enabled, coordinate transformations specified for the secondary scan head control have no effect.
- Coordinate transformations are applied to the output coordinates of all vector commands (jump or mark list commands, but also `goto_xy` or `goto_xyz` commands) and arc commands. The transformation only becomes effective after microvectorization and after incorporation of any wobbel or Processing-on-the-fly settings, but before image field correction. The matrix transformation is thereby executed before the offset translation.

- The coordinate transformation enables, for example, scalable marking. When scaling, take into account that the resulting effective jump or marking speed changes (even direction-dependently if the scaling factors k_x and k_y differ). Under some circumstances, the jump and marking speeds must therefore be suitably adjusted.
- For 3D vector commands, the transformation generally only affects the X and Y components. Only the offset affects all three components.
- For utilizing the complete real image field even if a coordinate transformation (as rotation, shrinkage or shift) is activated, the extended value range of the virtual 24-bit image field can be used (see [page 135](#)).
- For 2D Processing-on-the-fly applications, you can additionally define coordinate transformations for the entire virtual image field. The RTC5 applies such coordinate transformations to the already microvectorized coordinates, but – unlike the previously described head-specific transformations – already *before* Processing-on-the-fly correction. See the section “Coordinate Transformations in the Virtual Image Field”, page 208.

Notes for RTC4 Users

- For loading RTC5 correction tables by **load_correction_file**, transformation of the correction tables is *not* settable.
- For the RTC5, coordinate transformations can be separately defined for the primary and secondary scan head connectors. This fulfills the same purpose as the transformations defined by the RTC4’s **load_correction_file** command.
- The RTC5 consequently first executes coordinate transformations *after* microvectorization, but directly *before* image field correction (with corresponding consequences for the effective jump or marking speed, as previously mentioned).
- The RTC5 does not support the RTC4’s coordinate transformations (collectively for both scan heads) before microvectorization.

8.3 Online Positioning

The preceding chapter 8.2 "Coordinate Transformations", page 183 details how to precisely align a scan system relative to the image field. The user program can, for example, determine the required transformation values by automatic position analysis for a workpiece on a conveyor belt and then execute the associated transformations.

However, it is not easy to achieve well-controlled timing (referenced to the RTC5's 10 µs clock) while positioning a workpiece and aligning the scan system by the (control) commands described in chapter 8.2 "Coordinate Transformations", page 183. For applications in which such timing is important, the RTC5 command set also includes commands for so-called online positioning. Here, data for an offset and/or rotation coordinate transformation or a general matrix operation (e.g. directly from position analysis of the workpiece) can be inputted by the McBSP/SPI interface.

The reading of online positioning data by the McBSP/SPI interface needs to be activated and configured as desired with the commands `set_mcbsp_x`, `set_mcbsp_y` and/or `set_mcbsp_rot` or `set_mcbsp_matrix` (or by the equivalent list commands) (see below). With `apply_mcbsp` or `apply_mcbsp_list`, you can acquire the most recent fully transferred values and define the required coordinate transformations (as with `set_offset` and/or `set_angle` or `set_matrix` or the equivalent list commands). Here, as with the commands described in chapter 8.2 "Coordinate Transformations", page 183 an `at_once` parameter can be used to specify when the newly defined (total) transformation should take effect.

For precise timing, execution of the list command that triggers the transformation (depending on the `at_once` parameter, this would be `apply_mcbsp_list` or `jump_abs` or `jump_rel` or any other list command) can be made dependent on the input of an external control signal (for conditional command execution, see page 244).

The McBSP/SPI interface is described on chapter 4.4.6 "SPI / I2C Socket Connector", page 54.

Configuring Online Positioning

The commands `set_mcbsp_x`, `set_mcbsp_y` and/or `set_mcbsp_rot` (or the equivalent list commands) determine both how the values inputted at the McBSP/SPI interface are interpreted by the RTC5 and which internal memory location is used to read the values:

- Depending on which of the above commands is called, the RTC5 interprets the inputted values as offsets in the X and/or Y directions and/or as rotation values or as matrix coefficients. The desired scaling factor always needs to be supplied as a command parameter (except with `set_mcbsp_matrix`, see command description). The three options `x,y` and `rot` can be used either separately or in any desired combination. By the appropriate command, each option can be enabled or disabled independently of the other two. In contrast, the `matrix` option cannot be used in conjunction with other options.
- As soon as one of the four options becomes activated, all values subsequently inputted at the McBSP/SPI interface are internally stored in memory location 1 or possibly memory location 2 (see below). Transferred values can subsequently be queried by `read_mcbsp` or applied in coordinate transformations by `apply_mcbsp` or `apply_mcbsp_list`.
- **x, y or rot**
If you activate only one of the three options, then X or Y offset correction values can be supplied as *signed* 32-bit values or rotation correction values as *unsigned* 32-bit values.
The McBSP/SPI input values are transferred to internal memory location 1.
- **x and y (without rot)**
If you activate X and Y offset corrections, but no rotation correction, then the two offset correction values must be supplied as a 16-bit signed value, each, combined to a 32-bit value (the X value in the lower 16 bits and the Y value in the upper 16 bits).
The McBSP/SPI input values are transferred to internal memory location 1.

- **x or y and rot**

If you activate an X or a Y offset correction together with a rotation correction, then the offset and rotation correction values should be alternatingly supplied as 32-bit values. The McBSP/SPI input values are then alternatingly transferred to internal memory locations 1 and 2. The RTC5 identifies the data type by examining the coding bit #31 (bit #31 = 0 for offset values, bit #31 = 1 for rotation correction values). Signed 31 bits are effectively available for transferring offset values. 31 bits *without* sign are available for rotation correction values.

- **x, y and rot**

If you activate all three options together, then the two offset correction values must be combined and supplied as one 32-bit value alternatingly supplied with the rotation correction value as a second 32-bit value. The McBSP/SPI input values are likewise alternatingly transferred to internal memory locations 1 and 2. The RTC5 identifies the data type by examining the coding bit #31 (Bit #31 = 0 for offset values, Bit #31 = 1 for rotation correction values). Signed 15 bits are effectively available for transferring offset values (whereby X values reside in the lower 16 bits and Y values in the upper 16 bits). 31 bits *without* sign are available for rotation correction values.

- The last two cases designate the data type by coding bit #31. Though this makes the order of transmission irrelevant, always *both* data types nevertheless must be transmitted (preferably always alternatingly). If a request is made by **apply_mcbsp** (or **apply_mcbsp_list**) when two values of the same data type exist in both memory locations, then the most recently transferred value is always used. If two identical Bit #31 codings are present, then the last transfer should have already ended at the time of the request.

- **matrix**

With this option activated, matrix coefficients are then transferable as 32-bit signed values. The indices are encoded in the data word (see command description). McBSP/SPI input values get transferred to internal memory location 1.

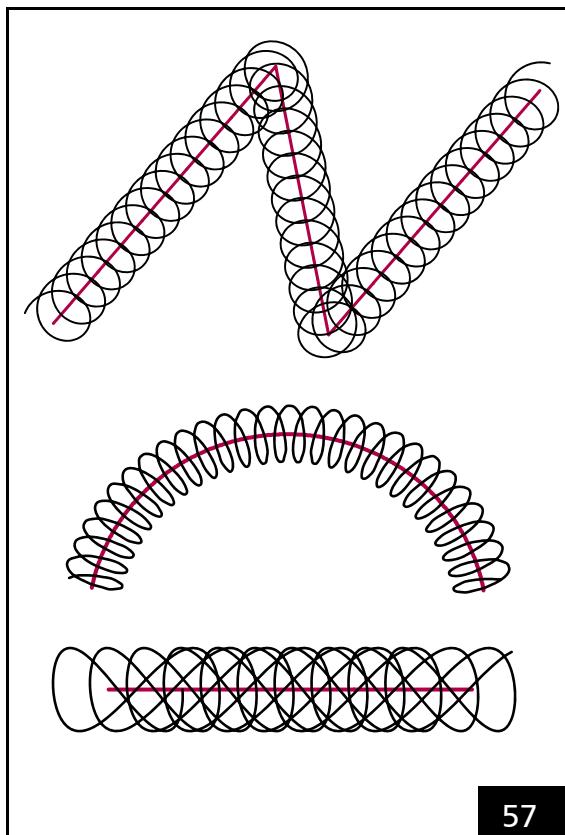
Notes

- You can use online positioning in conjunction with an encoder-controlled Processing-on-the-fly application, but *not* in conjunction with a Processing-on-the-fly application controlled by McBSP/SPI signals:
 - When you use the commands for configuring online positioning, then Processing-on-the-fly correction activated by **set_fly_x_pos**, **set_fly_y_pos**, **set_fly_rot_pos**, **set_mcbsp_in**, **set_mcbsp_in_list**, **set_multi_mcbsp_in** or **set_multi_mcbsp_in_list** gets automatically deactivated. Subsequently, McBSP/SPI input values are copied to internal memory locations 1 and possibly 2 (see above) and are then available for online positioning, but no longer for a Processing-on-the-fly application.
 - In reverse, **set_fly_x_pos**, **set_fly_y_pos**, **set_fly_rot_pos**, **set_mcbsp_in**, **set_mcbsp_in_list**, **set_multi_mcbsp_in** or **set_multi_mcbsp_in_list** deactivate a previously activated online positioning. Subsequently, McBSP/SPI input values are then copied possibly instead of or additionally to the internal memory location 0 and/or 3 and are then available for the Processing-on-the-fly application, but no longer for online positioning.
 - If you switch off online positioning by **set_mcbsp_x**, **set_mcbsp_y** or **set_mcbsp_rot**, then the data is continued to be copied to internal memory locations 1 or 1 and 2 (as long as data is transmitted), but it is no longer applied (**apply_mcbsp** has no effect).

8.4 Wobble Mode

The wobble mode allows varying the line width for laser marking.

For this purpose, an ellipse-shaped motion is added to the regular, linear movement of the output position. This results in a spiral movement of the laser focus in the image field, see [figure 57](#). Alternatively, a combination with a figure-of-8 (horizontal or vertical to the direction of motion) can be activated.



57

Principle of the wobble mode. Top: circular wobble. Middle: ellipse-shaped wobble. Bottom: figure-of-8 wobble (horizontal 8).

A broadening of the original line is obtained by choosing suitable values for the transverse and longitudinal amplitudes and the frequency of the wobble movement. With figure-of-8s, broader mid-line processing can be achieved by appropriate parameter values. If the specified transverse and longitudinal amplitudes are identical, then the wobble shape remains stationary in space; otherwise the orientation of the wobble shape follows the current direction of motion.

As of version DLL 543, OUT 543, RBF 524 the present wobble amplitude (from [set_wobble](#) or [set_wobble_mode](#)) can be recorded with trigger signal 53 (see [set_trigger](#) or [set_trigger4](#)). The format of the data is ((transversal << 16) + longitudinal).

During Processing-on-the-fly correction by the McBSP/SPI or encoder interfaces where the scan head only compensates differences between actual external movements and intended total motion (see [chapter 8.7.2 "Compensation of Linear Movements"](#), [page 201](#), a slight jitter in the direction of galvanometer motion might occur, particularly during exact external path motions. Here, the wobble movement superimposed onto the direction of motion does correspondingly jitter, too. You can avoid such jitter by specifying a fixed (instead of the momentary) direction of motion for the wobble shape by the [set_wobble_direction](#) list command.

For optimum marking results, the wobble frequency should be appropriate for the specified marking speed. In some cases it can be useful to adjust the marking speed when the wobble mode is used.

After [set_wobble](#) or [set_wobble_mode](#), the wobble start point is always set for the same value relative to the vector/arc startpoint and direction. The Wobble phase is then continued both within an uninterrupted polyline (including arcs) and after interruptions (e.g. a jump command) until [set_wobble](#) or [set_wobble_mode](#) are called again.

The wobble mode cannot be combined with:

- Sky Writing
- Pixel output mode
- Jumps
- [laser_on_list](#)

For further details, see [set_wobble](#) and [set_wobble_mode](#).

For many welding applications, the default set of "classic" wobble shapes (circle, ellipse, sine, figure-8) does not produce optimal (smooth) results in the area of the weld seam. With ellipses, for example, high-speed motion occurs parallel to translation movements and low-speed motion occurs in the opposing direction, resulting in uneven energy deposition. [set_wobble_vector](#) lets you define a wobble shape customized for your user program, consisting of up to 512 piecewise linear sections, while also specifying variation of laser power along that shape

(see also [set_wobble_control](#)). However, [set_wobble_vector](#) cannot be used together with the Softstart function nor with automatic or vector-based laser control.

8.4.1 Wobble Shapes – Important Notes on Choosing Appropriate Parameter Values

“Classic” Wobble Shapes

“Classic” wobble shapes are defined using the [set_wobble_mode](#) or [set_wobble](#) command. Only assign hardware appropriate values to the Transversal, Longitudinal and Freq parameters.



Caution!

- Too big values define control situations where very high waste heat is produced. Galvanometer and digital control board/amplifier board overheating and permanent damage may occur even in short-term operation (overload!). Take the highest possible dynamics of scan head and laser into account.
- If the frequency values are too high the galvanometers may not be able to follow the nominal curve. This may lead to unexpected marking results.

Rule of thumb to estimate appropriate maximum values:

(1) Maximum frequency: $F = 1/(10 \times T)$

where T = tracking error⁽¹⁾

(2) Wobble amplitude⁽²⁾: $A = T \times V$

where V = typical positioning speed⁽¹⁾

Notes

- Also take the path velocity on the wobble shape itself into account. For circular wobble shapes it is calculated as follows:

(3) Path velocity $V_{\text{Path}} = 2 \times \pi \times A \times F$.

- Make sure that the combination of the used values and the trajectory velocity are suitable for a long-term operation without causing damages (process safety!).
- Check the temperature status already during an evaluation period (see [get_head_status](#)) in order to recognize potential overload situations at an early stage. With iDRIVE systems you can also read-out the present temperatures of galvanometers and/or digital control boards (see [control_command](#)).

Example of Use

System Specification

- Tracking error $T = 0.33$ ms.
- Calibration $\pm 0.349 \text{ rad}_{\text{optical}}$
 $(= 10^\circ \text{ mechanical})$ at $\pm 503,316$ bit.
– This is an angle control AC of
 $\approx 1.44 \times 10^6 \text{ bit/rad}_{\text{optical}}$
 $(\approx 50,300 \text{ bit}^\circ \text{ mechanical})$.
- Calibration factor⁽³⁾ $K = 5,000 \text{ bit/mm}$.
- Typical positioning speed
 $V_{\text{rad}} = 100 \text{ rad}_{\text{optical}}/\text{s}$.
– Converted to control values this corresponds to a typical positioning speed of
 $V = V_{\text{rad}} \times AC$
 $\approx 1.44 \times 10^8 \text{ bit/s} = 1,440 \text{ bit}/10 \mu\text{s}$.
- In the image field this corresponds to a typical positioning speed of $V/K = 28.8 \text{ m/s}$.

(1) See technical specifications in the scan head manual

(2) At the maximum frequency estimated with (1)

(3) See ReadMe file of correction file or [get_table_para](#)

Wobbel Parameters

- The maximum frequency estimated with rule of thumb (1):
 $F = 1/(10 \times 0.33 \times 10^{-3} \text{ s}) \approx 300 \text{ Hz.}$
 - Wobbel amplitude estimated with rule of thumb (2):
 $A = 0.33 \times 10^{-3} \text{ s} \times 1.44 \times 10^8 \text{ bit/s}$
 $\approx 47,500 \text{ bit.}$
 - In the image field this corresponds to a wobbel amplitude of $A/K \approx 9.5 \text{ mm.}$
 - Path velocity of circular wobbel shapes as given by rule of thumb (3) is:
 $V_{\text{Path}} \approx 895 \text{ bit}/10 \mu\text{s.}$
 - The *maximum marking speed* results from V_{Path}/K (in this example $89,500,000 \text{ bit/s} / 5,000,000 \text{ bit/m} \approx 17.9 \text{ m/s.}$ Note that this value is a rough estimate. Furthermore, it is dependent on a position due to the correction file (which is non-linear).
The calculated path velocity for wobbel only (!)
 - shall be lower than the specified maximum positioning speed,
 - but there may be certain circumstances where it is much higher than the *typical marking speed.*
- Make sure that your values are suitable for operation (temperature status and so on, same as above).

"Freely Definable Wobbel Shapes"

When defining "freely definable wobbel shapes" (`set_wobbel_vector` command) take the dynamics of the scan head into account. As with "classic" wobbel shapes (see above) the maximum positioning speed may not exceed significantly. That is, every individual microvector of the "freely definable wobbel shape" may not significantly exceed a duration of $10 \mu\text{s}$ times maximum positioning speed. Otherwise, a galvanometer overheating may occur.



8.5 Using Several Different Correction Tables

8.5.1 Configuration with Two 2D Scan Systems

The RTC5 can store four different correction tables at the same time. For two 2D scan systems controlled from a single RTC5 Board, each scan system can thereby receive its own separate image field correction.

The two desired correction tables can be assigned to either of the two scan head control ports by the command **select_cor_table** or **select_cor_table_list** (see also [page 43](#)).

The following steps describe how to use two correction tables in a double scan system configuration:

- ▶ Load each of the desired 2D correction files with the command

```
load_correction_file(Name, n, 2)  
(n = 1...4).
```

Refer to the command **load_correction_file** for details.
- ▶ Afterwards, you have to assign a loaded 2D-correction table to the first and the second scan head each by calling the command

```
select_cor_table(n1, n2)
```

with (n1 and n2 = 1...4).
- ▶ Use **set_matrix**, **set_offset**, **set_scale**, **set_angle** or the corresponding list commands to specify additional gain, rotation and offsets to align the two image fields precisely with respect to each other.

Notes

- The default setting for **select_cor_table** and **select_cor_table_list** is (1, 0), i.e. correction table #1 is used for the first scan head. The output signals for the second scan head are turned off.
- The RTC5 returns to this setting after every reset (by **load_program_file**). If a different setting is to be used, the command **select_cor_table** or **select_cor_table_list** must be called again after the reset. **load_program_file** deletes correction tables number 3 and 4.
- The scan head control ports *cannot* be simultaneously assigned two 3D correction tables, nor can they be simultaneously assigned both a 2D and a 3D correction table.

8.5.2 Using Several Correction Tables with a Single Scan System

It can also be useful to work with several different correction tables in a *single* scan system, e.g. if a pointer laser and a main laser with a different wavelength are used. Specialized applications sometimes also require different correction files quickly in succession. Because no correction files can be loaded during list processing, downloading between processing would result in non-productive time expenditure. If the RTC5's 3D option has been enabled, then up to four 3D correction tables, or a mix of 2D and 3D correction tables, can be loaded into its memory.

- ▶ Use **load_correction_file** to load the correction files.
- ▶ Afterward, use the commands

```
select_cor_table(n, 0) (n = 1...4)
```

to switch from one correction table to the other.

8.6 Controlling a 3-Axis Scan System (Optional)

If the 3D option of the RTC5 is enabled, then the RTC5 can also be used for controlling a 3-axis scan system consisting of an XY scan head and a Z-axis dynamic focus unit (varioSCAN).

8.6.1 Intended Use

In a 3-axis scan system, the mirrors of the XY scan head deflect the laser beam to the desired XY position and the Z-axis simultaneously adjusts the focal length of the scan system and thus the laser focus along the optical axis to the desired Z position.

3-axis scan systems can be used for positioning the laser focus within a flat processing field without the need for a flat field objective. This is often the case in applications for which standard flat field objectives are unavailable.

3-axis scan systems can also be used as 3D beam deflection systems. Here, the laser focus is guided along the contour of the workpiece being processed, thus enabling processing in three dimensions.

SCANLAB offers varioSCAN, varioSCAN_{de}, varioSCAN FC and varioSCAN FLEX dynamic focusing units to extend XY scan heads to 3-axis scan systems.

8.6.2 Connection and Initialization

To control a 3-axis scan system by an RTC5:

- the 3D option of the RTC5 must be enabled (for an installed RTC5, this can be checked by the command `get_RTC_version`),
- the XY scan head must be attached to the primary scan head connector and the Z axis to the secondary scan head connector⁽¹⁾ and
- a 3D correction table (D3_*.CT5) must be assigned by `select_cor_table` or `select_cor_table_list` to the primary scan head connector⁽¹⁾.

Other than that, no additional drivers or software files (e.g. further DLL or DSP program files) are needed for controlling a 3-axis scan system. Except for the 3D correction table (see above), initialization and program launching remain unchanged (see [page 66](#)).

The standard DLL (RTC5DLL.dll) supports all commands for controlling 3-axis scan systems. However, the full functionality of the commands (outputting Z coordinates) is only available if the 3D option has been enabled.

Multiple 3-axis scan systems can be simultaneously controlled (by multi-board commands) from a single PC if a corresponding number of RTC5 Boards with enabled 3D options are installed in that PC.

(1) If, in addition, the "second scan head control" option is enabled, it is alternatively possible to assign a 3D correction table to the secondary scan head connector and to attach the XY scan head to the secondary scan head connector and the Z axis to the primary scan head connector (see also "[Interfaces to Scan System](#)", [page 42](#) and "[2D and 3D Correction Files](#)", [page 137](#)).

8.6.3 3D Marking Commands

The RTC5's command set includes the following 3D vector and timed 3D vector commands:

- `goto_xyz`
- `jump_abs_3d`, `timed_jump_abs_3d`
- `jump_rel_3d`, `timed_jump_rel_3d`
- `mark_abs_3d`, `timed_mark_abs_3d`
- `mark_rel_3d`, `timed_mark_rel_3d`

In addition the RTC5 command set provides the following 3D arc commands which allow spiral marking:

- `arc_abs_3d`, `arc_rel_3d`

Except for the additional motion in the third dimension, these commands function identically to the corresponding 2D marking commands:

- The specified vectors or arcs are microvectorized (with the same output period as with 2D commands, see [page 107](#)).
- As with 2D commands, jump and marking speeds can be specified by `set_jump_speed` and `set_mark_speed` (see [page 103](#)). As for timed 2D vector commands, for timed 3D vector commands, the speeds are automatically determined based on the specified jump or marking duration.
- 3D image field correction is applied in accordance with the 3D correction table assigned by `select_cor_table` or `select_cor_table_list` (see [page 136](#)).
- For jump and mark commands, the laser control signals are switched on and off while taking delay settings into account (see [page 111](#)).

The RTC5 simultaneously calculates output values for the galvanometer scanners of the scan system and the corresponding focal lengths (or focal intercepts) and output values for the Z-axis. Here, the RTC5 uses appropriate data from the assigned 3D correction table. SCANLAB individually calculates each correction table based on the optical configuration of the specific 3-axis system and all available optical system data (such as mirror geometry, calibration, objective specifications and Z-axis optics specifications). Each correction table is stored in a correction file named (*.CT5) in the RTC5 software package (see also `load_correction_file`).

In RTC5 mode, the X and Y vector coordinates of a vector must be specified as signed 20-bit numbers (i.e. as numbers between -524288 and +524287), however the Z coordinates in a 3D system must be signed 16-bit numbers (i.e. numbers between -32768 and +32767). Therefore, the calibration factor (the ratio of a point coordinate in *bits* and the actual position of the point in *millimeters*) is 16 times smaller in the Z direction than in the X and Y directions:

$$K_z = K_{xy} / 16.$$

In contrast, the following should be used for RTC4 compatibility mode (as with the RTC4):

$$K_x = K_y = K_z = K_{xy} / 16.$$

The RTC5 (in RTC5 mode as well as in RTC4 compatibility mode) automatically upscales Z coordinate values internally to 20-bit values, so that the three dimensions of space can be handled equivalently in terms of jump or marking speed values.

Notes

- If vector-defined laser control is activated by `set_vector_control`, then para-mark and para-jump commands (`para_jump_abs_3d`, `timed_para_jump_abs_3d`, `para_jump_rel_3d`, `timed_para_jump_rel_3d`, `para_mark_abs_3d`, `timed_para_mark_abs_3d`, `para_mark_rel_3d`, `timed_para_mark_rel_3d`) can also be used. These commands simultaneously vary a signal parameter linearly along the mark or jump vector (see [page 166](#)). With (Ctrl = 7), for example, the focus shift (i.e. an offset to the calculated focal length) varies (as with `set_defocus` or `set_defocus_list`).
- The size of the usable image field and the maximum focus shift in the Z direction (height of the usable working volume) can be obtained from the "Readme.txt" file supplied with the 3D correction file or from the user manual of the 3-axis scan system or the varioSCAN ("Technical Specifications" chapter).
- If a Z axis serves the purpose of maintaining the laser focus in a particular plane, then 2D vector commands can be used, too (after the desired Z position was set by a 3D vector command). 2D vector commands have no effect on the Z position, but regulate the focal length – if the 3D option is enabled and a 3D correction table was assigned – so that the laser beam maintains focused in the same plane.



- If the 3D option is not enabled or no 3D correction table has been assigned, then 3D marking commands do have the same effect as 2D marking commands. However, microvectorization is calculated like a 3D command and hence influences the effective jump or marking speed in the XY plane.
- If the 3D option is not enabled or no 3D correction table has been assigned, then 3D vector commands do have the same effect as 2D vector commands. However, microvectorization is calculated like a 3D command and hence influences the effective jump or marking speed in the XY plane (this does not apply to the 3D arc commands `arc_abs_3d` and `arc_rel_3d`).
- `set_offset_xyz` or `set_offset_xyz_list` can be used for setting an offset z_0 for the Z coordinate. In addition, `set_defocus` or `set_defocus_list` can be used for defining an offset to the calculated focal length, which then appropriately affect the Z output value (in the direction opposite to z_0).
- During a marking process (while a mark or arc command is being processed), the scan system's focal length is continuously readjusted, so that the laser focus remains sharp within the current image plane. In contrast, during execution of a jump command (or `goto_xyz`) readjustment of the focus is not really necessary. Therefore, the commands `set_delay_mode` and `set_delay_mode_list` allow defining the value of its DirectMove3D parameter to provide a jumping method that changes the Z output value directly (linearly) from its start value to its end value. This can help to avoid needless movements of the Z-axis.
- For 3D marking commands, the following settings (if made) *directly* affect the X and Y output values, but have no effect or only an *indirect* effect on Z output values (see also [page 141](#)):
 - a wobble motion enabled by `set_wobble` or `set_wobble_mode` (see [page 189](#)).
 - a Processing-on-the-fly correction enabled by `set_fly_x`, `set_fly_y`, `set_fly_rot` or `set_fly_x_pos`, `set_fly_y_pos`, `set_fly_rot_pos` (see [page 199](#)).
 - a coordinate transformation for the X and Y output coordinates defined by `set_matrix`, `set_offset`, `set_offset_xyz`, `set_scale`, `set_angle` or corresponding list commands (see [page 183](#)).
 - a gain and offset correction activated by `auto_cal` or `set_hi` (see [page 224](#)).
- The RTC5 command set also includes 3D micro vector commands (see [page 221](#)).

8.6.4 Adjustment

SCANLAB calculates 3D correction tables so that the plane of the focus mid-position (without setting an offset by `set_defocus`, `set_defocus_list`, `set_offset_xyz` or `set_offset_xyz_list`) is achieved with $z = 0$. Particularly when the 3-axis scan system (with its respective 3D correction table) is intended for maintaining the laser focus in a specific plane, you must therefore adjust the mechanical distance between the scan system and the working plane so that this plane coincides with the $z = 0$ plane. If the mechanical distance is different, then the laser focus (at least at the edges of the working field or volume) might not be sharp.

For the varioSCAN and varioSCAN_{de} a specific distance to the scan system has to be maintained. The needed mechanical distances can be found in the respective 3-axis scan system or varioSCAN user manual.

Usually, the working distance used for calculating a correction table can also be queried by `get_table_para`.

Once the working distance and any distance to the scan system are correctly adjusted, then the focus position should be fine-tuned. For this, scan a test pattern onto the middle of the $z = 0$ working plane. The optimum focus position for processing results can be achieved:

- with a varioSCAN and a varioSCAN_{de} by manually adjusting the focusing ring,
- with a varioSCAN FLEX by adjusting the focusing optic's position by the command `move_to`,
- with a varioSCAN FC (also with an intelliWELD FC) by adjusting coefficient A of the assigned 3D correction table⁽¹⁾.

Checking the Z-Axis Calibration

The optimum output values for the Z-axis also depend on various parameters such as beam divergence of the used laser and tolerances of the optical components. Such data is generally not available to SCANLAB and therefore not reflected in calculating 3D correction tables.

Therefore, in some cases the pre-calculated correction table might not fit optimally to the individual scan system. To test whether this is the case, run a laser marking test that covers the entire image field. Check if the laser focus meets the requirements of your application. If you find that the spot diameter varies considerably, it might be necessary to re-calibrate the Z-axis correction.

Here, users can adjust coefficients A, B and C of the parabolic function ($z_{\text{out}} = A + Bl + Cl^2$) that determines the relationship between the Z output value z_{out} and the focus length value l . Here, for each point ($x|y|z$) in the working volume, the (unit-free) focus length value l corresponds to the focus length difference between the specified point ($x|y|z$) and the point ($0|0|0$).

The aim of the Z-axis calibration procedure is to determine suitable coefficients A, B and C. These can be subsequently transmitted to the RTC5 with the command `load_z_table`.

The ABC coefficient values of a specified correction file on the PC can be read-out directly with `read_abc_from_file` and written into with `write_abc_to_file`.

(1) Coefficients A, B and C can be queried by `get_head_para` and newly defined by `load_z_table`. Only A should be modified. B and C should be transmitted unchanged – as queried – by `load_z_table`.

Procedure

- (1) Adjust the correct mechanical distance between the scan system and the $z = 0$ working plane and – if applicable – the correct distance between the varioSCAN and the scan system⁽¹⁾.
- (2) Load the desired 3D correction table by **load_correction_file** and assign it by **select_cor_table**.
- (3) Use the command **get_head_para** to read out the assigned coefficients A , B and C .
For the varioSCAN FC (also with an intelliWELD FC), steps (4) to (7) do not need to be performed. In this case, proceed directly to step (8).
- (4) Use the command **goto_xyz** to move the laser spot to the reference point⁽²⁾.
- (5) Place a test object at the reference point.
- (6) Use the command **load_z_table**(0, 0, 0) to set the Z axis to the neutral (middle) position.
- (7) Now adjust the laser focus
 - with a varioSCAN or a varioSCAN_{de} by manually adjusting the focusing ring
 - with a varioSCAN FLEX by adjusting the focusing optic's position by the command **move_to**.
- (8) Move the focus to an arbitrary point ($x | y | z$) within the required image space using the command **goto_xyz**⁽³⁾.
- (9) Use the command **get_z_distance** to query the focus length value l (in bits) set by the RTC5 for this point⁽⁴⁾.
- (10) Optimize the laser focus at this new location.
Therefore, use the command **load_z_table**($A=z_{\text{out}}$, 0, 0) to vary the Z output value z_{out} until the quality of the laser focus meets your requirements⁽⁵⁾.

- (1) See the corresponding 3-axis scan system's or the varioSCAN's user manual.
- (2) The reference point's coordinate values (in mm) are provided in the "Readme.txt" file that accompanies your 3D correction file or in the 3-axis scan system's or the varioSCAN's user manual. If you are using an F-Theta objective, the reference point is generally the origin $(0 | 0 | 0)$. The reference point is a point in the $z = 0$ working plane for which a middle focus length value is required for a sharp laser focus. The laser beam shall be focused to the reference point when the Z axis is set to the neutral position (Z output value $z_{\text{out}} = 0$).
- (3) If you are using a scan system with an F-Theta objective, it is sufficient to select various points $(0 | 0 | z)$ on the Z coordinate axis. The maximum possible working volume is specified in the corresponding user manual.

(11) Repeat steps (8) through (10) for as many locations $(x | y | z)$ as possible⁽⁶⁾ and write down the values $(l | z_{\text{out}})$ for each new point. If possible, seek to thereby cover the entire working space required by your application.

(12) Fit your set of $(l | z_{\text{out}})$ values to a function of the type $z_{\text{out}} = A + Bl + Cl^2$.

(13) Use the resulting coefficients A , B and C to adjust the 3D correction table by **load_z_table**(A , B , C).

Values loaded by **load_z_table** are overwritten by a subsequent **load_correction_file** command

(load_correction_file sets the three coefficients A , B and C to the default values of the loaded correction table⁽⁷⁾). After each **load_correction_file**, you should therefore also call **load_z_table**(A , B , C) again.

Test for 3-Axis Scan Systems with F-Theta Objective

- With the adjusted 3D correction table, mark two identical large squares using 3D vector commands (see [page 194](#)). Mark one of the squares with the work piece in Z-position $z = z_{\text{min}}$, the other one with $z = z_{\text{max}}$.
- These two squares should match exactly. If this is not the case, your correction file is not perfectly suited to your objective. To solve this problem, measure the size (x and y) of both squares and report these values to SCANLAB. You will then receive a new 3D correction file.

- (4) The focus length value l can be positive or negative.
- (5) The optimal z_{out} output value can be positive or negative. When seeking the optimal laser focus, you can calculate a starting value in accordance with $A + Bl + Cl^2$ by using the previously read focus length value l and the previously read or used values A , B and C .
- (6) Note that larger z control values lead to shorter working distances and that the focus thereby shifts toward the scan system. The test object might have to be tracked accordingly.
- (7) Each 3D correction table calculated by SCANLAB contains values for the coefficients A , B and C (see also the respective "Readme.txt" file or **get_table_para**).

8.6.5 Enhanced 3D Correction

The image size of 3D scan systems – particularly those without objectives or with non-telecentric F-Theta objectives – depends on its distance to the scan head's objective (i.e. from the Z coordinates). The image field size typically gets stretched or squeezed linearly with the Z value. Therefore, SCANLAB 3D correction files include stretch correction factors to compensate for this effect (see [page 140](#)).

With some objectives, the typical stretching is combined with a change in the image geometry. Then additional stretch corrections are necessary which compensate the image geometry changes XY-position-dependent but linear in Z. The stretch correction values are meaning the Z gradient for the location deviation at this point.

Assumption: for any chosen non-zero Z plane, (X, Y) is the desired position and (X', Y') the measured position. The corrections <StretchX> and <StretchY> are then calculated as follows:

$$\begin{aligned} <\text{StretchX}> &= (X - X') / Z \\ <\text{StretchY}> &= (Y - Y') / Z \end{aligned}$$

The measurement can be performed as often as desired in an iterative procedure to obtain optimal correction value settings. Each newly determined correction value can simply be added to the prior values for a total correction.

You can then use the **load_stretch_table** command to load the enhanced 3D correction onto the RTC5 from an ASCII text file. This file must contain corrections for a complete rectangular grid. The number of gridlines and their spacings can be freely defined and even differ in X and Y. If the absolute values of the corrections exceed 0.03125, then they are clipped to this limit value. The corrections are bilinearly interpolated for data points within the specified grid and linearly extrapolated for data points outside the grid.

After **load_program_file**, enhanced 3D correction is inactive by default. It becomes active upon loading a valid table onto the RTC5 Board and can subsequently be deactivated by calling **load_stretch_table** using a null pointer instead of the filename.

The following rules apply to the text file:

- The file can contain one or several tables, even tables of other types.
- The 2D stretch correction table must begin with the caption [StretchTable<No>], whereby <No> corresponds to the number supplied with the **load_stretch_table** command.
- This is directly followed by a block of data points.
- If several tables with the same number exist, then only data from the first encountered table is read. The others are ignored.
- Reading of the text data terminates upon end of file or upon a line containing a caption.
- All characters to the right of a semicolon are treated as comments and ignored.
- The order of data points is up to you.
- The maximum length of a data line is 255 characters.
- A data line contains two position coordinates (in bits, signed integers) and two correction values (dimensionless, signed floating point numbers, use the period (.) or comma as the decimal separator), each separated by spaces or tabs:
 - <Xpos> <Ypos> <StretchX> <StretchY>
 - If a data point reoccurs, then the most recently read value is used; the others are ignored.
 - Empty lines or incomplete data lines are invalid and are ignored.

8.7 Processing-on-the-fly (Optional)

8.7.1 Intended Use and Initialization

With its Processing-on-the-fly option enabled, the RTC5 allows processing of parts in motion (e.g. parts on a conveyor belt, rotating plate or XY translation stage), as well as stationary parts with a moving scan system (e.g. by a robot arm).

To adjust laser scan processes to the current work-piece position relative to the scan system, the position of the workpiece or scan system can be indirectly (by encoder counters) or directly (by the McBSP/SPI interface) forwarded to the RTC5. With activated Processing-on-the-fly correction, the coordinate values of all vector and arc commands are transformed based on the forwarded position values.

If the position values are forwarded by encoder counters, then the motion is simulated or detected by user-supplied incremental encoders. The simulated or detected encoder signals trigger an (RTC5) internal counter (encoder counter) whose counter values then correspond to the current position and are applied as the basis for Processing-on-the-fly correction (see “[Synchronization by Encoder Signals](#)”, page 246). If forwarded by the McBSP/SPI interface, the input values are directly applied as the basis for Processing-on-the-fly correction (see “[Synchronization and Online Positioning by McBSP/SPI Signals](#)”, page 248).⁽¹⁾

The commands for activating and deactivating Processing-on-the-fly correction are supported by the standard DLL (RTC5DLL.dll, if the Processing-on-the-fly option is enabled by SCANLAB). No additional drivers or software files (e.g. further DLL or DSP program files) are needed. Initialization and program launching remain unchanged as well (see [page 66](#)). List commands are available for activating and deactivating Processing-on-the-fly correction. The parameters required for activation can be determined by a calibration procedure (see below).

Activated Processing-on-the-fly correction *directly* affects only the X and Y output values, but (with the 3D option enabled) has no effect or only an *indirect* effect on Z output values (see also [page 141](#)). If the “second scan head control” option has been enabled and both scan head connectors have been assigned a 2D correction table, then Processing-on-the-fly corrections are equally applied at both scan head connectors.

For an installed RTC5, the command `get_RTC_version` can be used to check whether the Processing-on-the-fly option is enabled.

As of version DLL 530, OUT 531 a Processing-on-the-fly correction for the Z-axis can be activated (see [chapter 8.7.11 “Processing-on-the-fly Correction for the Z-Axis — “FlyZ Correction” \(as of Version DLL 530, OUT 531\)”, page 215](#)).

Notes

- Activated Processing-on-the-fly correction can affect not only vector and arc commands, but (as of version OUT 515) also processing of normal list commands that do not produce scanner motion (e.g. `list_nop`, `set_control_mode_list`). Even with such commands, the positions of the X and Y axes are continually tracked in accordance with the current Processing-on-the-fly input signals (position of the workpiece or scan system). In older versions, execution of such commands was sometimes followed by a hard jump to the changed position value.

(1) Simultaneous usage of both forwarding methods for Processing-on-the-fly correction of two independent motions is not possible (see [section “Overview”, page 200](#)). The McBSP/SPI interface cannot be simultaneously used for both Processing-on-the-fly applications and online positioning (see [page 188](#)).



Overview

The following encoder-based Processing-on-the-fly corrections of workpiece motions are available:

- **set_fly_x, set_fly_y**: Compensation of linear workpiece motions (e.g. by conveyor belt or XY stage) – see [chapter 8.7.2, page 201](#).
- **set_fly_rot**: Compensation of rotary workpiece motions (e.g. by rotating plate) – see [chapter 8.7.3, page 204](#).
- **set_fly_2d**: Compensation of 2D workpiece motions (e.g. XY stage) – see [chapter 8.7.4, page 206](#).

Additionally, the following McBSP/SPI-based Processing-on-the-fly corrections of scan-system motions are available:

- **set_fly_x_pos, set_fly_y_pos**: Compensation of linear or 2D scan-system motions (e.g. robot arms) – see [chapter 8.7.2, page 201](#).
- **set_fly_rot_pos**: Compensation of scan-system rotary motions (e.g. robot arms – see [chapter 8.7.3, page 204](#).
- **set_mcbsp_in, set_mcbsp_in_list**: Compensation of linear, 2D or rotary scan-system motions (e.g. robot arms) – see [chapter 8.7.2, page 201](#) and [chapter 8.7.3, page 204](#).

The following Processing-on-the-fly corrections can be combined:

- **set_fly_x** is combinable with **set_fly_y**.
- **set_fly_x_pos** is combinable with **set_fly_y_pos**.

Further combinations are *not* possible (except those mentioned in [chapter 8.7.11](#)). For example, encoder-based **set_fly_x** correction of linear workpiece motion is combinable *neither* with **set_fly_rot** correction of workpiece rotary motions *nor* with McBSP/SPI-based Processing-on-the-fly correction of scan-system motions.

The last command always determines the overall correction. For example, **set_fly_rot** deactivates any previously activated Processing-on-the-fly correction and activates compensation of workpiece rotary motions. The only exceptions are the above-mentioned combinations (e.g. **set_fly_y** does *not* deactivate **set_fly_x**).

However, mutual synchronization of any Processing-on-the-fly corrections by **wait_for_encoder_mode**, **wait_for_encoder_in_range** and **wait_for_mcbsp** is possible (see [chapter 8.7.7, page 211](#)).

Furthermore, encoder-based Processing-on-the-fly corrections can be combined with tracking error compensation (see [chapter 8.7.10, page 215](#)).

8.7.2 Compensation of Linear Movements

Processing-on-the-fly correction for linear workpiece movements can be activated by the list commands `set_fly_x` and/or `set_fly_y` or alternatively by the list commands `set_fly_x_pos` and/or `set_fly_y_pos` or by `set_mcbsp_in` or `set_mcbsp_in_list`

(Mode = 1...3)⁽¹⁾. A scaling factor must thereby be specified.

With `set_multi_mcbsp_in` or `set_multi_mcbsp_in_list`, you can activate additional Processing-on-the-fly correction with positional values for linear motion in all three coordinate directions without bit-resolution restrictions. No scaling factor is required.

Processing-on-the-fly correction can be stopped (simultaneously for both directions) by the command `fly_return` (see the corresponding notes on page 209).

Correction by Encoder Counter(s)

If position values for Processing-on-the-fly correction are forwarded by the internal encoder counters, then Processing-on-the-fly correction must be activated by `set_fly_x` and/or `set_fly_y`. Here, the scaling factor [in bits per count] defines the relation between the shift [in bits] of the current output position in the image field and one counter pulse (count) of the corresponding encoder counter (see below).

The `set_fly_x` or `set_fly_y` command reset the respective encoder counter (Encoder0 or Encoder1) to zero⁽²⁾.

Thereafter, the output value is calculated from the current output position by adding (for each direction) the product of the scaling factor and the current counter value. This correction is performed every 10 µs⁽³⁾.

- (1) There are restrictions on combining Processing-on-the-fly corrections with each other (see the section "Overview", page 200).
- (2) With `set_control_mode`, bit#9 can be set in advance to determine whether the counter should reset immediately or only after a subsequent start trigger (i.e. with a subsequent external start signal, `simulate_ext_start` or `simulate_ext_start_ctrl` command, possibly postponed by a track delay defined by `simulate_ext_start`, `set_ext_start_delay` or `set_ext_start_delay_list`; see also `set_control_mode`, bit#2). If the counter only gets reset after the subsequent trigger, then this effectively eliminates the 10 µs jitter (random time offset between the start signal and the list start).
- (3) The encoder counters are 32-bit counters (for signed 32-bit values). If the maximum (minimum) count is reached, then counting continues with the minimum (maximum) count.

Notes

- `set_fly_x` and `set_fly_y` can be used in combination for 1D Processing-on-the-fly applications (e.g. an obliquely angled conveyor belt) as well as for encoder-based 2D Processing-on-the-fly applications (e.g. an XY translation stage), particularly for separate or separable marking tasks in the real image field. For continuous marking in the virtual image field, SCANLAB instead recommends using `set_fly_2d` (see chapter 8.7.4 "Compensation of 2D Motions", page 206 and chapter 8.7.11 "Processing-on-the-fly Correction for the Z-Axis — "FlyZ Correction" (as of Version DLL 530, OUT 531)", page 215).

Determining Scaling Factors

If the workpiece position is to be registered by one (or two) incremental encoder(s), then a calibration procedure is required to determine the scaling factor:

First, the command `get_encoder` must be used to determine the encoder increments i_x and i_y [in counts per mm] for each direction:

- ▶ Read the counter start value by `get_encoder` and begin the movement.
- ▶ Stop the movement and read the counter end value by `get_encoder`⁽⁴⁾.
- ▶ Measure the distance travelled in mm.
- ▶ The encoder increment i can then be calculated as follows:

$$i = (\text{counter end value} - \text{counter start value}) / \text{distance travelled}$$

In a second step, the scaling factors Scale_x and Scale_y [in bits per count] can be calculated from the determined encoder increments as follows:

$$\begin{aligned}\text{Scale}_x &= K / i_x \\ \text{Scale}_y &= K / i_y\end{aligned}$$

whereby K is the calibration factor [in bits per mm] (see page 134).

- (4) Alternatively, the counter start and end values can be stored in a buffer on the RTC5 by the list command `store_encoder` and then retrieved from there by the control command `read_encoder`.

If the workpiece moves at a constant speed v_x or v_y [in mm per second] and an encoder simulation was activated by **simulate_encoder**, then the scaling factors are calculated as follows:

$$Scale_x = K \times v_x / (1000000 \text{ counts/s})$$

$$Scale_y = K \times v_y / (1000000 \text{ counts/s})$$

It might be necessary to adjust the signs of the scaling factors to the direction of movement.

Correction by McBSP/SPI Interface

If position values for Processing-on-the-fly correction are forwarded by the McBSP/SPI interface, then Processing-on-the-fly correction must be activated by **set_fly_x_pos** and/or **set_fly_y_pos**. Here, the scaling factor [in bits per bits] defines the relation between the shift [in bits] of the current output position in the image field and the input value [in bits] at the McBSP/SPI interface (see below).

Thereafter, the output value is calculated from the current output position by adding (for each direction) the product of the scaling factor and the current input value at the McBSP/SPI interface. This correction is performed every 10 µs.

For one-dimensional correction, the McBSP/SPI interface provides a (signed) 32-bit value. For two dimensions, in contrast, the position values of the two axes can only be forwarded by a (signed) 16-bit value, each. Here, the X axis gets the lower 16 bits and the Y axis gets the upper 16 bits of the value at the McBSP/SPI interface. For a description of the interface, see [page 54](#).

Determining the Scaling Factor

If workpiece or scan system position is to be registered by the McBSP/SPI interface, then a calibration procedure is required to determine the scaling factor:

First, the command **read_mcbsp** must be used to determine the position increments i_x and i_y [in bits per mm] for each direction:

- ▶ Read the start input value (by **read_mcbsp**) and begin the movement
- ▶ Stop the movement and read the end input value (by **read_mcbsp**)
- ▶ Measure the distance travelled in mm

- ▶ The position increment i can then be calculated as follows:

$$i = (\text{end input value} - \text{start input value}) / \text{distance travelled}$$

In a second step, the scaling factors $Scale_x$ and $Scale_y$ [in bits per bits] can be calculated from the determined position increments as follows:

$$Scale_x = K / i_x$$

$$Scale_y = K / i_y$$

whereby K is the calibration factor [in bits per mm] (see [page 134](#)).

Notes

- After **set_fly_x_pos** and **set_fly_y_pos**, the input values received at the McBSP/SPI interface automatically get copied to internal memory location 0. This still applies even after Processing-on-the-fly correction gets switched off by **fly_return**, **set_fly_x_pos**, **set_fly_y_pos** or **set_fly_rot_pos**, as well as after a reset by **load_program_file**. The current data at memory location 0 can be queried by **read_mcbsp(0)**.
- In contrast, activation of Processing-on-the-fly correction by **set_mcbsp_in** or **set_mcbsp_in_list** might also result in McBSP/SPI input values being copied to internal memory locations 1, 2 and/or 3 (see [page 203](#)). If online positioning is activated, then they are copied to internal memory location 1 or possibly 2 (but not to memory locations 0 or 3) (see [page 187](#)).
- After Processing-on-the-fly correction is activated by **set_multi_mcbsp_in** or **set_multi_mcbsp_in_list**, the transmitted data gets consecutively written to memory locations 0 through 3. You can also query it from there (unsorted) with **read_mcbsp**.

Correction by McBSP/SPI Interface with Additional McBSP/SPI Input

To activate Processing-on-the-fly correction for linear motions using McBSP/SPI input values (as an alternative to `set_fly_x_pos` or `set_fly_y_pos`), you can also call `set_mcbsp_in` or `set_mcbsp_in_list` (Mode = 1...3).

These commands offer the advantage of using the McBSP/SPI interface to input additional desired signals that should not be subjected to Processing-on-the-fly correction even when it is activated. For this, all McBSP/SPI input values must be coded by bit#31.

- Bit#31 = 0: The input value gets copied to internal memory location 0 and is applied for Processing-on-the-fly correction.
- Bit#31 = 1: The input value gets copied to internal memory location 3 but is not applied for Processing-on-the-fly correction.

Notes

- All input values always get copied alternatingly to internal memory locations 1 and 2 and subsequently, in accordance with their bit#31 coding, to internal memory locations 0 and 3. But after deactivation of correction by `set_mcbsp_in(0)` or `set_mcbsp_in_list(0)`, they only get copied to memory locations 1 and 2.
- You can query the data currently stored at internal memory locations 0 - 3 by `read_mcbsp`.
- You can specify a scaling factor by `set_mcbsp_in` or `set_mcbsp_in_list` (as with `set_fly_x_pos` and `set_fly_y_pos`). For two-dimensional correction, the scaling factor applies to both axes simultaneously. Calibration for determining the scaling factor is the same as for `set_fly_x_pos` and `set_fly_y_pos` (see above).
- For transferring one-dimensional correction values, 31 bits with sign are available (bit#31 is reserved as the coding bit). In contrast, only 15 bits with sign are available per axis for transferring two-dimensional correction values, whereby the X value lies in the lower 16 bits and the Y value in the upper 16 bits of the value at the McBSP/SPI interface. For a description of the interface, see [page 54](#).

Correction by McBSP/SPI Interface with Enhanced McBSP/SPI Input

As an alternative to the previous chapter's methods, you can also activate Processing-on-the-fly correction of linear motion with `set_multi_mcbsp_in` or `set_multi_mcbsp_in_list`.

These commands have an advantage over `set_mcbsp_in` or `set_mcbsp_in_list` in that they offer Processing-on-the-fly correction not only in the X and Y directions, but also in the Z direction and with laser power variation. Furthermore, up to four additional signals can be transmitted for usage as you wish.

Each 10 µs, data asynchronously transmitted to McBSP/SPI memory locations 0 through 3 gets sorted and copied to an additional internal memory location. From there it is available for final usage and can be read-out using `read_multi_mcbsp`.

8.7.3 Compensation of Rotary Movements

Before activating Processing-on-the-fly correction for rotary XY movement, you must define the rotation center by `set_rot_center` or `set_rot_center_list`. The rotation center may also be situated outside the image field. The Processing-on-the-fly correction itself can be activated by the list command `set_fly_rot` or alternatively by `set_fly_rot_pos` or by `set_mcbsp_in` or `set_mcbsp_in_list` (Mode = 4).

Processing-on-the-fly correction can be stopped by the command `fly_return` (see the corresponding notes on [page 209](#)).

Notes

- You cannot combine Processing-on-the-fly rotation correction with other Processing-on-the-fly corrections (see [section "Overview", page 200](#)).
- Simultaneous Processing-on-the-fly rotation correction for both scan head connectors is only practical if both attached scan heads are aligned to exactly the same rotation center.

Correction by Encoder Counter

If an incremental encoder is used to detect the angular position of the parts to be processed, the encoder must be connected to the ENCODER X input (see also [page 247](#)).

Then, Processing-on-the-fly correction must be activated by `set_fly_rot`⁽¹⁾. The parameter `Resolution` [in counts per revolution] must thereby be specified. The inverse of this parameter defines the rotation [in number of revolutions] of the current output position in the image field corresponding to one counting unit (see below).

The `set_fly_rot` command resets the encoder counter Encoder0 to zero.

Thereafter, the output value is calculated from the current output position by a linear transformation (rotation around the specified rotation center, where $\text{rotation angle} / 360^\circ = \text{current Encoder0 counter value} / \text{Resolution}$). This correction is performed every 10 µs⁽²⁾.

Determining the Resolution Parameter

If the angular position of the workpiece is to be registered by an incremental encoder, then a calibration must be performed to determine a value for the `Resolution` parameter:

- ▶ Read the counter start value (by `get_encoder`) and begin the rotation (any desired number of full rotations)
- ▶ Count the number of revolutions.
- ▶ Stop the rotation and read the counter end value (by `get_encoder`)⁽³⁾.
- ▶ The parameter `Resolution` [i.e. the counts per revolution] can then be calculated as follows:

$$\text{Resolution} = (\text{counter end value} - \text{counter start value}) / \text{number of revolutions}$$

Note that the number of counts per revolution is equal to the number of encoder signal edges per revolution i.e. four times the number of encoder increments per revolution (see [page 247](#)).

If the workpiece rotates at a constant speed ω [in number of revolutions per second] and an encoder simulation was activated by `simulate_encoder`, then the `Resolution` parameter can be calculated as follows:

$$\text{Resolution} = (1000000 \text{ counts/second}) / \omega$$

It might be necessary to adjust the sign of `Resolution` to the direction of rotary movement.

(1) See footnote ⁽²⁾ on [page 201](#).

(2) The encoder counters are 32-bit counters (for signed 32-bit values). If the maximum (minimum) count is reached, then counting continues with the minimum (maximum) count.

(3) Alternatively, the counter start and end values can be stored in a buffer on the RTC5 by the list command `store_encoder` and then retrieved from there by the control command `read_encoder`.



Correction by McBSP/SPI Interface

If angle-position values for Processing-on-the-fly correction of rotary movement are forwarded by the McBSP/SPI interface, then Processing-on-the-fly correction must be activated by `set_fly_rot_pos`. Here, the required `Resolution` parameter has the same meaning as with `set_fly_rot` and can be similarly determined (see above), with the difference that McBSP/SPI input values are queried by `read_mcbsp` (while counter values are queried by `get_encoder`).

Notes

- As with using McBSP/SPI input values to compensate linear motions, McBSP/SPI input values get copied to internal memory location 0 (see notes on [page 202](#)).

Correction by McBSP/SPI Interface with Additional McBSP/SPI Input

You can also activate Processing-on-the-fly correction of rotary movements with McBSP/SPI input values (as an alternative to `set_fly_rot_pos`) by calling `set_mcbsp_in` or `set_mcbsp_in_list` (Mode = 4).

These commands offer the advantage of using the McBSP/SPI interface to input additional desired signals that should not be subjected to Processing-on-the-fly correction even when it is activated. For this, all McBSP/SPI input values must be coded by bit#31.

Notes

- All input values always get copied alternately to internal memory locations 1 and 2 and subsequently, in accordance with their bit#31 coding, to internal memory locations 0 and 3. But after deactivation of correction by `set_mcbsp_in(0)` or `set_mcbsp_in_list(0)`, they only get copied to memory locations 1 and 2.
- You can query the data currently stored at internal memory locations 0 - 3 by `read_mcbsp`.
- You can specify a rotation resolution by `set_mcbsp_in` or `set_mcbsp_in_list` (as with `set_fly_rot_pos`). Calibration for determining the rotation resolution is the same as for `set_fly_rot_pos` (see above).
- For transferring rotary correction values, 31 bits with sign are effectively available.

8.7.4 Compensation of 2D Motions

You can use the `set_fly_2d` command to activate encoder-based 2D Processing-on-the-fly correction (e.g. for XY stages), particularly for continuous marking in the virtual image field. Unlike activation by `set_fly_x` and `set_fly_y`, this offers the following advantages:

- `set_fly_2d` simultaneously resets both encoders (whereas the separate commands `set_fly_x` and `set_fly_y` do so with a slight temporal offset between both channels)
- `set_fly_2d` allows compensation of non-linear relations between encoder values and actual XY stage motions (see the section "2D Encoder Compensation for XY Stages", page 206).
- `set_fly_2d` allows using coordinate transformations within the virtual image field (see the section "Coordinate Transformations in the Virtual Image Field", page 208). As of version DLL 541, OUT 541 this is also possible with `set_fly_x` and/or `set_fly_y`.
- Even during an interruption of a `set_fly_2d` marking by `wait_for_encoder_mode` or `wait_for_encoder_in_range`, the galvanometer scanner positions receive continuous Processing-on-the-fly correction in accordance with the current XY-stage encoder values (whereby the laser focus remains stationary relative to the XY stage). Thus, unnecessary jumps are avoided after XY stage motions (see chapter 8.7.6 "Virtual Image Field", page 210).

Notes

- You cannot combine `set_fly_2d` correction with other Processing-on-the-fly corrections (see the section "Overview", page 200).

2D Encoder Compensation for XY Stages

For particularly demanding marking requirements, it might be necessary to also compensate an XY translation stage's mechanical deviations. Here, you can use the `load_fly_2d_table` command to load a 2D compensation table onto the RTC5 (see below). Then – during a Processing-on-the-fly application initiated by `set_fly_2d` – encoder values are two-dimensionally interpolated and compensated just like with field correction tables.

To avoid unnecessary initialization motions, you can use the `init_fly_2d` command to define a desired translation stage start position as the reference value for 2D encoder compensation (and to store it on the RTC5 Board). Upon every subsequent encoder reset by `set_fly_2d`, the current position is automatically applied as the new reference position, so that the relation between current encoder values and the stage's absolute position is retained for compensation. This relation remains even upon termination with `fly_return` or upon a new start with `set_fly_2d`, but not if you meanwhile activate other Processing-on-the-fly corrections or reset the encoders by an external /START (see `set_control_mode`, bit # 9 = 1). At any time, you can query the currently valid reference values by `get_fly_2d_offset`.

Encoder compensation is applied exclusively to Processing-on-the-fly correction. The commands `get_encoder`, `store_encoder` and `read_encoder`, as well as `wait_for_encoder`, `wait_for_encoder_mode` and `wait_for_encoder_in_range` use the original uncompensated encoder values. The same applies to recording by `set_trigger` (Signal = 43 or 44) and `get_value`.



For the **load_fly_2d_table** command, you need to provide an ASCII text file that contains a compensation table. This table must have encoder compensations for reference points on a rectangular grid. The number of grid lines and their spacing is up to you (both may also differ in X and Y). Missing reference point data is automatically assigned a compensation of 0. The largest occurring encoder reference points form a frame within which encoder compensation values are bilinearly interpolated. Encoder values outside this frame is clipped to this frame prior to interpolation. Therefore, the reference points should cover at least the range of the XY stage required by your application. Reference points with values exceeding ± 524288 can result in some loss of precision.

After **load_program_file**, 2D encoder compensation is inactive by default. It becomes active as soon as a valid table from an ASCII-readable text file gets loaded onto the RTC5 Board. You can subsequently deactivate it by calling **load_fly_2d_table** using a null pointer instead of the filename.

The following rules apply to the text file:

- The file can contain one or several tables, even tables of other types.
- The 2D compensation table must begin with the caption `[Fly2DTable<No>]`, whereby `<No>` corresponds to the number supplied with the **load_fly_2d_table** command.
- This is directly followed by a block of data points.
- If several tables with the same number exist, then only data from the first encountered table is read. The others are ignored.
- Reading of the text data terminates upon end of file or upon a line containing a caption.
- All characters to the right of a semicolon are treated as comments and ignored.
- The order of data points is up to you.
- The maximum length of a data line is 255 characters.
- A data line contains the two reference point coordinates for Encoder0 and Encoder1 (signed integers) and two compensation values (signed integers), each separated by spaces or tabs:
`<Encoder0> <Encoder1> <Encoder0-delta>
<Encoder1-delta>`
- If a data point reoccurs, then the most recently read value is used; the others are ignored.
- Empty lines or incomplete data lines are invalid and are ignored.



Coordinate Transformations in the Virtual Image Field

For **set_fly_2d** Processing-on-the-fly applications, you can also define coordinate transformations in the virtual image field. As of version DLL 541, OUT 541 this is also possible with **set_fly_x** and/or **set_fly_y**.

Such “virtual coordinate transformations” (particularly translations and rotations) are sometimes needed to compensate certain mechanical tolerances of object positioning when performing continuous marking larger than the real image field.

Unlike other head-specific coordinate transformations, the (possibly additional) virtual coordinate transformations are applied to the entire virtual image field *before* Processing-on-the-fly correction.

You can define virtual coordinate transformations by the commands **set_matrix**, **set_offset** and **set_offset_xyz** (parameter HeadNo = 4). They let you define matrix coefficients and offsets (however **set_offset_xyz** ignores OffsetZ). The data is cached on the board and applied for calculations after the next **set_fly_2d** command. The parameter at_once has no significance if HeadNo = 4. While a Processing-on-the-fly application is active, you cannot change parameters for virtual coordinate transformation.

The range for offsets is ± 23 bits. Matrix coefficients must not exceed an absolute value of 1.5. As with head-specific transformations, the offset is applied after the matrix operation.

For virtual coordinate transformations you *cannot* use:

- **set_scale**
- **set_angle**
- any coordinate transformation list commands

8.7.5 Deactivating Processing-on-the-fly Corrections

All Processing-on-the-fly corrections can be deactivated (simultaneously for both directions) by the command **`fly_return`**, which also allows the new output position to be defined.

Notes

- If Processing-on-the-fly correction is not explicitly deactivated, then it is also active during execution of subsequent lists, but not in the pause between two lists⁽¹⁾.
- Processing-on-the-fly correction enabled by **`set_fly_x`**, **`set_fly_y`**, **`set_fly_2d`**, **`set_fly_rot`**, **`set_fly_x_pos`**, **`set_fly_y_pos`** or **`set_fly_rot_pos`** also gets deactivated if the same command is called again but this time with invalid parameter values. This could lead to a jump to an unintended output position (also refer to the command descriptions).
- If Processing-on-the-fly was activated by **`set_fly_x`**, then **`set_fly_y`** does not deactivate it and vice versa. The same applies to **`set_fly_x_pos`** and **`set_fly_y_pos`**. Other than that, every Processing-on-the-fly command automatically deactivates any Processing-on-the-fly correction activated by another Processing-on-the-fly command (see also [section "Overview", page 200](#)). Here, a hard jump to a new output position might occur.
- Processing-on-the-fly correction activated by **`set_mcbsp_in`** or **`set_mcbsp_in_list`** also gets deactivated if you call **`set_mcbsp_in`** or **`set_mcbsp_in_list`** with Mode = 0. Here, a hard jump to an uncorrected output position might occur.
- Processing-on-the-fly correction activated by **`set_fly_x_pos`**, **`set_fly_y_pos`**, **`set_fly_rot_pos`**, **`set_mcbsp_in`** or **`set_mcbsp_in_list`** also gets deactivated if you call the command for configuring online positioning (see [page 188](#)). Here, a hard jump to a new output position might occur.

(1) **`set_end_of_list`** does not deactivate Processing-on-the-fly correction. On the other hand, the correction does not affect the control commands **`goto_xy`** and **`goto_xyz`**.

8.7.6 Virtual Image Field

When using the virtual 24-bit image field's extended range (see [page 135](#)) for Processing-on-the-fly applications, you can also load command lists with objects up to 16 times larger than the real 20-bit image field. These command lists get fully and autonomously processed without the need for additional control measures.

In 1D Processing-on-the-fly applications (e.g. work-pieces on a conveyor belt), objects can only exceed the real image field in one dimension (parallel to the Processing-on-the-fly direction). In 2D Processing-on-the-fly applications (e.g. with an XY translation stage), the command list's markable objects can be distributed across the entire virtual image field.

If your overall marking job consists of multiple marking sub-jobs ("tiles") whose extents do not exceed the real image field, then you can also use Processing-on-the-fly functionality to merely move each sub-job into the real image field for subsequent processing (here, sub-job processing even may occur without Processing-on-the-fly motions). But if the overall marking job *is not* decomposable into separate tiles (e.g. when marking a large closed frame around other objects), then continuous Processing-on-the-fly marking is necessary.

The possibility of marking objects outside the real image field exists only because of the following: When a list is processed with activated Processing-on-the-fly correction, then coordinate values outside the real image field get clipped to the boundaries of the real image field (20 bits) only after Processing-on-the-fly correction (see [chapter 7.3.5, page 141](#)). If the supplied coordinate values take into account the motion of the workpiece or scan system as well as the defined Processing-on-the-fly correction and any defined track delay (see [chapter 8.7.7, page 211](#)), then coordinate values outside the real image is transformed at runtime into coordinate values within the real image field.

Coordinate values residing outside the real image even *after* Processing-on-the-fly correction are still clipped to the boundaries of the real image field (here, the `get_marking_info` error bits get set automatically, see [page 213](#)).

To ensure that coordinate values never lie outside the 20-bit image field *after* Processing-on-the-fly correction, always structure your command lists so that – during list execution – exceedance of the 20-bit image field is always preceded (and thereby avoided) by an appropriate motion of the conveyor belt, XY stage etc. For this purpose, you must insert `wait_for_encoder`, `wait_for_encoder_mode`, `wait_for_encoder_in_range` or `wait_for_mcbsp` into the list. These commands interrupt list execution until specific encoder or McBSP/SPI values are achieved (see also [chapter 8.7.7, page 211](#)). The users are responsible for appropriate decomposition of the overall marking task and synchronization with 1D or 2D Processing-on-the-fly motion (of a conveyor belt, XY translation stage, etc.).

For `set_fly_2d` applications, coordinate transformations in the entire virtual image field are also possible (see the section "Coordinate Transformations in the Virtual Image Field", [page 208](#)).

8.7.7 Synchronization of Processing-on-the-fly Applications

Execution of command lists can be started by either a software command or an external start signal (the latter triggered, for instance, by a light barrier or a mechanical contact) (see [page 78](#)).

If execution of the scanning process should not be started directly after receipt of the light-barrier signal (e.g. because the transport system first needs to position the workpiece into the scan system's real image field), then some ways to implement an appropriate delay are:

- For position-forwarding by the McBSP/SPI interface: insert a `wait_for_mcbsp` command at the beginning of the command list.
- For position-forwarding by encoder/counters: insert `set_fly_x/set_fly_y` or `set_fly_rot` (to reset the counter(s)) and a subsequent `wait_for_encoder_mode` command at the beginning of the command list
- For position-forwarding by encoder counters: insert `set_fly_x/set_fly_y`, `set_fly_2d` or `set_fly_rot` (to reset the counter(s)) and a subsequent `wait_for_encoder_mode` or `wait_for_encoder_in_range` command at the beginning of the command list.
`wait_for_encoder_in_range` is useful for 2D encoder-based Processing-on-the-fly applications. This function waits until both encoders are simultaneously within the specified range and thus does not depend on the actual trajectory used to reach this range⁽¹⁾.
- Alternative for position-forwarding by encoder/counters: With external list starts, a track delay can be configured that postpones execution of the list start relative to the triggering input signal or corresponding command (see [page 240](#)). This also allows the execution of objects in the virtual 24-bit image field (outside the real 20-bit image field) to be delayed long enough for their coordinates to be transformed into the real image field by Processing-on-the-fly corrections.

External list starts triggered by an external start signal (or by `simulate_ext_start` or `simulate_ext_start_ctrl`) that do not execute immediately because of the track delay setting are held in a queue that can accommodate up to 8 starts (each start trigger is automatically generated when the delay has expired, see also [page 240](#)). This helps avoid dead time between the execution of multiple (Processing-on-the-fly) list programs. Moreover, the list command `simulate_ext_start` can be used to trigger further list starts at defined intervals.

The commands `wait_for_encoder`, `wait_for_encoder_mode`, `wait_for_encoder_in_range` and `wait_for_mcbsp` can also be used to interrupt a list for intermediate motions (of the conveyor belt or XY stage etc., e.g. when using a virtual image field – see [chapter 8.7.6, page 210](#)). If the intermediate motions are not for marking purposes, then you should switch off the "laser active" laser control signals prior to each motion (hence before interrupting the list by `wait_for_encoder` etc.). Otherwise, such motions might result in unintended marking or burn-in. For encoder-based Processing-on-the-fly applications, you can also use `park_position` to temporarily (before an interruption) move the laser focus to a safe park position. After the interruption, use `park_return` to move back to the starting position or some other position.

(1) If you would use `wait_for_encoder` or `wait_for_encoder_mode` instead, you would need to call these commands individually and consecutively for each encoder. Here, simultaneous fulfilment of both encoder criteria would depend on the XY stage motion's explicit trajectory and you would need to define an appropriate trajectory even before loading the lists.



Galvanometer scanner behavior during intermediate motions (i.e. during list interruption after `wait_for_encoder` etc.) is dependent on the applied Processing-on-the-fly correction:

- With McBSP/SPI-based Processing-on-the-fly correction as well as the encoder-based Processing-on-the-fly corrections `set_fly_x`, `set_fly_y` and `set_fly_rot`, the galvanometer scanners are stationary during list interruption. After interruption, the next marking may need to be preceded by a jump to the desired starting position.
- Encoder-based `set_fly_2d` Processing-on-the-fly correction continuously corrects the galvanometer scanner positions in accordance with the current encoder values (e.g. of an XY stage), whereby the laser focus remains stationary relative to the XY stage and is therefore always at the same XY-stage location during the intermediate motion). The advantage: when a continuous line is processed whose length requires fragmentation by a `wait_for_encoder_in_range` command, then the galvanometer scanners are already at the correct location after the intermediate motion (here, no jump is required after the interruption).
- With `set_fly_2d`, clipping might occur if the Processing-on-the-fly-corrected coordinate values exceed the real image field during a long intermediate motion. To avoid this, it might make sense to switch off Processing-on-the-fly correction before the motion (e.g. by `fly_return` – see [chapter 8.7.5, page 209](#); the galvanometer scanners then remain stationary). To switch correction back on after the intermediate motion, you can use the command `activate_fly_2d` instead of `set_fly_2d` (or `activate_fly_xy` instead of `set_fly_x` and `set_fly_y`). These commands do not reset the encoders, but instead recalculate the last Processing-on-the-fly-uncorrected coordinate values so that the Processing-on-the-fly-corrected output matches the current output. If resuming correction by these commands results in an error (e.g. because the current recalculated coordinate values would now exceed the 24-bit virtual image field), then an error bit gets set that is queryable by `get_marking_info` (bit # 9 = 1). If, during list processing, it is necessary to immediately respond to this error, then you can call the list command `if_not_activated` to possibly jump to an error-handling routine.

8.7.8 Encoder Resets

Many encoder-based Processing-on-the-fly applications (e.g. a conveyor belt continuously traveling in the same direction) need to occasionally reset the encoders (e.g. after an interruption by `wait_for_encoder`, before a new marking sequence etc.). For this, you can integrate Processing-on-the-fly reactivations into your lists by `set_fly_x`, `set_fly_y`, `set_fly_rot` or `set_fly_2d`.

In contrary, encoder-based Processing-on-the-fly applications for continuous marking (e.g. using an XY stage) should usually retain the relation between encoder values and absolute positions of the XY translation stage. For this purpose, you should use the `set_fly_2d` command to activate Processing-on-the-fly correction. Before a long interruption (by `wait_for_encoder` etc.), you can then deactivate Processing-on-the-fly correction with `fly_return`. And after the interruption you could use the command `activate_fly_2d` to resume correction (see also chapter 8.7.4 "Compensation of 2D Motions", page 206 and chapter 8.7.7 "Synchronization of Processing-on-the-fly Applications", page 211). Processing-on-the-fly correction can also be resumed with `activate_fly_xy`, but some functions are no longer available, particularly those necessary for the relation between current encoder values and absolute XY stage positions (see the section "2D Encoder Compensation for XY Stages", page 206).

8.7.9 Monitoring Processing-on-the-fly Corrections

If a Processing-on-the-fly user program is not optimized for the motion of the workpiece or scan system or if considerable unintended change of workpiece or scan system speed occurs during a Processing-on-the-fly operation, then the image field's boundaries might be reached. Because the RTC5 clips coordinate values at the boundaries to prevent unallowed values, this could cause some parts of the to-be-marked pattern to not be scanned.

To allow user programs to monitor Processing-on-the-fly applications, the RTC5 sets internal error bits (#0...3) if the image field boundaries are exceeded (and coordinate values get clipped). You can query these internal error bits by `get_marking_info`.

To avoid boundary exceedance, you should repeatedly call `get_marking_info` during test runs or normal operation of your Processing-on-the-fly application and modify your user program accordingly.

Notes

- Error bits #0...3 can also be used for determining which edge of the image field was exceeded. Each boundary exceedance results in setting of the corresponding error bit.
- Error bits #0...3 are reset during initialization (by `load_program_file`) and by the command `get_marking_info`. `get_marking_info` therefore returns information about errors that occurred since the last initialization or the last call of `get_marking_info`.
- The RTC5 sets an error bit if boundary exceedance is detected when calculating the output value from the specified coordinate values following application of Processing-on-the-fly correction. Any coordinate transformations, image field corrections and compensating gain and offset corrections applied during calculation of output values after Processing-on-the-fly correction (see page 141) are not taken into account here. During the adjustment phase of a Processing-on-

the-fly correction, you must therefore always ensure that coordinate points approaching the image field boundaries are checked for applicable limits.

- Error bit #9 indicates if the 24-bit virtual image field range was exceeded during resumption of Processing-on-the-fly correction by **activate_fly_2d** or **activate_fly_xy** (see chapter 8.7.7, page 211).

Customer-Defined Monitoring Area and Conditional Command Execution (as of Version DLL 525, OUT 527)

For Processing-on-the-fly applications, the RTC5 also checks for exceedance of a second value range whose boundaries can be specified by **set_fly_limits**. Such exceedances likewise result in setting internal error bits that can be queried by **get_marking_info** (error bits#4...7).

Moreover, the conditional commands **if_fly_x_overflow**, **if_fly_y_overflow**, **if_not_fly_x_overflow** or **if_not_fly_y_overflow** allow execution of any list command to be made dependent upon whether boundary exceedance in a customer-defined monitoring area occurred or not. The conditional commands have no effect if the condition (on error bits#4...7) specified as a command parameter is fulfilled (or not fulfilled). Otherwise, they result in skipping the next list command.

Notes

- Boundary exceedance of a customer-defined monitoring area does not necessarily result in clipping of the output coordinate values. Clipping (and setting error bits #0...3) only occurs if the maximum image field (-524288 ... 524287 bit) would be exceeded (the customer-defined monitoring area is typically smaller).
- Error bits #4...7 are reset during initialization (by **load_program_file**), but *not* by the command **get_marking_info**. Individual error bits, if applicable, get implicitly reset by the conditional commands and can also be explicitly reset by the command **clear_fly_overflow** (see command description).
- Error bits #4...7 (as do error bits #0...3) take into account neither coordinate transformations, image field corrections, compensating gain nor offset corrections.
- Monitoring of a rectangular area is also possible for Rotation-Fly applications, but monitoring of rotation angle areas is not possible.

8.7.10 Tracking Error Compensation of Encoder Values for Processing-on-the-fly Applications

Tracking errors of a scan system's galvanometer scanners can result in a certain amount of positioning inaccuracy, particularly for Processing-on-the-fly applications with variable encoder speeds.

Accordingly, you can activate tracking error compensation by `set_fly_tracking_error` for applications requiring maximum precision.

8.7.11 Processing-on-the-fly Correction for the Z-Axis — “FlyZ Correction” (as of Version DLL 530, OUT 531)

As of version DLL 530, OUT 531 the `set_fly_z` command lets you activate an encoder-based Processing-on-the-fly correction for the Z-axis (“FlyZ correction”).

You can add FlyZ to Processing-on-the-fly correction for the X and Y axes that had been activated by `set_fly_x/set_fly_y/set_fly_rot`. This makes dynamic marking possible for a scan head situated obliquely to workpiece motion (or, marking of workpieces moving on a surface skewed with respect to the scan head's working plane). With FlyZ correction activated, the workpiece motion's Z component (that is, the Z component in the coordinate system of the scan system) gets compensated by re-adjustment of the Z axis (varioSCAN dynamic focusing unit) in accordance with the current encoder signals.

Prerequisites

- On the RTC5 the Processing-on-the-fly option as well as the 3D option must be enabled.
- The desired marking must function properly when static (that is, without workpiece motion):
 - The user program must model workpiece skew by using suitable 3D vector and arc commands.
 - The varioSCAN must be capable of tracking the XY galvanometer scanner motions of the scan head.
- For moving workpieces, the varioSCAN must be capable of tracking workpiece motion. Take care to exceed neither the maximum focus range in Z direction nor the objective's depth of field. Ensure that the correction table's precision is sufficient in terms of edge sharpness, stretching etc. When used with `set_fly_rot`, ensure that the rotation angle across the image field is small enough and the rotation center is sufficiently far away. Users are responsible for appropriate testing. SCANLAB provides no additional functionality for this. A virtual image field for Z coordinates does *not* exist.

- See also the operational prerequisites for 3-axis scan systems in [chapter 8.6.2 "Connection and Initialization"](#), page 193.

Notes on Usage

- For general information on Processing-on-the-fly correction and determining scaling factors, see [chapter 8.7 "Processing-on-the-fly \(Optional\)"](#), page 199.
- When activating FlyZ correction by `set_fly_z`, you can specify which of the two encoder counters should be used. Because `set_fly_x` already uses encoder counter Encoder0 and `set_fly_y` uses encoder counter Encoder1 (`set_fly_rot` uses Encoder0), you might need to use one of the two encoders twice.
- FlyZ correction can be applied together with `set_fly_x/set_fly_y/set_fly_rot`.
- FlyZ correction can also be used alone (only encoder-based Z variation).
- Activated FlyZ correction can be deactivated by `fly_return_z` or `fly_return`. As a result, all Processing-on-the-fly corrections for all three spatial directions are simultaneously deactivated. You can supply a command parameter for a new output position (only the X and Y coordinates for `fly_return`, in addition the Z coordinate for `fly_return_z`).
- Activated FlyZ correction can also be deactivated by `set_fly_z` in conjunction with an invalid parameter (for example, `set_fly_z(ScaleZ=0)`).
 - If only Z correction was activated here, then a jump to an uncorrected output position might occur.
 - If Processing-on-the-fly corrections were also activated for other spatial directions (by `set_fly_x/set_fly_y/set_fly_rot`), then these do not get deactivated here (and no jump to an uncorrected output position occurs).
- If correction for all three spatial directions is activated by `set_fly_x`, `set_fly_y` and `set_fly_z`, then a subsequent `set_fly_x(ScaleX=0)` only deactivates X correction without affecting Y and Z correction (likewise for `set_fly_y(ScaleY=0)`). But if only Z correction (by `set_fly_z`) or two-dimensional correction (by `set_fly_z` and `set_fly_x` or `set_fly_y`) is activated, then a subsequent `set_fly_x`, `set_fly_y` or `set_fly_rot` with invalid parameter value (for example, `set_fly_x(ScaleX=0)`) also deactivates Z correction. In the latter case, a jump to a (partially) uncorrected output position might occur.
- `set_fly_z` deactivates Processing-on-the-fly correction previously activated by `set_fly_x_pos`, `set_fly_y_pos`, `set_fly_rot_pos`, `set_mcbsp_in` or `set_mcbsp_in_list`.
- Conversely, Processing-on-the-fly correction activated by `set_fly_z` gets deactivated by `set_fly_x_pos`, `set_fly_y_pos`, `set_fly_rot_pos`, `set_mcbsp_in` and `set_mcbsp_in_list`. Here, a hard jump to a new output position might occur.
- To avoid hard jumps, use only `fly_return_z` to deactivate Processing-on-the-fly correction.
- `get_marking_info` also provides information on possible range exceedances during FlyZ correction (bit #22...bit #25).
- The commands `set_fly_limits_z`, `if_fly_z_overflow` and `if_not_fly_z_overflow` are available for defining a customer-specific monitoring range for FlyZ correction and for corresponding conditional command execution, see [chapter 9.3.2 "Conditional Command Execution"](#), page 244. You can use `clear_fly_overflow` to reset the error bits (bit #24...bit #25 from `get_marking_info`) for customer-specific monitoring of FlyZ applications.

8.8 Pixel Output Mode – Scanning Raster Images (Bitmaps)

The vector commands described in [chapter 7.1](#) are intended for scanning vector based images. However, the RTC5 also allows reproduction of raster images (or bitmaps). That means black-and-white images or greyscale images can be created with a suitably prepared laser. Furthermore, raster and vector based images can be combined as desired.

8.8.1 Principle Of Operation

A raster image is created line by line, where each line consists of a number of equidistant pixels. A line is reproduced in a single scan. During this scan, the laser focus moves – as with a normal mark command – at an approximately constant velocity along the entire image line (the motion is microvectorized). The individual pixels are marked in passing: each pixel receives a laser pulse at the appropriate location. By varying the laser energy from pixel to pixel, greyscale images are produced (black-and-white images are also possible as a special case). For controlling the laser, pulse lengths (digital) and voltage levels (analog) are outputted.

Notes

- The pixel output mode can be combined with Processing-on-the-fly (see [page 199](#)). It cannot be combined with Sky Writing (see [page 127](#)) or Wobbel (see [page 189](#)).

8.8.2 Software Commands

Before starting an image line, you should execute a jump command to the line's start position.

At the beginning of each image line, the pixel output mode is activated with the command `set_pixel_line` or `set_pixel_line_3d`. The pixel distance and pixel output period (and, resultingly, the speed at which the image line is traversed) are simultaneously set as well. The pixel output period is defined by the parameter `HalfPeriod` (*half* pixel output period). This is also half the laser period. The pixel distance between two adjacent pixels in the line – and thus also the marking direction – is defined by a 2D vector (dx, dy) by `set_pixel_line` or (for pixel marking on sloped surfaces) by a 3D vector (dx, dy, dz) using `set_pixel_line_3d`. The `set_pixel_line`/`set_pixel_line_3d` commands are also used to define which of the two analog ports (ANALOG OUT1 or ANALOG OUT2) should output the analog voltage levels of subsequent `set_pixel` or `set_n_pixel` commands.

Directly after the `set_pixel_line`/`set_pixel_line_3d` command, `set_pixel` has to be called separately for each of the line's image elements. This defines the laser energies to be discharged at the corresponding pixel locations: a pulse duration and/or a 12-bit analog voltage level can be specified (see "[Laser Control](#)" on page 218). Pixel pulses are outputted at the LASER1 port, analog voltage levels at either the ANALOG OUT1 or ANALOG OUT2 analog port (see above).

To specify an identical analog voltage level or pulse duration for multiple (*n*) directly successive image elements in an image line, you can use the `set_n_pixel` command instead of (*n*) `set_pixel` commands. Then only one command is stored on the board, but the list executes an appropriate `set_pixel` command *n* times. Particularly for black & white images, this can drastically reduce the size of lists. Do not confuse `set_n_pixel` with the `n_set_pixel` command (multi-board version of the `set_pixel` command).

Prior to the end of an image line, no command other than `set_pixel` or `set_n_pixel` must be written into the list. After `set_pixel_line`/`set_pixel_line_3d`, the first list command that is *not* a `set_pixel` or `set_n_pixel` command stops the pixel output mode and thus processing of the image line. Each `set_pixel`/`set_n_pixel` command that does not follow

another `set_pixel`/`set_n_pixel` or `set_pixel_line`/`set_pixel_line_3d` command is ignored during processing and is thus a short list command (see [page 250](#)).

Notes

- The commands `set_pixel_line`, `set_pixel_line_3d`, `set_pixel` and `set_n_pixel` are list commands, i.e. they are written into a list.
- The number of pixels in an image line is limited only by the capacity of the RTC5 list buffer (see [page 651](#)). It is suggested – especially for large bitmaps – to set up a new list for each image line to avoid a list change during the execution of one line.
- Each image line must start with a `set_pixel_line` or `set_pixel_line_3d` command.
- The pixel distance (dX, dY) in the X and Y directions (in bits) (and with `set_pixel_line_3d` also dZ) can be specified with floating point numbers. This allows scaling and rotating the image without rounding errors.
- The (half) pixel output period can be any integer multiple of $1/64 \mu\text{s}$ (but at least $104/64 \mu\text{s}$) and is independent from the $10 \mu\text{s}$ output period of the galvanometer scanners' microvectorized motion. Very low pixel frequencies result in multiple scanner steps per pixel, higher frequencies can result in multiple pixel pulses or voltage level changes per scanner step.
- You can implement a pixel output mode in a $10 \mu\text{s}$ raster with variable speed and/or curvilinear paths with the help of microvector commands (see [page 221](#)).

8.8.3 Laser Control

Depending on the type of laser employed, the laser energy discharged at each pixel position can be varied by the laser pulse length or the laser power per pixel:

Variation Of Laser Pulse Length

The command `set_pixel` defines – for each pixel – the length of the pixel pulse (in units of $1/64 \mu\text{s}$), which is outputted at the LASER1 port (for synchronization see "[Timing](#)", [page 219](#)). The command `set_n_pixel` defines the pixel pulse length for n directly successive pixels of an image line.

Variation Of Laser Power

The command `set_pixel` allows specification of a 12-bit analog voltage level for each pixel. The value is transferred either to the ANALOG OUT1 port or to the ANALOG OUT2 port of the RTC5 (see above, for synchronization see "[Timing](#)", [page 219](#)). The command `set_n_pixel` defines the analog voltage level for n directly successive pixels of an image line.

Notes

- It is recommended that some experiments be performed to determine an appropriate gradation curve for producing smooth greyscales. The resulting pixel "colors" (greyscale values) strongly depend on the employed material and the laser.
- For an active pixel output mode:
 - The LaserOn signal remains continuously switched on.
 - In CO_2 mode, the previously set period and pulse length values are overwritten and the pixel pulses are outputted at the LASER1 port as well as (phase shifted) at the LASER2 port.
 - In YAG mode, only the previously defined values for the Q-Switch delay and/or the duration of the FirstPulseKiller signal are considered (see "[Timing](#)", [page 219](#)); the set period and pulse length values are overwritten; the pixel pulses are outputted at the LASER1 port and the FirstPulseKiller signal at the LASER2 port.

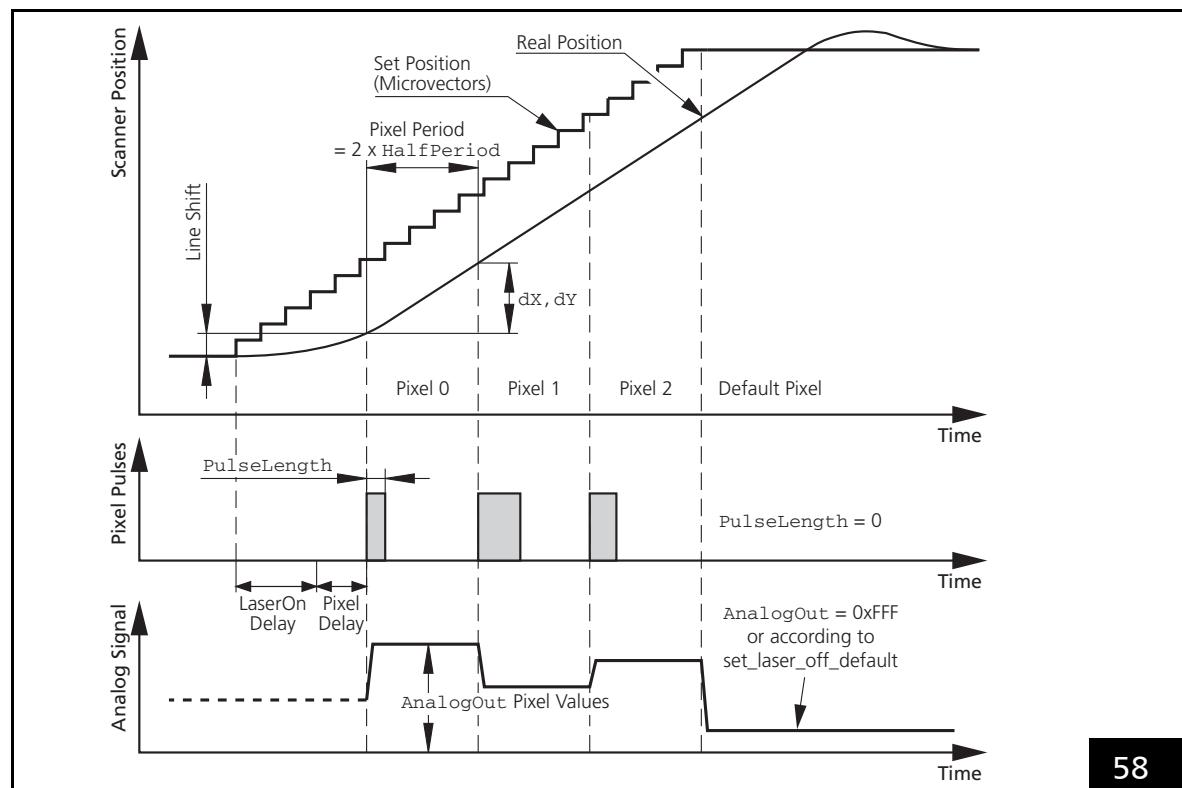
- In laser mode 4 and laser mode 6, the set free-wheeling standby pulses (i.e. no pixel pulses) are, as before, outputted at the LASER1 port and in laser mode 4 the FirstPulseKiller signal at the LASER1 port.
- No softstarts (see [page 152](#)) are performed.
- Sky Writing (see [page 127](#)) is not possible.
- If pixel output mode is deactivated (for example, even during a horizontal retrace jump between the processing of two image lines), then standby pulses (if previously activated) are outputted.
- The period and pulse length settings for normal marking must again be newly defined after the pixel output mode.

8.8.4 Timing

[Figure 58](#) shows the pixel output timing diagram for an image line with 3 pixels.

The movement of the galvanometer scanners is started with the first `set_pixel` or `set_n_pixel` command. The image line is traversed just like a normal mark command (the motion is microvectorized). After an initial acceleration phase, the scanners move with an approximately constant velocity – according to the specified half pixel output period `HalfPeriod` and the pixel distance (dx, dy) – along the entire image line.

At the end of the image line (before the last non-`set_pixel/set_n_pixel` command that follows a `set_pixel/set_n_pixel` command, thus ending the pixel output mode) a default pixel is outputted. The scanners thereby move with equal speed beyond the actual end point. 10 μ s clock cycles that are begun are executed to completion (particularly for pixel frequencies > 100 kHz).



58

Timing of the scanner positions and the laser control signals in the pixel output mode.
The pixel output period in this example is approx. 4.5 microsteps.

The `set_pixel_line` command switches off the “laser active” laser control signals at the beginning of an image line (a LaserOff delay is waited for, if necessary, before switching off – e.g. when a preceding mark command is still being processed). The “laser active” laser control signals are then switched on with the first `set_pixel/set_n_pixel` command, but delayed by a LaserOn Delay with respect to the start of scanner motion. The pixel pulses are outputted possibly (in accordance with the previously activated laser version and its set values) delayed by an additional Pixel Delay (see [figure 58](#)).

The Pixel Delay is

- = 0 in CO₂ mode
- = 0 (or Q-Switch delay) in YAG mode 1
- = length of the FirstPulseKiller signal (or Q-Switch delay) in YAG mode 2
- = 10 µs (or Q-Switch delay) in YAG mode 3
- = Q-Switch delay in YAG mode 5.

The analog signal (at the ANALOG OUT1 or ANALOG OUT2 port) changes synchronously with the leading edge of each pixel pulse (see [figure 58](#)).

Note: The DA converter requires about 1.5 µs ... 3 µs to produce a stable analog output signal. For pixel frequencies above around 100 kHz (i.e. for a Half-Period < approx. 320) digital-to-analog conversion cannot always be fully completed. For such pixel frequencies, users must carefully verify that sufficient capability is available.

The pixel pulse delay or analog level change relative to the scanner motion (i.e. the LaserOn delay and pixel delay) should be set in such a way that the galvanometer scanners’ acceleration phase (in which the scanners are brought to a constant speed) is masked. Alternatively, distortions during the acceleration phase can be suppressed by inserting some idle pixels (e.g. with zero pulse length) at the beginning of each image line.

The entire image line can be slightly shifted due to the lag between the set position and the real position and due to the masked acceleration phase (see “Line Shift” in [figure 58](#)). This can be compensated by adjusting the start position of each image line (backward-extending the image line).

At the end of the image line (at the beginning of the default pixel), the pixel pulse length and the analog output value is set to a default value, each. The default pixel pulse length value is 0 unless another value was defined by `set_default_pixel` or `set_default_pixel_list`. The default analog output value is 0xFF (=4095) unless another value was defined by `set_port_default` or `set_laser_off_default`. The “laser active” laser control signals are then switched off without a LaserOff delay during the next 10 µs clock cycle that follows the default pixel.

No scanner delay is inserted at the end of a pixel line because its value depends on the marking speed indirectly determined by the `HalfPeriod` and `(dx,dy)`. If needed, however, you can insert an appropriate `long_delay` after the pixel line’s last `set_pixel` or `set_n_pixel` command.

Notes

- The (half) pixel output period `HalfPeriod` should be adjusted to the longest required pulse length (for `HalfPeriod` < `PulseLength`/2, the pixel pulse only ends in the next pixel; therefore, the laser is not switched off between pixels). See also the prior comment on the DA converter.
- The (half) pixel output period `HalfPeriod`, the pixel distance `(dx,dY)` and (thus indirectly) the marking speed should be optimized for the dynamics of the scan system.
- The LaserOn delay should be appropriately set to take account of the scan system’s tracking error.
- In laser mode 4 and laser mode 6, the pixel pulses are not outputted at the LASER1 port (which instead outputs freewheeling standby pulses). Even so, they are internally generated and the analog voltage level changes synchronously.
- Output synchronization cannot be used together with pixel mode. You should first deactivate output synchronization.

8.9 Microvector Commands

The microvector commands `micro_vector_abs` and `micro_vector_rel` (and the corresponding 3D commands `micro_vector_abs_3d` and `micro_vector_rel_3d`) move the galvanometer scanners directly to the specified position by a hard jump. Here, the invoked vectors (unlike normal vector commands such as `jump_abs` or `mark_abs`) do *not* get subdivided by the RTC5 into microvectors, but instead execute within a single 10 µs clock period.

The microvector command parameters `LasOn` and `LasOff` let you individually switch the “laser active” laser control signals on and off with the specified delays. The laser delays defined by `set_laser_delays` are not overwritten. They remain valid for normal jump and mark commands.

Thus, you can mark lines and arcs by microvector commands to any desired degree of step fineness or coarseness and even using variable speed, independently of the currently defined jump or mark speeds. You can also thereby vary the line width within a line under steady laser power or implement a pixel output mode within the 10 µs raster with variable speed and/or a curvilinear path.

Notes

- Though microvector commands may pause for preceding scanner delays (e.g. the scanner delay of a preceding `jump_abs` command), they never self-initiate new scanner delays.
- You are solely responsible for appropriately parameterizing the microvectors: you must set appropriately small microvectors for to-be-marked lines, arcs or pixel images and you must incorporate any needed scanner delays to compensate for tracking error (e.g. by inserting several microvector commands with identical target coordinates). Avoid overlaps between `LaserOn` and `LaserOff`. Furthermore, you should not define new laser delays as long as the previous one has not yet expired (see [page 121](#), but new definition simultaneous to expiry is OK).
- Unlike normal vectors, microvector commands:
 - do not microvectorize vectors, but rather execute the vectors with a single hard jump
 - do not take into account wobble motions enabled by `set_wobble` or `set_wobble_mode` (see [page 189](#)) (but also do not deactivate them)
 - do not take Sky Writing mode into account (see [page 127](#)) (but also do not deactivate it).
- In contrast (as with normal vectors), the following corrections (in the listed order, if previously configured) are taken into account for microvector commands when calculating output values from the specified coordinate values:
 - If the command was called from within an “AbsCall” subroutine, then the supplied coordinate values receives an offset (based on the current coordinates at the time of the subroutine call – see also [page 83](#)).
 - If a Processing-on-the-fly correction was enabled by `set_fly_x`, `set_fly_y`, `set_fly_rot` or `set_fly_x_pos`, `set_fly_y_pos`, `set_fly_rot_pos`, then it is applied to the X and Y coordinates (see [page 199](#)).



- If a coordinate transformation was defined for aligning the scan system to the image field (see **set_matrix**, **set_offset**, **set_scale**, **set_angle** and corresponding list commands), then it is applied to the X and Y coordinates (see [page 183](#)).
 - The Z coordinates of **set_offset_xyz** or **set_offset_xyz_list** and defocusing by **set_defocus** or **set_defocus_list** is taken into account.
 - Coordinate values exceeding the real image field range (into the virtual image field range, see [page 135](#)) are clipped to the edge of the 20-bit real image field range.
 - Correction tables assigned by **select_cor_table** or **select_cor_table_list** result in 2D or 3D image field correction (see [page 136](#)).
 - If automatic self-calibration was enabled by **auto_cal** (see [page 224](#)), then a compensating “Gain” and “Offset” is taken into account when calculating output values (gain and offset correction can also be specified by **set_hi**, see [page 227](#)).
- Overflowing output values are, if necessary, clipped to the edge of the maximum possible range of values.

8.10 Timed Vector and Arc Commands

The normal vector commands (jump and mark commands) and arc commands are processed by the RTC5 in such a way that the laser focus moves along the surface of the image field with a defined speed, the *jump_speed* or the *mark_speed*. This is fine for most laser marking and laser material processing applications.

However, some applications require that each vector or arc command consumes exactly the same amount of *time*, regardless of its spacial length. In this case, it is practical to specify the *duration* of the jump or mark process, rather than the jump or mark speed.

The commands

- `timed_jump_abs`
- `timed_jump_rel`
- `timed_mark_abs`
- `timed_mark_rel`
- `timed_arc_abs`
- `timed_arc_rel`

allow specification of the duration of the vector or arc command with an accuracy of 10 µs (the output period of the microvectors) and in the range from 10 µs to 167772160 µs (≈ 2.8 min).

The jump or mark vector or the arc (but not ellipses) are split up into a number of microsteps that correspond exactly to the specified time, see also [chapter 7.1.2 "Microsteps", page 107](#). Of course, this means that the *jump or mark speed*, i.e. the velocity of the laser focus (and the angular velocity of the movement of the mirrors) depend on the length of the vector or arc.

Notes

- After a timed vector or arc command, a *jump delay*, a *mark delay* or a *polygon delay* is inserted, just like after a vector or arc command. That means the total time for the command is the *sum* of the specified time and the corresponding delay, see also [chapter 7.2.2 "Scanner Delays", page 113](#).
- If the vector-defined laser control is activated, also timed para commands can be used (timed para arc commands are *not* provided):
 - `timed_para_jump_abs`
 - `timed_para_jump_rel`
 - `timed_para_mark_abs`
 - `timed_para_mark_rel`
- If the 3D option is enabled, also timed 3D vector commands can be executed (timed 3D arc commands are *not* provided):
 - `timed_jump_abs_3d`
 - `timed_jump_rel_3d`
 - `timed_mark_abs_3d`
 - `timed_mark_rel_3d`.
- If additionally the vector-defined laser control is activated, also timed 3D para commands can be used:
 - `timed_para_jump_abs_3d`
 - `timed_para_jump_rel_3d`
 - `timed_para_mark_abs_3d`
 - `timed_para_mark_rel_3d`
- As of RTC5OUT.out version 526, even the execution times of timed marking commands (but not timed jump commands) with zero *spacial length* achieve the specified value (if the specified time > 5 µs) and these marking commands also behave like timed marking commands of finite spacial length with respect to laser control signals and mark/polygon delays (see also [page 116](#)). This lets you easily synchronize Processing-on-the-fly application timing even where Processing-on-the-fly compensation – from, for example, robot arm motion – has already been calculated into the marking coordinates. Another use is for separately marking “points” within a polyline.

8.11 Automatic Self-Calibration

8.11.1 Using for Drift Compensation

Long-term repeatability is very important in many applications, e.g. for rapid prototyping in which the processing operation can span several hours. For such laser applications, the galvanometer scanner's long-term drift and temperature drift, which manifest as a shift (offset drift) and increase or decrease in the size (gain drift) of the working image field, can exceed the allowed tolerances.

In such applications, it is helpful to start up the application only after the scanners have reached their operating temperature. In addition, the magnitudes of environmental fluctuations (e.g. operating temperature changes to which the scan system is exposed) should be kept as small as possible and the scan system preferably operated with a constant load.

For higher long-term repeatability requirements, SCANLAB scan systems can be (optionally for apertures $\geq 10\text{ mm}$) equipped with an additional internal sensor system for automatic self-calibration (ASC sensor system, Home-In sensors). Together with associated commands furnished by the RTC5, this reference system makes it possible to calibrate the position detectors of the galvanometer scanners at any desired time. Thus, the effects of gain and offset drift can be reliably compensated and positioning accuracy is maintainable over long periods of time. Remaining long-term drift effects are the same order of magnitude as short-term repeatability.

8.11.2 How it Works

The `auto_cal` command starts a measurement routine for determining the exact control values for reference positions (Home-In positions) defined by the internal sensor system. For this purpose, the RTC5 drives the galvanometer scanners to the appropriate positions within the scanning range.

For drift *measurement*, the routine should be executed

- when setting up the equipment (to determine reference values for the Home-In positions) and
- during an application's execution at appropriate time intervals (to determine if and how the Home-In positions have changed).

To *compensate* the drift, appropriate gain and offset values can be calculated and set separately for each galvanometer scanner based on the determined deviations between the current Home-In position and the reference value. All subsequent vector or arc commands for any points within the image field are then corrected (transformed) in accordance with these new gain and offset values.

The RTC5 provides commands for automatic self-calibration as well as customer-specific calibration. For automatic self-calibration, the RTC5 automatically determines and sets the gain and offset values.



Notes

- Prior to performing a measurement routine, you can use **auto_cal**(Command = 4) to check if the attached scan system in fact has an internal sensor system for automatic self-calibration (Home-In sensors) and if the sensor system is functioning properly. This ASC hardware check also occurs automatically by **auto_cal**(Command = 0) and sometimes for **auto_cal**(Command = 1 and 3) (see also **auto_cal** command description and **get_auto_cal**).
- During execution of the measurement routine determining the Home-In positions, the laser should be switched off and no other commands are transferred to the scan system.
- For Gain/Offset Correction:
Control values corrected by the applied image field correction table (*.CT5) are multiplied by the currently set gain value and then additively modified by the offset value. The final positions output to the galvanometer scanners are automatically clipped to the allowed range. Depending on the current XY coordinates, the limits can be reached even with gain and offset values within the allowed range. The gain and offset values set after initialization by **load_program_file** (Gain = 1.0 and Offset = 0) do not modify the control values for vector or arc commands.

- For iDRIVE scan systems (intelliSCAN, intelliSCAN_{de}, intelliDRILL, intellicube, intelliWELD), the following also applies (as of DLL 513):
 - At the beginning of a measurement routine, the XY2-100 status word is automatically selected to be returned from the scan system. At the end of the measurement routine, the previously set data type is restored.
 - If you intend to call **auto_cal** for the primary scan head connector, but automatic laser control was previously activated in mode 2 (whereby the “actual velocity” is automatically selected to be returned from the scan system), then **auto_cal** aborts with an error (return value 7). Here, you should deactivate automatic laser control before performing automatic self-calibration.
 - **auto_cal** also aborts (return value 1, 10 or 11) if the scale factor was previously set by **control_command**(Data = 12xx_H) to a value < 1. This is because the sensor positions then are not reachable (see also **control_command**(Data = 053F_H)). Here, you must therefore set the scale factor to 1 (default setting) by **control_command**(Data = 1283_H) and **control_command**(Data = 1200_H) prior to automatic self-calibration.

8.11.3 Determining Reference Values

Reference values can be determined by **auto_cal** (Command = 0). This starts an ASC hardware check and the measurement routine for determining the current Home-In positions. The determined reference values are stored – separately for the X and Y axes – in the DLL. Additionally, the reference values are stored in non-volatile memory (EEPROM) on the RTC5 Board. Thus, the reference values remain available for later calibration – even after a reset or initialization by **load_program_file** (the latter also applies to the detected ASC hardware type).

If – afterwards – automatic self-calibration shall be performed, then Command = 0 should be previously used for reference determination – here, the galvanometer scanners' gain and offset values are automatically set to their initialization values (Gain = 1.0, Offset = 0).

Notes

- After **auto_cal**(Command = 0), the galvanometer scanners are in the same position as before the command, though possibly corrected for changed offset/gain settings.
- Reference positions should be determined during adjustment of the equipment, e.g. when the scan system is installed or exchanged and subsequent calibration of the image field needs to be performed by test-pattern measurements. Reference values should also be newly determined when a major change in usage conditions necessitates new image field measurements.
- Reference positions should be determined under conditions (ambient temperature, load) typical for the application and after the overall system has fully attained its operating temperature. The reference positions should always be determined only after a warm-up time of more than 20 minutes and not before the TempOK signal has been activated.
- Execution of **auto_cal**(Command = 0) typically lasts up to 10 seconds.

8.11.4 Calibration During the Application

Automatic Self-Calibration

Automatic self-calibration of the scan system during an application can be executed with **auto_cal**(Command = 1). Here, a measurement routine is started for determining the current Home-In positions. From the resulting deviation compared to the previously (with Command = 0) stored reference value, the RTC5 calculates and sets gain and offset values – separately for the X and Y axis. This resulting drift compensation setting has immediate effect on the current output positions and all subsequent ones. It remains in effect until **auto_cal**(Command = 1) is called again or until drift compensation is shut off with **auto_cal**(Command = 2) (whereby the settings Gain = 1.0 and Offset = 0 take effect).

Notes

- In principle, after **auto_cal**(Command = 0, 1 and 2), the galvanometer scanners are in the same position as before the command. However, a newly calculated or set drift compensation might lead to correction of the position.
- After a reset or initialization (by **load_program_file**), automatic drift compensation is switched off – as with **auto_cal** (Command = 2). Nevertheless, previously determined reference values remain available.
- The calibration routine started with **auto_cal**(Command = 1) typically lasts 1-2 seconds (depending on the magnitude of the drift). But if **auto_cal**(Command = 1) is not preceded by **auto_cal**(Command = 0 or 4), then an automatic ASC hardware check is performed, which can extend the command's execution time by a few seconds.

Customer-Specific Calibration

Automatic self-calibration is midpoint centered, so that the image field's center remains stable. If an alternative is preferred (e.g. if the left edge should remain stable, size is irrelevant), then the calibration can also be externally calculated and set. For such a customer-specific calibration, the **get_hi_pos** command can be used to query the Home-In position that was detected and stored in the DLL with **auto_cal**(Command = 0, 1 or 3). The drift can be determined by repeated calls of the commands **auto_cal**(Command = 3) and **get_hi_pos**. From this, compensating gain and offset factors can be calculated and then set by **set_hi**. The resulting drift compensation setting has effect immediately on the current output position and all subsequent ones.

Notes

- **get_hi_pos** returns the current (last determined) Home-In positions. Directly after **auto_cal**(Command = 0), these are also the last determined and stored Home-In reference values. Directly after initialization (**init_rtc5_dll**), **get_hi_pos** returns the reference values from the EEPROM (see also **get_hi_pos** command description).
- After **set_hi**, a correction of the current position might occur at jump speed.
- Gain and offset correction factors can also be set by **set_hi** for systems *without* Home-In sensors.
- After **auto_cal**(Command = 3), the galvanometer scanners are in exactly the same position as before the command.
- Under certain circumstances the Home-In reference positions (not Home-In positions) of a scan head can be transferred from one RTC5 Board to another with **write_hi_pos**.

Supplemental Information about Calibration

- The accuracy of fit of the set drift compensation can decrease with increasing time. Therefore, calibration should be repeated after appropriate time intervals. The shorter the time interval between individual calibrations, the higher the attained long-term repeatability. Time intervals are typically in the range of minutes. Events such as workpiece changes or line feeds are ideal opportunities for conducting a new calibration.
- The accuracy of fit of the calibration, and the thereby attained long-term repeatability, are further enhanced by steady environmental and load conditions.
- If an error occurs (e.g. due to excessive spread during a measurement cycle, see below) during the measurement routine for determining the Home-In positions after **auto_cal**(Command = 0, 1 or 3), then no reference values are stored for the affected axis (Command = 0) and the gain and offset factors remain unchanged (Command = 1; in contrast, Command = 0 always initializes gain and offset). Then **get_hi_pos** returns 0 instead of faulty values.
- The measurement routine determines the Home-In positions during a cycle of several measurements. If deviations between the individual measurements are too large (maximum – minimum > 96 bits), the measurement routine aborts and an error (return value 2) is returned. An error within an individual measurement cycle might indicate that stable temperature conditions have yet to be attained or that a brief mechanical or electrical disturbance has occurred. If significant spreading occurs within an individual measurement cycle, we recommend the following:
 - either a further measurement cycle is immediately conducted or
 - the error is initially ignored while using the current gain and offset values until new correction values are determined by the next (successful) measurement cycle.



If significant spreading occurs across several measurement cycles, then it might be appropriate to halt the application until one of the subsequent measurement cycles is successful.

Significant spreading across several measurement cycles and over a long time period might indicate a defect in the internal sensor system or another part of the scan system. But because continuous (mechanical or electrical) external disturbances or contamination can also impair automatic self-calibration, the scan system and its environment should in such cases be appropriately inspected to assess overall functionality.

- Certain hardware error states (e.g. signal faulty or not found) get permanently stored in the EEPROM. After correcting the error, you must call **auto_cal** with `Command = 0` or `4` to clear the error state. Until this time, correction with `Command = 1` or `3` is not possible.

8.12 Camming

The **camming** command produces a marking that simulates the classic camshaft action of moving a valve tappet – or more generally a cam disk moving a lever.

The galvanometer scanner motion here is a lever movement defined as a two-dimensional curve. It is written in a list as a closed point-by-point sequence of **mark_abs** commands. The “resolution” of the curve’s (i.e. the distance between two points) is freely selectable – the finer the better.

The entire curve must fit within a contiguous list region, see **config_list**. Though it is not possible to switch among lists to load further portions of the curve.

“Propulsion” is furnished either by external fed in encoder pulses or by internally simulated ones.

Each outputted point is derived from the XY coordinate of a **mark_abs** command at the position

`FirstPos + Index, whereby Index = Round((EncoderCurrent - EncoderStart) × Scale).`

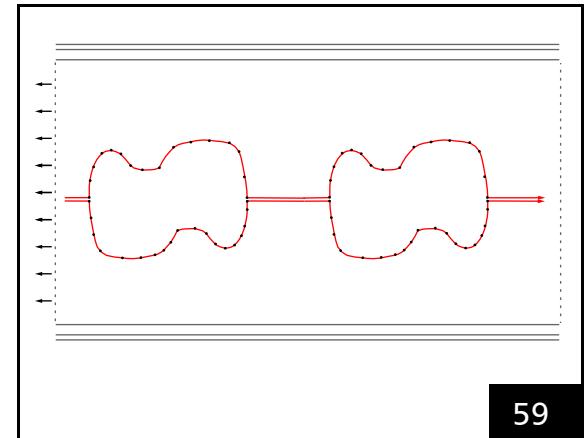
`FirstPos` is the first **mark_abs** command’s list position and `Scale` is a freely selectable scale factor. `EncoderStart` gets automatically determined when **camming** is called. There is no automatic encoder reset. The first outputted point is always `Index = 0`.

The individual points are executed every 10 µs as hard jumps without scanner delays. That is, **mark_abs** does not get microvectorized, see

micro_vector_abs. Here, `Scale` regulates how precisely the curve is sampled. The larger `Scale` is, the coarser is the piecewise linear approximation of the curve.

The number of encoder pulses per 10 µs clock period and the spacing of the points determine the actual marking speed.

Choose “resolution”, `Scale` and encoder speed appropriate for your scan head performance characteristics.



59

Camming example. A transport system moves a continuous workpiece. Two scan heads team up to mark the contours. Schematic depiction.

The camming process can be controlled in various ways, see **camming** command description:

Users can take over laser control for themselves (`Ctrl > 0`), for example, by using **laser_signal_on_list** before **camming** and **laser_signal_off_list** after, or the RTC5 can automatically switch the laser (`Ctrl = 0`) in accordance with laser delays (as with a normal polyline).

The curve can be executed once and then automatically ended (`Ctrl = 0` or `Ctrl = 1`). The list then continues by executing the next command that follows the end of the point list (the length of the point list is defined by `Npos` in the **camming** command parameter list).

In accordance with encoder direction point lists can also run backward. Then `Index = 0` ends the curve.

The curve can be repeated indefinitely (`Ctrl = 2` or `Ctrl = 3`). To avoid hard jumps at the start of a repeat, the point list shall represent a closed curve. To terminate indefinite repeating, a cancellation with **stop_execution** or **/STOP** is required.

At any time, the curve can be restarted at the address defined by **set_extstartpos** or **set_extstartpos_list** (typically but not necessarily `FirstPos`) by an external /START (independently of the current index, possibly hard-jumped to the first marking position).

Notes:

- Under normal operation external /STARTs are suppressed during list execution which is not the case for the camming process with indefinite repeating!

If `Ctrl = 2`, then the camming process waits at the end of the point list for a new start. If `Ctrl = 3`, then processing automatically starts again from the beginning (the point list is processed as a ring buffer).

Furthermore, the camming process can be combined with Processing-on-the-fly (which can apply its own scaling if different from that used by the **camming** command).

The laser control can be combined with the automatic vector-defined laser control (**set_vector_control**). This requires to use **para_mark_abs** commands instead of **mark_abs** commands. The value defined there is outputted immediately, thus allowing systematic variation of laser power along the curve. If automatic vector-defined laser control is not enabled, then no laser power is outputted by the **para_mark_abs** commands.

Automatic laser control based on encoder speed is also possible.

Alternatively, the commands **mark_abs_3d** or **para_mark_abs_3d** as well as **timed_mark_abs_3d** can be used. However, the Z coordinate and T parameter is ignored during outputting. Other commands within point lists are likewise ignored. Point output then remains unchanged for one clock period.

Notes for Testing

- If a curve point list is finished by **set_end_of_list** and does not cross the boundary between List 1 and List 2 (see [chapter 6.4 "List Handling", page 75](#)), then the curve shape can be tested using a normal list start, for example, by `execute_at_pointer(FirstPos)` (to some extend as a polyline, but with a predefined marking speed and using the currently defined variable polygon delay).
- The point list should be closed with **list_return**, if it is part of a subroutine and a **sub_call** call is used for testing.
- If the test shall include the hard jump, then the command **timed_mark_abs_3d** (with `Time = 10 µs`) can be used as well, but not **timed_para_mark_abs_3d**.

8.13 Time Measurements

RTC5 boards are equipped with an integrated timer, which is referred to as the “RTC5 timer” in the following. The RTC5 timer only counts clock cycles of list commands. Counting is paused during interruptions by `set_wait` or `pause_list`.

As of DLL 543, OUT 543, RBF 524 a “time stamp counter” is additionally available. It starts counting at 0 with `load_program_file` and counts uninterruptible all 10 µs clock periods.

In order to measure the marking time consumed by any particular marking process, `save_and_restart_timer` is called before and then after the marking process. `save_and_restart_timer` saves the present RTC5 timer value and resets it to 0. The elapsed time can then be read by `get_time`, which returns the RTC5 timer value saved during the most recent call of `save_and_restart_timer`.

Using the time stamp counter (available as of DLL 543, OUT 543, RBF 524) an absolute reference to the individual time periods can be established, even if the RTC5 timer has been reset by `save_and_restart_timer`. The time stamp counter can be recorded by the trigger signal 52 (see `set_trigger` or `set_trigger4`) and is read-out by `get_value(52)`.

The present RTC5 timer value can be read out by `get_lap_time`. It returns the elapsed time since the last call of `save_and_restart_timer` but without resetting the RTC5 timer to zero. In this way the interim execution time of lengthy marking processes can be monitored.

Notes

- `get_time` and `get_lap_time` only take list execution times into account (because the RTC5 timer only counts list command clock cycles and pauses during interruptions by `set_wait` or `pause_list`).
- To compare RTC5-internal `save_and_restart_timer` time measurements to external time measurements by the BUSY pin, you should insert a `list_nop` between `save_and_restart_timer` and `set_end_of_list`. This ensures that any scanner delay is processed before `set_end_of_list`. Without `list_nop`, `save_and_restart_timer` includes the scanner delay in its measurement even though it completes only after `set_end_of_list` (and therefore the BUSY pin is already LOW).



9 Programming Peripheral Interfaces

Scan systems are often used in equipment that needs to synchronize processing by the laser and scan system with other process steps (e.g. workpiece placement, robotic motion, process monitoring etc.).

For this purpose, the RTC5 provides a variety of peripheral interfaces (see [page 47](#)).

With the commands for programming these interfaces, you can supplementally and/or synchronously control the following in addition to lasers and scan systems:

- Signals transmitted for peripheral control
- Querying and evaluation of peripheral signals
- Control and synchronization of laser scan processes and peripheral control by external control signals

This is described in the following sections.

9.1 Signal Output

For peripheral control (e.g. controlling a workpiece transport system or a shutter), appropriate signals can be outputted by the interfaces described below.

The output values can be changed at any time by control commands or – during processing of a list – by list commands.

9.1.1 16-Bit Digital Output Port

The EXTENSION 1 socket connector provides a buffered 16-bit digital TTL output (DIGITAL OUT0 ... 15). The level of its output signals must be configured with a jumper (see [page 51](#)).

The `write_io_port_list`, `write_io_port`, `write_io_port_mask_list` and `write_io_port_mask` commands specify the digital output values. The output is in high-impedance mode (tri-state) until an initial value is assigned to it. In addition, the command `set_port_default` (Port = 3) can be used to define the value to be outputted at the 16-bit digital output port, as soon as processing of a list has ended with `stop_execution` or by an external stop signal. The default value also takes effect together with the position- and/or speed-dependent laser control (see `set_port_default`).

If automatic laser control is activated, then the value at the 16-bit digital output may automatically get adjusted (see [page 159](#)). You can log this by `set_trigger/set_trigger4`.

When the output value is changed, a LATCH signal is outputted at the EXTENSION 1 socket connector as a trigger signal for synchronization of data transmission.

The `get_io_status` command reads the current value of the digital output port.

9.1.2 8-Bit Digital Output Port

The EXTENSION 2 socket connector provides a (jumper-configurable) buffered 8-bit digital output port (DATA0 to DATA7) (see [page 52](#)).

Its output values can be set by `write_8bit_port` or `write_8bit_port_list`. The output is in high-impedance mode (tri-state) until an initial value is assigned to it. In addition, the commands `set_port_default` (`Port = 2`) or `set_laser_off_default` can be used to define the value to be outputted at the 8-bit digital output port, as soon as processing of a list has ended with `stop_execution` or by an external stop signal. The default value also takes effect together with the position- and/or speed-dependent laser control (see `set_port_default`).

If automatic laser control is activated, then the value at the 8-bit digital output may automatically get adjusted (see [page 159](#)). You can log this by `set_trigger/set_trigger4`.

When the output value is changed, a LATCH signal is outputted at the EXTENSION 2 socket connector as a trigger signal for synchronization of data transmission (provided that pin (17) has been correspondingly configured by the jumper setting, see [page 52](#)).

9.1.3 2 Bit Digital Output Port

The LASER connector provides a buffered 2-bit digital output port (DIGITAL OUT1 and DIGITAL OUT2) (see [page 48](#)). Its output values can be set by `set_laser_pin_out` or `set_laser_pin_out_list`. The output is in high-impedance mode (tri-state) until an initial value is assigned to it. In addition, the command `set_port_default` (`Port = 4`) can be used to define the value to be outputted at the 2-bit digital output port, as soon as processing of a list has ended with `stop_execution` or by an external stop signal.

9.1.4 12-Bit Analog Output Ports

The LASER connector provides the two 12-bit analog output ports, ANALOG OUT1 and ANALOG OUT2 (see [page 48](#)). The ANALOG OUT2 output port is also available by the MARKING ON THE FLY socket connector (see [page 53](#)).

The output values can be separately set by `write_da_x` or `write_da_x_list`. In addition, the commands `set_port_default` (`Port = 0 or 1`) or `set_laser_off_default` can be used to define the values to be outputted at the 12-bit analog output ports, as soon as processing of a list has ended with `stop_execution` or by an external stop signal. The default value also takes effect together with the position- and/or speed-dependent laser control (see `set_port_default`). If accordingly preset by corresponding commands, the values at the analog output ports are also changed by a softstart (see "[Softstart Mode](#)" on [page 152](#)) or in the pixel mode (see [page 217](#)).

If automatic laser control is activated, then the value at one of the 12-bit analog output ports may automatically get adjusted (see [page 159](#)). You can log this by `set_trigger/set_trigger4`.

9.1.5 Stepper Motor Control

Output Signals

The signals (ENABLE, DIRECTION and CLOCK) for controlling up to two stepper motors are outputted at the "STEPPER MOTOR" socket connector⁽¹⁾:

- You can appropriately change the ENABLE signal (to switch motor current on or off) during initialization by **stepper_init** and afterward by **stepper_enable** or **stepper_enable_list**.
- The RTC5 generates periodic CLOCK signal pulses (during reference movements by **stepper_init** and set-position movements by **stepper_abs** etc.). With each CLOCK pulse, the stepper motor executes a single step. You can adjust the CLOCK signal's pulse period (and thereby the speed of stepper motor motion) during initialization by **stepper_init** and afterward by **stepper_control** or **stepper_control_list** (the period is specified in units of 10 µs cycles).
- You can explicitly set the DIRECTION signal (and thereby the direction of stepper motor motion) during reference movements by **stepper_init**. In contrast, the DIRECTION signal is internally controlled during set-position movements by **stepper_abs** etc.: the signal gets set (to HIGH) if the next CLOCK pulse (in accordance with the defined set-position value) would increase the internal position variable. The DIRECTION signal also remains set for the cycles between two clock periods and even when the stepper motor has reached its set position. Only upon an actual change of direction does the DIRECTION signal correspondingly change in place of a CLOCK pulse. Here, output of the next CLOCK pulse is delayed by a full CLOCK pulse period (undefined truncation of CLOCK pulse periods never occurs).

Notes

- For signal specifications, see [page 58](#).
- For querying signals, see [page 236](#).
- Stepper motor signals are outputted independently of any executing lists. A **set_end_of_list**, **pause_list**, **set_wait**, **stop_execution** or external list stops do not terminate or pause the stepper motor motion.
- For changes of direction or pulse period, the new values do not become active until an already-begun period is complete. Thus, pulse intervals are never be shorter than the currently defined value. For change of direction, an additional empty period (without CLOCK pulse) gets inserted.

(1) Stepper motor signals are available only for RTC5 Boards with DSP version numbers 2 and higher (see **get_rtc_version** bits #16-23). For older RTC5 Boards, stepper motor commands do not execute.

Reference Movements and Position Initialization

With `stepper_init`, you can initiate reference movements to limit switches. Here, the desired direction can be specified by the `Dir` parameter. To ensure that, despite mechanical play or long signal transit times, the reached end position still does not lie beyond the limit switch, you can define a tolerance value `Tol` that moves the stepper motor in the opposite direction after reaching the limit switch.

SCANLAB recommends executing a (fast) reference movement with a short CLOCK pulse period (`Period`) first and then a further (shorter but slower) reference movement with a longer CLOCK pulse period.

A successfully concluded reference movement does set both the internal position variable (for the current position) and the internal set-position value to that defined by the `Pos` parameter. This is used as a reference for future set-position movements. The reached reference position is offset from the limit switch position by $\pm Tol$ (direction dependent).

If `Period = 0` and/or `Dir < 0`, then the reference movement does not occur. Instead, the current stepper motor position becomes the new reference position (with the value newly defined in `Pos` as the position variable and set-position value).

Because `stepper_init` always stops a previously begun stepper motor movement, you could also use this command as an emergency stop for the stepper motor.

Parallel execution of reference movements for both stepper motors is also possible, but cannot be simultaneously started through a single command.

Set-Position Movements

Set-position movements can be initiated by the commands `stepper_abs`, `stepper_rel`, `stepper_abs_no` and `stepper_rel_no` or the corresponding list commands.

Specify absolute set-position values for `_abs` commands and relative values for `_rel` commands (always as CLOCK pulse units). `_no` commands only produce set-position movements for one stepper motor output, the other set-position commands do so for both stepper motor outputs simultaneously.

With `stepper_wait`, you can interrupt further execution of a list until a started stepper motor movement is completed.

The list commands for set-position movements are short list commands. Therefore, a stepper motor movement can also execute synchronously with a galvanometer scanner movement.

If the limit switch activates during a set-position movement, then the movement immediately aborts and cannot be resumed as a normal set-position movement. You either have to request a reference motion or mechanically deactivate the limit switch. If a stepper motor movement aborts once (e.g. also by `Period = 0`), then the existing set position gets overwritten by the current position value. Therefore the stepper motor movement to the original set position cannot be resumed by eliminating the cause of interruption (limit switch or CLOCK pulse period = 0). Instead, it needs to be newly triggered.

To work around this behavior, the consideration of the limit switch can be deactivated by `stepper_disable_switch`. Then, the "release" can occur without carrying-out an initialization movement once again, even if a limit switch is present. The deactivation is especially useful, if a continuously rotating rotary axis is controlled by the stepper motor. During an initialization movement a possible deactivation of a limit switch is not considered.



Querying Signals and Status Values

The current status of stepper motor signals (ENABLE, DIRECTION, CLOCK and SWITCH), the currently defined CLOCK pulse period and the current values of internal position variables for both stepper motors can be queried by `get_stepper_status`.

The `get_stepper_status` command also returns the Busy and Init statuses of both stepper motors. The Init status is set during reference movements and the Busy status during set-position movements.

As long as the Init status is set, no set-position commands (`stepper_abs`, `stepper_rel`, etc.) are permitted; control commands (except `stepper_init`) are denied with a `get_error_return` value of `RTC5_BUSY` and list commands wait until the Init status gets reset. In some circumstances, the list itself or the movement process must be aborted.

Terminating Infinite Movements

Depending on your chosen parameters, sometimes very long or even infinite stepper motor movements can be initiated by `stepper_init`, `stepper_abs`, e.g. if no limit switch exists in the specified direction or if a very large set-position value is combined with a long pulse period.

- If an infinite movement is started by a control command (e.g. `stepper_abs`), then this control command completes at the latest when the positive time (in seconds) supplied for the `WaitTime` parameter has expired. However, the stepper motor's infinite movement itself continues and you might need to separately abort it by setting the CLOCK pulse period (e.g. by the control command `stepper_control`) to 0 (emergency stop) or by defining an appropriate new set position.
- If you start an infinite movement by a list command (e.g. `stepper_abs_list`) and wait for its completion by `stepper_wait`, then further list execution is blocked as long as the infinite movement is not aborted by a control command such as `stepper_control` with `Period = 0` or an appropriate new set position. You could also abort the list by `stop_execution` or an external list stop. But here, too, the stepper motor's infinite movement needs to be separately stopped with `stepper_control(Period = 0)`.



9.1.6 RS232 Interface

The "RS232" socket connector provides an RS232 interface (see [page 54](#)). The interface can be configured by `rs232_config`. `rs232_write_data` can be used to send single data words (bytes) to the RS232 interface. Texts can be sent to the interface by `rs232_write_text` or `rs232_write_text_list`.

9.1.7 McBSP/SPI Interface

At the McBSP/SPI interface, see [chapter 4.4.6 "SPI / I2C Socket Connector", page 54](#), a 32-bit data word every 10 µs at DX0 pin (7) is permanently outputted.

The `set_mcbsp_out` and `set_mcbsp_out_ptr` commands (analogously to `set_trigger`) allow selection of the signal types to be outputted there:

- The command `set_mcbsp_out` lets you specify two signal types for simultaneous output at the McBSP/SPI-interface. A 16-bit portion of the first signal type is packed along with a 16-bit portion of the second signal type into a 32-bit data word for output every 10 µs. For a detailed description, see `set_mcbsp_out`.
- The command `set_mcbsp_out_ptr` lets you define a list of up to 8 signal types. The signals are outputted sequentially in the specified order. For every 10 µs clock cycle, the lower 24 bits of the corresponding data signal and the associated signal type number (8 bits) are packed into a 32-bit data word and outputted at the McBSP/SPI interface. For a detailed description, see `set_mcbsp_out_ptr`.

Notes

- For signals and operating conditions, see [chapter 4.4.6 "SPI / I2C Socket Connector", page 54](#).
- In the default setting, the McBSP/SPI interface always outputs bits #4-19 of the cartesian control values for the X and Y axes (SampleX and SampleY) in a common 32-bit data word. This is equivalent to specifying `set_mcbsp_out(7, 8)`.
- Signals specified by `set_mcbsp_out` or `set_mcbsp_out_ptr` are outputted (possibly sequentially) until you call one of these two commands again.



9.2 Signal Input

Signals of peripherals (e.g. signals of a transport system, workpiece recognition system or process monitoring camera) can be queried by the interfaces described below through control commands at any desired time or – during processing of a list – with list commands.

9.2.1 16-Bit Digital Input

The EXTENSION 1 socket connector provides a 16-bit digital TTL input (DIGITAL IN0 ... 15) (see [page 51](#)) for receiving 16-bit digital values. For synchronization of data transmission, the EXTENSION 1 socket connector also provides a SYNC signal.

The `read_io_port` or `read_io_port_buffer` and `read_io_port_list` commands can be used to read the current value of the digital input port.

Further commands are provided for conditional command execution (see [page 244](#)).

9.2.2 2-Bit Digital Input

For querying 2-bit digital values, the LASER connector provides a 2-bit digital input (DIGITAL IN1 and DIGITAL IN2, see [page 48](#)). Its input value can be read by `get_laser_pin_in`.

Further commands are provided for conditional command execution (see [page 244](#)).

9.2.3 RS232 Interface

The “RS232” socket connector provides an RS232 interface for reading signals (see [page 54](#)). The interface can be configured by `rs232_config`. Data can be read by `rs232_read_data`.

9.2.4 McBSP/SPI Interface

At the McBSP/SPI interface, see [chapter 4.4.6 “SPI / I2C Socket Connector”, page 54](#), the 32-bit data word most recently fully transmitted to the specified memory location can be queried with `read_mcbsp`. It is up to users whether to interpret it as one 32-bit value or two 16-bit values.

Signals (position values) received by the McBSP/SPI interface can also be integrated directly into Processing-on-the-fly correction of workpiece or scan-system motion (see [page 199](#) and [page 248](#)) or can be used for online positioning (see [page 187](#)).

Notes

- For signals and operating conditions, see [chapter 4.4.6 “SPI / I2C Socket Connector”, page 54](#).

9.3 Control by External Signals

The previously-discussed input and output of peripheral signals can be synchronized with scan system and laser control, as described below:

- The necessary list commands can be inserted at appropriate places in command lists.
- Execution of necessary control commands can be made dependent on the current status of list execution. For this, the list status can be requested by [read_status](#) (see [page 76](#)) and the list execution status by [get_status](#) (see [page 77](#)). In addition, the BUSY list execution status is provided by the BUSY OUT signal at the LASER connector ([page 47](#)), at the EXTENSION 1 socket connector ([page 51](#)) and at the MARKING ON THE FLY socket connector ([page 53](#)).

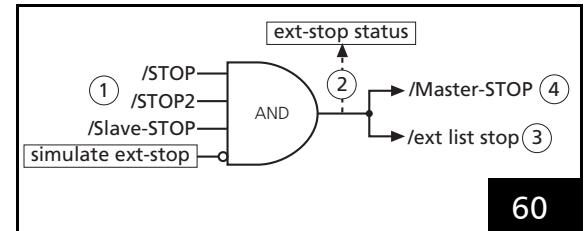
Moreover, the RTC5 provides commands and interfaces (described in the following sections) that allow external control signals (e.g. from a light-barrier or from an encoder) to directly control and synchronize execution of command lists or individual commands (including the output of peripheral signals).

9.3.1 Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization

External List Stop

By a signal at the inputs /STOP, /STOP2 or /Slave STOP, or by the command [simulate_ext_stop](#), an external list stop can be initiated (see (1) and (3) in [figure 60](#)). Like a call of the command [stop_execution](#), this causes execution of the currently running list to be immediately aborted and the “laser active” laser control signals to be switched off (but not deactivated).

Additionally, the 16-bit digital output port of the EXTENSION 1 socket connector, the 8-bit digital output port (DATA0 to DATA7) of the EXTENSION 2 socket connector as well as the 2-bit digital output port and the two 12-bit analog output ports (ANALOG OUT1 and ANALOG OUT2) of the LASER connector are set to the previously defined stop-case values – by [set_port_default](#) – if these were not defined as “-1”.



60

External list stop (see text for description)

The inputs for external stop signals (1) are always unlocked so that an external stop can occur at any time.

The /STOP input is accessible by the 15-pin D-SUB LASER connector (see [page 47](#)), the /STOP2 input at the MARKING ON THE FLY socket connector (see [page 53](#)). Both signal inputs are internally connected to +3.3 V by pull-up resistors (4.7 kΩ). Both inputs are TTL active-LOW and level sensitive. A list abort is triggered as soon as at least one of the two inputs is set to LOW (i.e. to 0 V or ground) for at least 10 µs.

If the RTC5 Board is coupled by the MASTER or SLAVE connector to another board, then external list stops are passed on from the master to the slave: a Stop signal issued by the master board’s MASTER output (4) is received at the slave board’s SLAVE input (1), triggers a list stop at the slave board and is again issued by the MASTER output (4) of the slave board. List stops triggered by [stop_execution](#) are not passed on from master to slave, but external list stops triggered by [simulate_ext_stop](#) are passed on (see also [chapter 6.6.3 “Master/Slave Operation”, page 93](#)).

The command [get_startstop_info](#) queries the current stop status (i.e. whether one of the inputs is currently set to LOW) (2) and whether or not an external list stop has occurred.

As of Version DLL 528, OUT 530, after setting [set_control_mode](#) bit#1 = 1, the external start queue’s entries get explicitly cancelled upon an external list stop. For older versions and if [set_control_mode](#) bit #1 = 0, the queue only gets disabled.

External List Start

By a signal at the inputs /START, /START2 or /Slave-START, or by the commands **simulate_ext_start** or **simulate_ext_start_ctrl**, an external list start can be initiated (see (1), (5) and (7) in figure 61). This starts execution at the beginning of "List 1". But the commands **set_extstartpos** or **set_extstartpos_list** also allow pre-selection of another absolute start address. A list is only started if neither the BUSY status (as during list execution) nor the INTERNAL-BUSY status (as e.g. with **goto_xy**) nor the PAUSED status (after **pause_list**, **stop_list** or **set_wait**) is set at the moment.

Before the /START, /START2 or /Slave-START inputs (1) can be used, they must be enabled with the command **set_control_mode** (3).

The /START input is accessible by the 15-pin D-SUB LASER connector (see page 47), the /START2 input at the MARKING ON THE FLY socket connector (see page 53). Both signal inputs are internally connected to +3.3 V by pull-up resistors (4.7 kΩ). Both inputs are TTL active-LOW and edge sensitive (HIGH to LOW level transition). A list start is triggered – after activation by **set_control_mode** – as soon as one of the three input signals changes from HIGH to LOW (i.e. to 0 V or ground).

If the RTC5 Board is coupled by the MASTER or SLAVE connector to another board, then external list starts are passed on from the master to the slave: a start signal issued by the master board's MASTER output (8) is received at the slave board's SLAVE input (1), triggers a list start at the slave board and is again issued by the MASTER output (8) of the slave board.

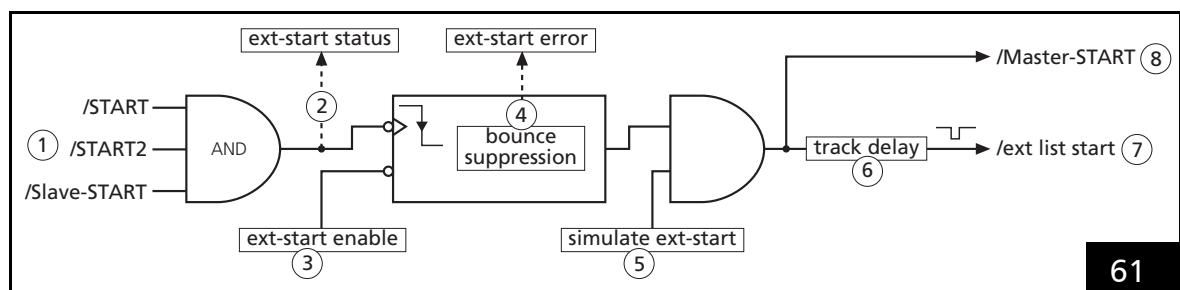
List starts triggered by **execute_list** or **execute_at_pointer** are not passed on from Master to slave, but external list starts triggered by **simulate_ext_start** or **simulate_ext_start_ctrl** are passed on (see also chapter 6.6.3 "Master/Slave Operation", page 93).

The command **get_startstop_info** queries the current start status (i.e. whether one of the inputs is currently set to LOW) (2) and whether a list has successfully started.

The command **bounce_supp** enables debouncing of start signals received at the /START, /START2 or /Slave-START inputs (4). Start signals occurring within the defined debouncing time after a successful start signal are thereby suppressed. The **get_marking_info** command queries whether a start signal was suppressed (4).

simulate_ext_start_ctrl can be used to start by control command a synchronous list start of master/slave-synchronized boards. However, as of version DLL 543, OUT 543, **simulate_ext_start_ctrl** may be disabled (because bit #4 of **set_control_mode** or **set_control_mode_list** is set).

The list command **simulate_ext_start** can be used, for instance, to trigger further list starts at defined intervals after the successful one-time external start of a list (see below).



External list start (see text for description)

External List Start with Track Delay

For many applications (e.g. if a workpiece must be initially transported from the light barrier to the scan system), the list start must be delayed with reference to the triggering start signal.

For this purpose, the commands `set_ext_start_delay`, `set_ext_start_delay_list` or `simulate_ext_start` allow configuring a track delay (see (6) in figure 61) that postpones execution of a list start relative to the triggering input signal or corresponding command. The track delay is specified in counting units of an internal encoder (encoder-counter) that itself can be triggered by an external or simulated encoder signal (see [page 246](#)).

External list starts triggered by an external start signal or by `simulate_ext_start` or `simulate_ext_start_ctrl` that do not execute immediately because of the track delay setting are held in a queue that can accommodate up to 8 starts (each start trigger is automatically generated when the delay has expired). This can be useful, for instance, when processing multiple workpieces transported to the scan system (even) at irregular intervals: here, up to 8 workpieces can simultaneously reside within the track delay (distance between the light barrier and scan system). If more external starts are triggered than can be simultaneously held in the 8-start wait loop, then an error bit is set, which can be queried by `get_startstop_info` (bit#11). If a track delay is set, then any previous queue is canceled (see `set_ext_start_delay` and `simulate_ext_start`).

Notes

- Note that the /START, /START2 and /Slave-START inputs are *edge* sensitive (HIGH to LOW level *transition*), whereas the /STOP, /STOP2 and /Slave-STOP inputs are *level* sensitive.
- An explicit call of the command `stop_execution` disables the /START, /START2 and /Slave-START inputs. An external stop signal also (at least temporarily) disables these inputs, i.e. as long as one of the inputs /STOP, /STOP2 or /Slave-STOP is LOW. The command `set_control_mode` can be used to define whether or not the /START, /START2 and /Slave-START inputs also stay disabled when the external stop signal is no longer active.
- The command `set_control_mode` additionally allows activation or deactivation of the inputs /START, /START2 or /Slave-START and deactivation of track delay.
- External starts are also suppressed after `pause_list`, `stop_list` or `set_wait` (PAUSED status is set). `restart_list`, `stop_execution`, `release_wait` or an external list stop ends suppression of the list start.
- If list inputs are not yet finished, a buffer flush should be initiated before an external start, e.g. by `set_input_pointer(get_input_pointer ())`, so that possibly buffered list commands are fully transferred to list memory (see [page 75](#)).
- If a master board is started internally (e.g. by `execute_list_pos`) and subsequently a slave board by `simulate_ext_start`, then the master and slave boards do not run synchronously if a home jump was previously activated by `home_position` or `home_position_xyz`: the home return executes on the master board *before* `simulate_ext_start` starts the slave board, but executes on the slave board *afterward*. While the home return executes on the slave board, the master board continues running. This asynchronicity does not occur if all boards are started by an external start signal (or by `simulate_ext_start` or `simulate_ext_start_ctrl`) or if no home jump is activated.

Regular (Periodic) External List Starts

Notes

- This functionality requires a firmware file in version 507 or higher (see [get_RTC_version](#)) and a DSP program file in version 511 or higher (see [get_hex_version](#)).

By [set_control_mode](#) and [set_control_mode_list](#) (bit #10), equidistant external list starts can be created that are independent of the time point of the start trigger as long as they occur within the specified track delay. This strongly periodic list processing is – independently of a list's actual duration of execution and the exact time point of the external list start – exactly synchronized to the RTC5's 10 µs clock.

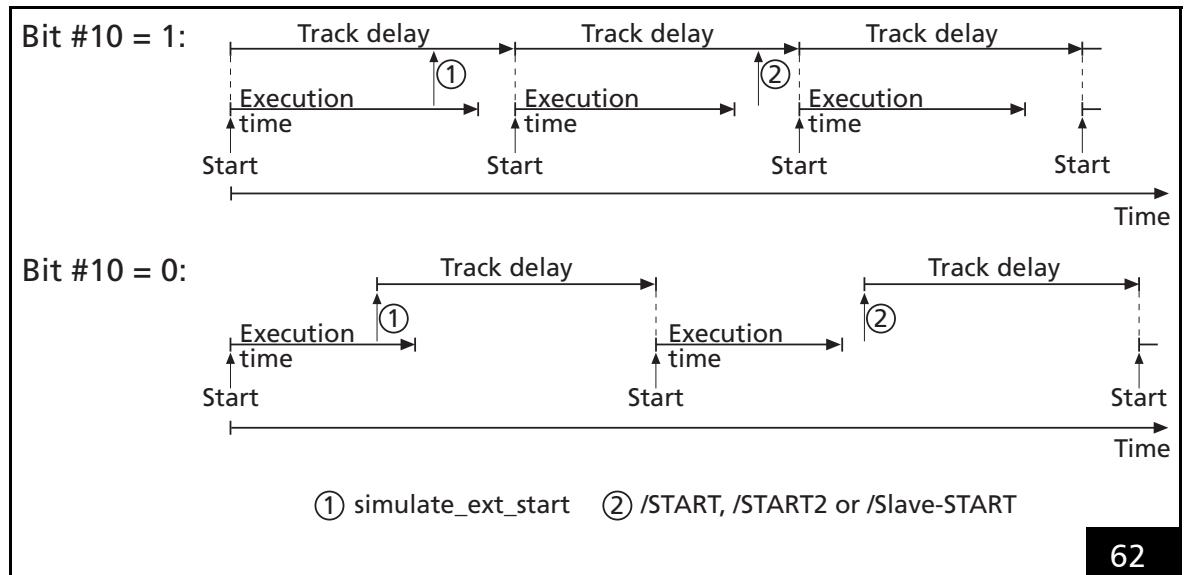
If desired, set bit #10 = 1 (Mode|Bit#10) to configure the internal encoder-counter's processing so that the track delay of an external list start is *not* counted only beginning with the time point of the triggering external start signal or [simulate_ext_start](#) command (bit #10 = 0) but already beginning with the most recently executed external list start (also executed by an external start signal or [simulate_ext_start](#) command) – see [figure 62](#).

For activation of this mode, an external list start must have successfully occurred (only one-time) in mode bit #10=0 (Mode &~Bit#10). Each subsequent external list start must be requested within the specified track delay.

Example (in PASCAL) of a typical command sequence (without use of an external start signal):

```
set_control_mode (Mode &~Bit#10) ;
// (one-time) reset (disable) bit #10 (initialization)
set_start_list_pos(ListNo, Pos) ;
// open some list
// afterward: some commands
simulate_ext_start(Delay, EncoderNo) ;
// first time start in mode bit #10 = 0, otherwise in
// mode bit #10 = 1
set_control_mode_list (Mode|Bit#10) ;
// set bit #10 = 1
// afterward: further commands
set_end_of_list;
// close the list
execute_list_pos(ListNo, Pos) ;
// (one-time) start the list
```

If the first start is to be triggered externally (e.g. by /START or by [simulate_ext_start_ctrl](#)) rather than by an [execute_list_pos](#) command, but all subsequent starts triggered by [simulate_ext_start](#), then [set_control_mode_list](#) in the above example must be called before [simulate_ext_start](#).



Regular and irregular external list starts (see text for description)



As of Version DLL 528, OUT 530, after setting **set_control_mode** bit#1 = 1, the external start queue's entries get explicitly cancelled upon an external list stop (thereby, external starts can be permanently stopped by an external list stop).

For older versions and for **set_control_mode**, bit #1 = 0 (default setting) – after an otherwise infinitely repetitive series has been stopped (e.g. by **set_control_mode**, bit #0 = 0) – you should deactivate the track delay and cancel the queue of not-yet-executed external list starts by **set_ext_start_delay** (Delay = 0). Otherwise, the next “equidistant” external start does not have the correct gap. **set_control_mode**(bit #2 = 1) alone is not sufficient for termination, because the track delay is reactivated by any not-yet-executed **simulate_ext_start** command.

If a further external start is missed within the track delay, then you should delete the wait loop (otherwise the encoder counter needs to run through a full 31-bit sequence before a start can again be successfully triggered). Deletion can be accomplished by resetting the track delay with **set_ext_start_delay**. In any case, bit #10 needs to first be reset (disabled) for initialization and then a (one-time) external start must be triggered (for external start signals bit #0 must be set) before bit #10 can again be set (see example). Otherwise, the first track delay in the wait loop is undefined.

9.3.2 Conditional Command Execution

The RTC5 provides so-called "conditional commands" that also allow the execution of individual list commands to be made dependent on external control signals.

These commands read the current value of the 16-bit digital input port at the EXTENSION 1 socket connector (see [page 51](#)) and act in accordance with the obtained value:

- Conditional Jumps:

The commands **list_jump_pos_cond** (synonymous with **list_jump_cond**) and **list_jump_rel_cond** result in – depending on the queried value – either a jump within a buffer area or no jump. The thereby specified jump addresses must fulfill the same conditions as with the **list_jump_pos** and **list_jump_rel** commands.

- Variable-distance jump:

The **switch_iport** command produces a relative list jump whose jump distance depends on the queried value.

- Conditional Calls of Non-Indexed Subroutines:

The commands **list_call_cond** and **list_call_abs_cond** either call or do not call – depending on the queried value – a non-indexed subroutine at a specified memory address.

- Conditional Calls of Indexed Subroutines:

The commands **sub_call_cond** and **sub_call_abs_cond** either call or do not call – depending on the queried value – an indexed subroutine with a specified index.

- Conditional output of peripheral signals:

The **set_io_cond_list** and **clear_io_cond_list** commands associate the output value of the 16-bit digital output port at the EXTENSION 1 socket connector (see [page 51](#)) directly with the signals of the digital input port: Depending on the current value of the digital input, individual bits of the output port are set or cleared.

- Conditional execution of any desired list commands:

The commands **if_cond** and **if_not_cond** have no effect if the condition for the queried value is fulfilled (or not fulfilled). Otherwise, they result in skipping the next list command. This means any list command can be made conditionally executable.

Example: the command sequence

```
if_cond(...)  
list_call(...)
```

is functionally identical to the command

```
list_call_cond(...)
```

The execution of any desired list command can also be made dependent on the current value at the 2-bit digital input port of the LASER connector. Therefore, the commands **if_pin_cond** and **if_not_pin_cond** are provided (these commands are functionally similar to the commands **if_cond** and **if_not_cond**). Reliable functioning of these conditional commands requires that the signals at the 2-bit digital input port remain unchanged for at least 10 µs.

As of version DLL 543, OUT 543, a condition for the 16-bit digital input port of the EXTENSION 1 socket connector can be defined by the control command **set_pause_list_cond**. If this condition is met, a currently executed list is paused by an automatically set **pause_list**. The list can only be resumed by **restart_list**. The condition is checked once per 10 µs clock cycle. A simultaneously present /STOP signal takes precedence over the conditional **pause_list**.



Programming Examples

The following programming examples are written in PASCAL.

(1) Confirm a signal:

```

set_start_list(1);
...
set_io_cond_list(0, 0, 1);                                // set bit #0 of the 16-bit digital output port
list_jump_rel_cond(0, 1, 0);                                // loop until the signal is confirmed
// (i.e. bit #0 of the digital input turns HIGH)
clear_io_cond_list(0, 0, 1);                                // clear bit #0 of the 16-bit output
list_jump_rel_cond(1, 0, 0);                                // loop until the signal is confirmed
...
set_end_of_list;
execute_list(1);

```

(2) If the lower four bits of the digital input have the value (0110), set bit #1 of the 16-bit digital output, otherwise clear it:

```

set_start_list(2);
...
//RTC4 style: list_jump_cond($0006, $0009, get_input_pointer + 3);
//this command uses absolute addresses and is not relocatable
//the following RTC5 command uses relative addresses and is relocatable
list_jump_rel_cond($0006, $0009, 3);                      // skip the next two commands if the state
// of the 16-bit input is (xxxx xxxx xxxx 0110)
clear_io_cond_list(0, 0, 2);                                // clear bit #1 of the 16-bit output and ..
//RTC4 style: set_list_jump(get_input_pointer + 2);
//this command uses absolute addresses and is not relocatable
//the following RTC5 command uses relative addresses and is relocatable
list_jump_rel(2);                                         // .. skip the next command
set_io_cond_list(0, 0, 2);                                // set bit #1 of the 16-bit output
...
set_end_of_list;
execute_list(2);
...
bit1 := (get_io_status AND $0002)                         // returns the current state of bit #1

```

(3) Choose between 15 small subroutines at defined memory addresses:

```

...
for i := 1 to 15 do
  list_call_cond(i, 15-i, i*100);                          // call subroutine at address i*100
// if [bit #3...bit #0] (binary) = i
...

```

(4) Choose between 15 indexed subroutines:

```

...
for i := 1 to 15 do
  sub_call_cond(i, 15-i, i);                               // call subroutine with index i
// if [bit #3...bit #0] (binary) = i
...

```

9.3.3 Synchronization by Encoder Signals

Use

When processing moving workpieces, the laser scan processes need to be adapted to the current workpiece position.

To incorporate the current workpiece position, the RTC5 can evaluate signals of up to two user-supplied incremental encoders. Though incremental encoders generally do not (directly) register the current workpiece position, they register the motion of the transport system (conveyor belt, rotating plate, etc.)⁽¹⁾: For each transport motion, they provide signals (depending on the direction of movement) to the RTC5 which can result in incrementing or decrementing of its two internal encoder counters. The states of the RTC5's encoder counters thereby correspond directly to the position of the workpiece⁽²⁾.

If workpieces are always processed at a constant speed and an encoder is therefore not needed, then the encoder signals can also be simulated by **simulate_encoder**, so that both encoder counters also increment at a constant counting rate without an external encoder signal. Of course, the encoder simulation can also be used to achieve a constant counting rate of the encoder counter for laser scan processes that are not dependent on workpiece motion.

The current counts of both encoder counters can be queried by the control command **get_encoder**. Alternatively, they can be stored in a buffer on the RTC5 by the list command **store_encoder** and then retrieved from there by the control command **read_encoder**. In addition, the RTC5 automatically evaluates the current counts if execution of the laser scan processes is controlled as follows:

- For Processing-on-the-fly-applications (see [page 199](#)), the coordinate values of all vector and arc commands are transformed in accordance with the encoder counters' current counts (i.e. in accordance with the current workpiece position).
- By the list command **wait_for_encoder_mode**, further execution of a list can be postponed until the selected encoder counter (i.e. the workpiece position) has overstepped or understepped a pre-defined value.
- For external list starts, a track delay can be defined by **simulate_ext_start**, **set_ext_start_delay** or **set_ext_start_delay_list** for postponing execution of a list start relative to the triggering input signal or corresponding command (see [page 240](#)).
- Encoder-speed-dependent automatic laser control (see [page 168](#)) uses the current encoder speed (counter pulses in the most recent 10 µs interval) of an encoder counter to control a laser signal parameter.

(1) The actual workpiece position can also be forwarded by the McBSP/SPI interface to the RTC5 (see [page 248](#)).

(2) The encoder counters are 32-bit counters (for signed 32-bit values). Upon reaching the maximum (minimum) counter value, counting continues with the minimum (maximum) value. A counter reset only occurs if triggered by Processing-on-the-fly-commands (see **set_fly_x**, **set_fly_y**, **set_fly_rot**, **set_control_mode** (bit #9 = 1) can be used to precisely synchronize the encoder reset with external start signals, thus avoiding the 10 µs jitter (random time offset between the start signal and the list start)).

Inputs for External Encoder Signals

For receiving encoder signals, the MARKING ON THE FLY socket connector provides two encoder inputs (ENCODER X and ENCODER Y, see [page 53](#)).

For linear movements of the parts to be processed, up to two user-supplied incremental encoders (that determine, independently from each other, the work-piece's motion in the X and Y directions) can be connected to these inputs.

For rotational movements only one incremental encoder is necessary. For Processing-on-the-fly applications, it must be connected to the ENCODER X input.

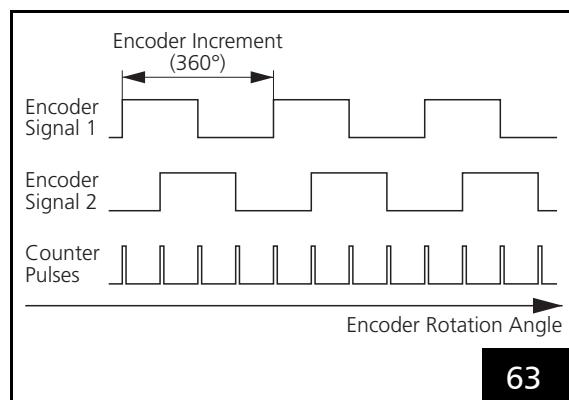
Each encoder input is designed for a pair of standardized differential input signals (RS422; HIGH level ≥ 2.0 V, LOW level ≤ 0.8 V).

The signals of encoder input ENCODER X trigger encoder counter Encoder0, the signals of encoder input ENCODER Y trigger encoder counter Encoder1.

The input signals of the user-supplied encoder must not exceed a maximum allowed frequency of 4 MHz (i.e. 16 encoder signal edges per μ s).

Encoder Simulation

The encoder simulation can be activated (and deactivated) by `simulate_encoder`. The RTC5 provides a periodic 1 MHz clock signal. After the encoder simulation is activated, the selected encoder counter (or both counters) is incremented at this clock rate. Signals of an attached incremental encoder are then ignored.



Timing diagram of a typical encoder signal pair and of the corresponding RTC counter pulses

Figure 63 shows the timing diagram of a typical encoder signal pair. The second encoder signal is usually phase-shifted by 90° relative to the first signal. The internal RTC encoder counter triggers at each edge of both signals, i.e. one encoder increment results in four counter pulses (counts). The relative 90° phase-shift of the two signals allows the RTC Board to detect not only the speed but the direction of movement as well. Depending on the direction of movement, the counter value is increased or decreased.

9.3.4 Synchronization and Online Positioning by McBSP/SPI Signals

For processing moving workpieces with a (stationary) scan system or a (stationary) workpiece with a moving scan system (e.g. by a robot arm), the laser scan processes need to be adapted to the workpiece's current position relative to the scan system (Processing-on-the-fly). Before processing randomly or imprecisely positioned and oriented workpieces, the scan system needs to be precisely aligned to the workpiece (online positioning).

The current position of the workpiece or scan system can be forwarded by the McBSP/SPI interface to the RTC5. The input value most recently fully transmitted at the McBSP/SPI interface can be queried by `read_mcbsp`. In addition, the RTC5 automatically evaluates the current input value if execution of the laser scan processes is controlled as follows:

- For Processing-on-the-fly-applications (see [page 199](#)), the coordinate values of all vector and arc commands are transformed in accordance with the current input value (i.e. in accordance with the current position of the workpiece or scan system).
- By the list command `wait_for_mcbsp`, further execution of a list can be postponed until the input value (i.e. the position of the workpiece or scan system) has reached, overstepped or understepped a pre-defined value.
- By online positioning (see [page 187](#)), the scan system is aligned in accordance with the current input values.

Notes

- To compensate linear motion, cartesian coordinates can be forwarded by the McBSP/SPI interface. Compensation of rotational motion, on the other hand, requires transmission of angle positions (i.e. values proportional to the current angle position – similar to synchronization by encoder signals, see [page 246](#)). It does not matter if the transmitted signal resets or simply counts further upon overflow (angle > 360°).
- The signals at the McBSP/SPI interface have no impact on the track delay, which can be defined for external list starts (by `simulate_ext_start`, `set_ext_start_delay` or `set_ext_start_delay_list`).



9.4 Periodical I/O Signals

With the commands **periodic_toggle** and **periodic_toggle_list** periodically repeated signals can be outputted at a selectable IO port (ANALOG OUT1 output port, see [page 48](#), ANALOG OUT2 output port, see [page 48](#), 8-bit digital output port, see [page 52](#), 16-bit digital output port, see [page 51](#), 2-bit digital output port, see [page 48](#)). Thereby, for example, external peripheral equipment can be triggered synchronously to list execution.

Notes

- At a **set_end_of_list**, **stop_execution** or external /STOP the periodical signals continue, even if they have been activated by the list command **periodic_toggle_list**.
- As of version DLL 543, OUT 543, **periodic_toggle** and **periodic_toggle_list** toggle endless with Count = $2^{32}-1$.

10 Commands and Functions

10.1 Overview

The following pages describe the complete RTC5 command set (control commands and list commands). The commands are listed according to their intended use. The page numbers refer to chapter 10.2 "RTC5 Command Set", page 261, where the commands (alphabetically ordered) are explained in detail.

10.1.1 Nomenclature

Multi-Board Commands

All commands marked (n_) in the following list also exist in a version for multiple RTC5 Boards installed in one computer. See chapter 6.6, page 92 for detailed information about these *multi-board* commands.

Nearly all single-board commands are also available in multi-board form. Exceptions are explicitly noted in the description list (in chapter 10.2).

Normal, Short and Variable List Commands and List Multi-Commands

The list commands of the RTC5 command set vary somewhat in their length of command execution. To differentiate, list commands in the list description (in chapter 10.2) are designated as "normal", "short", "variable" and "multi".

- Normal list commands require a full 10 µs clock period for command execution.
- Short list commands require less time for command execution. Therefore, they can be carried out along with the next list command, one directly after the other within a single 10 µs clock period, during which control commands cannot execute. In contrast, a short list command that immediately follows a normal list command executes in the subsequent 10 µs clock period.

The quicker execution of short list commands reduces total list processing time. In addition, during a polyline, the laser power can, for instance, be varied or the IO ports can be addressed (see **write_da_x_list**) between the polyline's individual mark vectors (see **set_laser_pulses**), all without interrupting the polyline (the laser remains on).

In contrast, if a specific time behavior is desired (10 µs clock period), you can insert an additional **list_nop** or **list_continue** command after any short list command to ensure that the next command only executes in the following 10 µs clock period. Insertion of **list_nop** (but not insertion of **list_continue**) results in the interruption of the polyline (the "laser active" laser control signals are switched off).

Currently, up to 12 short list commands per 10 µs clock period are possible. However, the maximum number can be lower, depending on the workload of the RTC5 Board and the DSP version⁽¹⁾. Short list commands that alter the output pointer (e.g. **sub_call**, **list_return** or **list_jump_pos**) count as two commands. If the maximum number is exceeded, a 10 µs clock period is inserted (equivalent to an additionally inserted **list_continue**, during the polyline the laser remains on)⁽¹⁾.

A maximum of two short list commands per 10 µs clock period are allowed before a normal list command. If a normal list command succeeds more than two short list commands, then the short list commands execute immediately and the normal list command execute delayed by a 10 µs clock period.

(1) On older RTC5 Boards where DSP version < 2 (**get_rtc_version** bits #16-23), only up to 8 short list commands per 10 µs clock period are possible. As of RTC5OUT.out version 517 or lower, a **list_nop** is automatically inserted for exceedances of the maximum number (the polyline is interrupted).

The maximum number of currently up to 12 short list commands may change in the future. For fully future-safe applications, only one short list command should precede a normal list command. If necessary, you should explicitly insert a **list_continue** or **list_nop** (**list_nop** interrupts the polyline).

- The command execution lengths of variable list commands are dependent on additional parameters and the user program. Details are provided in the corresponding command description.
- About list multi-commands, see [page 252](#).

Notes

- Where applicable, the total execution time of normal and variable list commands equals the sum of the command execution time and the execution time of the process initiated by the command. A subsequent list command only runs after this total execution time has completed. Example: the total execution time of the normal list command **mark_abs** is 10 µs (command execution time) + the execution time of the marking process. The latter is dependent on the settings of such parameters as marking length, marking speed, and delays etc.

Undelayed and Delayed Short List Commands (as of Version OUT 515)

Most short list commands (e.g. **list_jump_pos**, **list_call**, **sub_call** or **list_return**) (with all software versions) execute before the next list command and prior to a possible scanner delay (jump, mark or polygon delay). These commands are called “undelayed short list commands” in the corresponding command descriptions (in [chapter 10.2](#)).

However (as of version OUT 515), some short list commands only execute after the respective scanner delay (i.e. directly before the next command). Such commands are called “delayed short list commands” in the corresponding command descriptions. These include commands that immediately affect output or laser power (e.g. **set_rot_center_list**, **set_wobble_mode**, **write_da_x_list**, **set_laser_pulses**, **set_standby_list**, **set_mark_speed**, **set_encoder_speed**) or commands that affect data acquisition or time measurement (e.g. **set_trigger** or **save_and_restart_timer**).

If such (now delayed) commands were to execute with older software versions, then laser power, for example, would change at the end of a marking command (during a still-active mark or polygon delay) instead of at the beginning of the next marking command. **set_trigger** and **save_and_restart_timer** would erroneously take into account the delay of a preceding command instead of the delay of the subsequent command.

For several short list commands in a row, even a “delayed short list command” only executes delayed if no further “delayed short list commands” directly follow (i.e. if it is the last one in that sequence of commands or if an “undelayed short list command” directly follows)⁽¹⁾.

(1) This applies as of RTC5OUT.out Version 518. For versions 515-517 or lower, a “delayed short list command” among several short list commands in a row only execute delayed if it is at the end of that sequence of commands.



For several “delayed short list commands” in a row, all short list commands at the beginning of such a sequence execute undelayed (i.e. even before a scanner delay). So if several short list commands are placed between two marking commands within a polyline, then the most important “delayed” commands (e.g. the commands that change laser power) should therefore be placed at the end (i.e. directly before the next marking command) or directly before an “undelayed short list command”. Alternatively, you can also explicitly initiate processing of the scanner delay (e.g. by `list_nop`), so that all subsequent short list commands in fact always execute “delayed”.

If a normal list command succeeds more than two short list commands, then all short list commands (even “delayed” short list commands) execute immediately (“undelayed”) and the normal list command executes delayed by a 10 µs clock period.

List Multi-Commands

A list multi-command has multiple components that accordingly occupy multiple list buffer positions. The initial components are always undelayed short list commands. The final component is always a short, normal or variable list command. All components immediately execute successively. Any still-pending delayed short list commands execute first. The RTC5 command set currently only contains two-component list multi-commands (e.g. `wait_for_encoder_in_range` or `set_pixel_line_3d`).

10.1.2 Compatibility

For RTC5 customers who previously used the RTC4, the command descriptions (in [chapter 10.2](#)) note in each command’s “RTC4→RTC5” section:

- to what extent a command differs from that of the RTC4,
- whether the command is new to the command set,
- whether and how parameter values are converted in RTC4 compatibility mode.

Some RTC3/RTC4 commands and a few RTC2 commands emulated by the RTC3/RTC4 are not supported by the RTC5. These commands are listed in [chapter 10.3 “Unsupported RTC2/RTC3/RTC4 Commands”](#), page 642.

For general information about migrating from the RTC4 to the RTC5 and about the RTC4 compatibility mode, see [chapter 2.7](#), page 31.



10.1.3 Version Information

Starting with DLL 515, OUT 514, RBF 512, descriptions for a number of commands now include version information, listed under "Version info":

- For newly added commands: the software or DSP version in which they become available
- For some older commands: information on implemented changes.

All new commands and changes to old commands are also listed in the file
RTC5_Software_RevisionHistory_<Date>.pdf
included in the RTC5 software package.

10.1.4 Optional Functions

The RTC5 can be configured for functionality not available in the standard version (see [page 27](#)). If this extended functionality is not enabled or installed by SCANLAB, then the associated commands have only partial or non-existent effect. For example, commands to activate Processing-on-the-fly correction have no effect if the Processing-on-the-fly option has not been enabled. If the 3D functionality option has not been enabled, then 3D vector commands still execute, but no Z-axis signals are outputted. Likewise, if the "second scan head control" option has not been enabled, then commands that can explicitly specify the scan head connector (e.g. `set_matrix`) have no effect on the secondary scan head connector.

If applicable, the individual command descriptions describe such limitations.

10.1.5 Control Commands

DLL Initialization

free_RTC5_dll	299
get_RTC_mode	318
init_RTC5_dll	348
set_RTC4_mode	541
set_RTC5_mode	541

Using Multiple RTC5 Boards in One Computer

acquire_RTC	261
release_RTC	448
rtc5_count_cards	453
select_RTC	460

List Memory Commands

(n_) config_list	281
(n_) get_config_list	300
(n_) get_list_space	313
(n_) load_disk	379
(n_) save_disk	455

Board Initialization and Field Correction

(n_) get_head_para ⁽¹⁾	306
(n_) get_sync_status ⁽¹⁾	326
(n_) get_table_para ⁽¹⁾	327
(n_) load_correction_file ⁽¹⁾	375
(n_) load_program_file ⁽¹⁾	389
(n_) load_stretch_table ⁽²⁾	392
(n_) load_z_table ⁽²⁾	396
read_abc_from_file	438
(n_) select_cor_table ⁽¹⁾	457
(n_) set_dsp_mode ⁽¹⁾	476
(n_) sync_slaves ⁽¹⁾	603
write_abc_to_file	635

Laser Mode and Parameters⁽¹⁾

(n_) config_laser_signals	280
(n_) get_standby	320
(n_) load_auto_laser_control	373
(n_) load_position_control	387
(n_) set_auto_laser_control	464
(n_) set_auto_laser_params	467
(n_) set_encoder_speed_ctrl	479
(n_) set_firstpulse_killer	483
(n_) set_laser_control	504
(n_) set_laser_mode	508
(n_) set_laser_pulses_ctrl	511
(n_) set_pulse_picking	538
(n_) set_pulse_picking_length	538
(n_) set_qswitch_delay	539
(n_) set_softstart_level	553
(n_) set_softstart_mode	555
(n_) set_standby	557

Setting the Scanner Parameters⁽¹⁾

(n_) load_varpolydelay	395
(n_) set_delay_mode	475
(n_) set_jump_speed_ctrl	502
(n_) set_mark_speed_ctrl	514
(n_) set_sky_writing	546
(n_) set_sky_writing_limit	547
(n_) set_sky_writing_mode	548
(n_) set_sky_writing_para	550

Coordinate Transformations⁽¹⁾

(n_) set_angle	462
(n_) set_defocus ⁽²⁾	473
(n_) set_matrix	515
(n_) set_offset	530
(n_) set_offset_xyz ⁽²⁾	531
(n_) set_scale	542

Online Positioning⁽¹⁾

(n_) apply_mcbsp	264
(n_) set_mcbsp_matrix	520
(n_) set_mcbsp_rot	523
(n_) set_mcbsp_x	524
(n_) set_mcbsp_y	525

Status Monitoring and Diagnostics⁽¹⁾

(n_) get_head_status	307
(n_) get_overrun	318
(n_) get_value	332
(n_) get_values	334
(n_) get_waveform	336
(n_) measurement_status	417
(n_) stop_trigger	597

(1) See also the corresponding list commands.

(2) Only with enabled 3D option.



iDRIVE Commands

(n_) control_command	283
(n_) get_transform	329
(n_) read_user_data	447
(n_) send_user_data	461
transform	627
(n_) upload_transform	630

Pixel Output Mode – Scanning Raster Images (Bitmaps)

(n_) set_default_pixel	472
------------------------------	-----

I/O Commands⁽¹⁾

(n_) get_free_variable	304
(n_) get_io_status	310
(n_) get_mcbsp	317
(n_) mcbsp_init	415
(n_) mcbsp_init_spi	416
(n_) periodic_toggle	434
(n_) read_analog_in ⁽²⁾	439
(n_) read_io_port	440
(n_) read_io_port_buffer	441
(n_) read_mcbsp	443
(n_) read_multi_mcbsp	444
(n_) rs232_config	450
(n_) rs232_read_data	451
(n_) rs232_write_data	452
(n_) rs232_write_text	452
(n_) set_free_variable	495
(n_) set_laser_off_default	508
(n_) set_mcbsp_freq	517
(n_) set_mcbsp_out_ptr	522
(n_) set_port_default	537
(n_) write_8bit_port	634
(n_) write_da_1	636
(n_) write_da_2	636
(n_) write_da_x	637
(n_) write_io_port	640
(n_) write_io_port_mask	641

Starting and Stopping Lists by External Control Signals and Master/Slave Synchroni- zation⁽¹⁾

(n_) get_counts	301
(n_) get_master_slave	316
(n_) get_startstop_info	321
(n_) set_control_mode	470
(n_) set_extstartpos	481
(n_) set_max_counts	517
(n_) simulate_ext_start_ctrl	584
(n_) sync_slaves	603

List Handling and Status⁽¹⁾

(n_) auto_change	273
(n_) auto_change_pos	274
(n_) get_lap_time	311
(n_) get_status	323
(n_) get_wait_status	335
(n_) pause_list	434
(n_) quit_loop	436
(n_) read_status	445
(n_) release_wait	449
(n_) restart_list	450
(n_) start_loop	585
(n_) stop_execution	596
(n_) stop_list	596

Input Pointer Commands

(n_) get_input_pointer	310
(n_) get_list_pointer	312
(n_) load_list	385
(n_) set_input_pointer	497
(n_) set_start_list	558
(n_) set_start_list_1	558
(n_) set_start_list_2	558
(n_) set_start_list_pos	559

Output Pointer Commands

(n_) execute_at_pointer	295
(n_) execute_list	295
(n_) execute_list_1	295
(n_) execute_list_2	295
(n_) execute_list_pos	296
(n_) get_out_pointer	318

(1) See also the corresponding list commands.

(2) Only with RTC5 PCI Express Board or with RTC5 PCI Board and installed ADC add-on board



Subroutine Commands⁽¹⁾

(n_) <code>copy_dst_src</code>	293
(n_) <code>get_char_pointer</code>	300
(n_) <code>get_sub_pointer</code>	325
(n_) <code>get_text_table_pointer</code>	328
(n_) <code>load_char</code>	374
(n_) <code>load_sub</code>	393
(n_) <code>load_text_table</code>	394
(n_) <code>set_char_pointer</code>	468
(n_) <code>set_char_table</code>	469
(n_) <code>set_sub_pointer</code>	560
(n_) <code>set_text_table_pointer</code>	561

Direct Laser and Scan Head Control⁽¹⁾

(n_) <code>disable_laser</code>	294
(n_) <code>enable_laser</code>	294
(n_) <code>get_laser_pin_in</code>	311
(n_) <code>get_z_distance</code> ⁽²⁾	337
(n_) <code>goto_xy</code>	338
(n_) <code>goto_xyz</code> ⁽²⁾	339
(n_) <code>laser_signal_off</code>	356
(n_) <code>laser_signal_on</code>	357
(n_) <code>set_laser_pin_out</code>	509

Version Commands

(n_) <code>get_dll_version</code>	301
(n_) <code>get_hex_version</code>	308
(n_) <code>get_RTC_version</code>	319
(n_) <code>get_serial_number</code>	320

Error Commands

(n_) <code>get_error</code>	302
(n_) <code>get_last_error</code>	312
(n_) <code>reset_error</code>	449
(n_) <code>set_verify</code>	571
<code>verify_checksum</code>	631

Date, Time, Serial Numbers⁽¹⁾

(n_) <code>get_list_serial</code>	313
(n_) <code>get_serial</code>	319
(n_) <code>select_serial_set</code>	461
(n_) <code>set_serial</code>	544
(n_) <code>set_serial_step</code>	545
(n_) <code>time_update</code>	606

Processing-on-the-fly Commands⁽¹⁾

(n_) <code>get_encoder</code>	301
(n_) <code>get_fly_2d_offset</code>	304
(n_) <code>get_marking_info</code>	314
(n_) <code>init_fly_2d</code>	347
(n_) <code>load_fly_2d_table</code>	381
(n_) <code>read_encoder</code>	440
(n_) <code>set_ext_start_delay</code>	482
(n_) <code>set_fly_tracking_error</code>	489
(n_) <code>set_mcbsp_in</code> ⁽³⁾	518
(n_) <code>set_multi_mcbsp_in</code> ⁽³⁾	526
(n_) <code>set_rot_center</code>	540
(n_) <code>simulate_encoder</code>	582
(n_) <code>simulate_ext_stop</code>	584

Stepper Motor Control⁽¹⁾

(n_) <code>get stepper_status</code>	324
(n_) <code>stepper_abs</code>	586
(n_) <code>stepper_abs_no</code>	587
(n_) <code>stepper_control</code>	588
(n_) <code>stepper_enable</code>	590
(n_) <code>stepper_disable_switch</code>	589
(n_) <code>stepper_init</code>	591
(n_) <code>stepper_rel</code>	593
(n_) <code>stepper_rel_no</code>	594

Jump Mode⁽¹⁾

(n_) <code>get_jump_table</code>	311
(n_) <code>load_jump_table</code>	382
(n_) <code>load_jump_table_offset</code>	383
(n_) <code>set_jump_mode</code>	498
(n_) <code>set_jump_table</code>	503

Other Control Commands

(n_) <code>auto_cal</code>	270
(n_) <code>bounce_supp</code>	275
(n_) <code>get_auto_cal</code>	299
(n_) <code>get_galvo_controls</code>	305
(n_) <code>get_hi_data</code>	309
(n_) <code>get_hi_pos</code>	309
(n_) <code>get_time</code>	328
(n_) <code>home_position</code>	340
(n_) <code>home_position_xyz</code>	341
(n_) <code>move_to</code>	422
(n_) <code>set_hi</code>	496
(n_) <code>set_pause_list_cond</code>	533
(n_) <code>write_hi_pos</code>	639

(1) See also the corresponding list commands.

(2) Only with enabled 3D option.

(3) Limited functionality if no Processing-on-the-fly option

10.1.6 List Commands

Board Initialization and Field Correction

(n_) `select_cor_table_list` var 459

Jump Commands

(n_) <code>jump_abs</code> nor	350
(n_) <code>jump_rel</code> nor	352
(n_) <code>para_jump_abs</code> nor	422
(n_) <code>para_jump_rel</code> nor	424
(n_) <code>timed_jump_abs</code> nor	609
(n_) <code>timed_jump_rel</code> nor	611
(n_) <code>timed_para_jump_abs</code> nor	617
(n_) <code>timed_para_jump_rel</code> nor	620

3D Jump Commands

(n_) <code>jump_abs_3d</code> nor (1)	351
(n_) <code>jump_rel_3d</code> nor (1)	353
(n_) <code>para_jump_abs_3d</code> nor (1)	423
(n_) <code>para_jump_rel_3d</code> nor (1)	425
(n_) <code>timed_jump_abs_3d</code> nor (1)	610
(n_) <code>timed_jump_rel_3d</code> nor (1)	612
(n_) <code>timed_para_jump_abs_3d</code> mul (1)	619
(n_) <code>timed_para_jump_rel_3d</code> mul (1)	621

Microvector Commands

(n_) <code>micro_vector_abs</code> nor	418
(n_) <code>micro_vector_abs_3d</code> nor	419
(n_) <code>micro_vector_rel</code> nor	420
(n_) <code>micro_vector_rel_3d</code> nor	421

(1) Only with enabled 3D option.

nor normal list command

var variable list command

us undelayed short list command

ds delayed short list command

mul list multi-command

Mark Commands

(n_) <code>arc_abs</code> nor	266
(n_) <code>arc_rel</code> nor	268
(n_) <code>mark_abs</code> nor	398
(n_) <code>mark_ellipse_abs</code> nor	405
(n_) <code>mark_ellipse_rel</code> nor	406
(n_) <code>mark_rel</code> nor	407
(n_) <code>para_mark_abs</code> nor	427
(n_) <code>para_mark_rel</code> nor	429
(n_) <code>set_ellipse</code> us	477
(n_) <code>timed_arc_abs</code> nor	607
(n_) <code>timed_arc_rel</code> nor	608
(n_) <code>timed_mark_abs</code> nor	613
(n_) <code>timed_mark_rel</code> nor	615
(n_) <code>timed_para_mark_abs</code> nor	622
(n_) <code>timed_para_mark_rel</code> nor	625

3D Mark Commands

(n_) <code>arc_abs_3d</code> nor (1)	267
(n_) <code>arc_rel_3d</code> nor (1)	269
(n_) <code>mark_abs_3d</code> nor (1)	399
(n_) <code>mark_rel_3d</code> nor (1)	408
(n_) <code>para_mark_abs_3d</code> nor (1)	428
(n_) <code>para_mark_rel_3d</code> nor (1)	430
(n_) <code>timed_mark_abs_3d</code> nor (1)	614
(n_) <code>timed_mark_rel_3d</code> nor (1)	616
(n_) <code>timed_para_mark_abs_3d</code> mul (1)	624
(n_) <code>timed_para_mark_rel_3d</code> mul (1)	626

Text Commands

(n_) <code>mark_char</code> us	400
(n_) <code>mark_char_abs</code> us	401
(n_) <code>mark_text</code> var	412
(n_) <code>mark_text_abs</code> var	413
(n_) <code>select_char_set</code> us	456

Date, Time, Serial Numbers

(n_) <code>mark_date</code> nor	402
(n_) <code>mark_date_abs</code> nor	404
(n_) <code>mark_serial</code> nor	409
(n_) <code>mark_serial_abs</code> nor	411
(n_) <code>mark_time</code> nor	413
(n_) <code>mark_time_abs</code> nor	415
(n_) <code>select_serial_set_list</code> uk	461
(n_) <code>set_serial_step_list</code> nor	546
(n_) <code>time_fix</code> nor	604
(n_) <code>time_fix_f</code> nor	604
(n_) <code>time_fix_f_off</code> nor	605

Status Monitoring and Diagnostics

(n_) <code>set_trigger</code> ds	562
(n_) <code>set_trigger4</code> ds	568

Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization

(n_) <code>set_control_mode_list</code>	<code>nor</code>	472
(n_) <code>set_extstartpos_list</code>	<code>us</code>	481

List Handling and Structured Programming

(n_) <code>list_continue</code>	<code>nor</code>	363
(n_) <code>list_jump_pos</code>	<code>us</code>	365
(n_) <code>list_jump_rel</code>	<code>us</code>	367
(n_) <code>list_next</code>	<code>us</code>	369
(n_) <code>list_nop</code>	<code>nor</code>	369
(n_) <code>list_repeat</code>	<code>us</code>	370
(n_) <code>list_until</code>	<code>us</code>	372
(n_) <code>long_delay</code>	<code>nor</code>	397
(n_) <code>set_end_of_list</code>	<code>nor</code>	480
(n_) <code>set_list_jump</code>	<code>us</code>	513
(n_) <code>set_wait</code>	<code>nor</code>	572

Subroutine Commands

(n_) <code>list_call</code>	<code>us</code>	358
(n_) <code>list_call_abs</code>	<code>us</code>	359
(n_) <code>list_call_abs_repeat</code>	<code>us</code>	360
(n_) <code>list_call_repeat</code>	<code>us</code>	362
(n_) <code>list_return</code>	<code>us</code>	371
(n_) <code>sub_call</code>	<code>us</code>	598
(n_) <code>sub_call_abs</code>	<code>us</code>	599
(n_) <code>sub_call_abs_repeat</code>	<code>us</code>	600
(n_) <code>sub_call_repeat</code>	<code>us</code>	601

Setting the Laser Parameters

(n_) <code>config_laser_signals_list</code>	<code>ds</code>	280
(n_) <code>set_auto_laser_params_list</code>	<code>ds</code>	467
(n_) <code>set_encoder_speed</code>	<code>ds</code>	478
(n_) <code>set_firstpulse_killer_list</code>	<code>us</code>	483
(n_) <code>set_laser_delays</code>	<code>us</code>	507
(n_) <code>set_laser_pin_out_list</code>	<code>us</code>	509
(n_) <code>set_laser_pulses</code>	<code>ds</code>	510
(n_) <code>set_laser_timing</code>	<code>ds</code>	512
(n_) <code>set_pulse_picking_list</code>	<code>us</code>	539
(n_) <code>set_qswitch_delay_list</code>	<code>us</code>	539
(n_) <code>set_softstart_level_list</code>	<code>nor</code>	554
(n_) <code>set_softstart_mode_list</code>	<code>var</code>	556
(n_) <code>set_standby_list</code>	<code>ds</code>	558
(n_) <code>set_vector_control</code>	<code>us</code>	569

Setting the Scanner Parameters

(n_) <code>set_delay_mode_list</code>	<code>mul</code>	476
(n_) <code>set_jump_speed</code>	<code>us</code>	502
(n_) <code>set_mark_speed</code>	<code>ds</code>	514
(n_) <code>set_scanner_delays</code>	<code>ds</code>	544
(n_) <code>set_sky_writing_limit_list</code>	<code>us</code>	547
(n_) <code>set_sky_writing_list</code>	<code>nor</code>	547
(n_) <code>set_sky_writing_mode_list</code>	<code>nor</code>	549
(n_) <code>set_sky_writing_para_list</code>	<code>nor</code>	552

Coordinate Transformations

(n_) <code>set_angle_list</code>	<code>var</code>	463
(n_) <code>set_defocus_list</code>	<code>var (1)</code>	474
(n_) <code>set_matrix_list</code>	<code>var</code>	516
(n_) <code>set_offset_list</code>	<code>var</code>	530
(n_) <code>set_offset_xyz_list</code>	<code>var (1)</code>	532
(n_) <code>set_scale_list</code>	<code>var</code>	543

Online Positioning

(n_) <code>apply_mcbsp_list</code>	<code>nor</code>	265
(n_) <code>set_mcbsp_matrix_list</code>	<code>us</code>	521
(n_) <code>set_mcbsp_rot_list</code>	<code>us</code>	523
(n_) <code>set_mcbsp_x_list</code>	<code>us</code>	524
(n_) <code>set_mcbsp_y_list</code>	<code>us</code>	525

Direct Laser and Scan Head Control

(n_) <code>laser_on_list</code>	<code>var</code>	354
(n_) <code>laser_on_pulses_list</code>	<code>var</code>	355
(n_) <code>laser_signal_off_list</code>	<code>nor</code>	356
(n_) <code>laser_signal_on_list</code>	<code>nor</code>	357
(n_) <code>para_laser_on_pulses_list</code>	<code>var</code>	426

Pixel Output Mode – Scanning Raster Images (Bitmaps)

(n_) <code>set_default_pixel_list</code>	<code>us</code>	472
(n_) <code>set_n_pixel</code>	<code>var</code>	529
(n_) <code>set_pixel</code>	<code>var</code>	534
(n_) <code>set_pixel_line</code>	<code>nor</code>	535
(n_) <code>set_pixel_line_3d</code>	<code>mul (1)</code>	536

(1) Only with enabled 3D option.

`nor` normal list command

`var` variable list command

`us` undelayed short list command

`ds` delayed short list command

`mul` list multi-command

I/O Commands	
(n_) clear_io_cond_list us	279
(n_) periodic_toggle_list us	435
(n_) read_io_port_list us	442
(n_) rs232_write_text_list var	453
(n_) set_free_variable_list us	495
(n_) set_io_cond_list us	497
(n_) set_mcbsp_out us	521
(n_) write_8bit_port_list ds	635
(n_) write_da_1_list ds	636
(n_) write_da_2_list ds	637
(n_) write_da_x_list ds	638
(n_) write_io_port_list ds	640
(n_) write_io_port_mask_list ds	641
Conditional Commands	
(n_) if_cond us	342
(n_) if_not_cond us	344
(n_) if_not_pin_cond us	346
(n_) if_pin_cond us	347
(n_) list_call_abs_cond us	360
(n_) list_call_cond us	361
(n_) list_jump_cond us	364
(n_) list_jump_pos_cond us	366
(n_) list_jump_rel_cond us	368
(n_) sub_call_abs_cond us	599
(n_) sub_call_cond us	600
(n_) switch_iport us	602
Processing-on-the-fly Commands	
(n_) activate_fly_2d var (1)	262
(n_) activate_fly_xy var (1)	263
(n_) clear_fly_overflow us	278
(n_) fly_return nor	297
(n_) fly_return_z nor	298
(n_) if_fly_x_overflow us	342
(n_) if_fly_y_overflow us	343
(n_) if_fly_z_overflow us	343
(n_) if_notActivated us	344
(n_) if_not_fly_x_overflow us	345
(n_) if_not_fly_y_overflow us	345
(n_) if_not_fly_z_overflow us	346
(n_) park_position var (1)	431
(n_) park_return var (1)	432
(n_) set_ext_start_delay_list nor	483
(n_) set_fly_2d nor (1)	484
(n_) set_fly_limits us	485
(n_) set_fly_limits_z us	486
(n_) set_fly_rot nor (1)	487
(n_) set_fly_rot_pos nor (1)	488
(n_) set_fly_x nor (1)	490
(n_) set_fly_x_pos nor (1)	491
(n_) set_fly_y nor (1)	492
(n_) set_fly_y_pos nor (1)	493
(n_) set_fly_z nor (1)	494
(n_) set_mcbsp_in_list us (2)	519
(n_) set_multi_mcbsp_in_list nor (2)	528
(n_) set_rot_center_list ds	540
(n_) simulate_ext_start nor	583
(n_) store_encoder us	597
(n_) wait_for_encoder nor	631
(n_) wait_for_encoder_in_range mul	632
(n_) wait_for_encoder_mode nor	633
(n_) wait_for_mcbsp nor	634
Stepper Motor Control	
(n_) stepper_abs_list us	587
(n_) stepper_abs_no_list us	588
(n_) stepper_control_list us	589
(n_) stepper_enable_list us	590
(n_) stepper_rel_list us	593
(n_) stepper_rel_no_list us	594
(n_) stepper_wait nor	595
Jump Mode	
(n_) set_jump_mode_list nor	501
Other List Commands	
(n_) camming nor	276
(n_) range_checking us	436
(n_) save_and_restart_timer ds	454
(n_) set_wobble ds	573
(n_) set_wobble_control us	575
(n_) set_wobble_direction us	576
(n_) set_wobble_mode ds	577
(n_) set_wobble_offset ds	579
(n_) set_wobble_vector us	580

(2) Limited functionality if no Processing-on-the-fly option
 nor normal list command
 var variable list command
 us undelayed short list command
 ds delayed short list command
 mul list multi-command

(1) Only with enabled Processing-on-the-fly option



10.1.7 Data Types

The following table defines the formats and ranges of the different data types used by the RTC5 commands:

Data Format	Range	Pascal	C, C++	C#
unsigned 32-bit value	[0; $(2^{32}-1)$]	longword	unsigned long	uint
signed 32-bit value	[- 2^{31} ; + $(2^{31}-1)$]	longint	long	int
64-bit IEEE floating point format		double	double	double
pointer to a null-terminated ANSI string, 1 byte per char	4 Byte (for Win32-based applications) 8 Byte (for Win64-based applications)	pchar	char*	string

Pointer to Locations in the PC Memory

Some commands (e.g. `get_transform`, `get_values`, `get_waveform`, `transform` or `upload_transform`) have pointers to locations in the PC's memory as parameters. In C# and Pascal, appropriate pointer data types are heretofore used (see import declarations). In C and C++, the data type `ULONG_PTR` is used for this pointer parameters. The `ULONG_PTR` data type is defined in the C and C++ import declarations as follows (`ULONG_PTR` = unsigned 32-bit value for Win32-based applications, `ULONG_PTR` = unsigned 64-bit value for Win64-based applications):

```
#if !defined(ULONG_PTR)
    #if defined(WIN32)
        #define ULONG_PTR UINT
    #else
        #define ULONG_PTR UINT64
    #endif
#endif
```

Usually, the data type `ULONG_PTR` is also appropriately defined in the Windows header file `<BaseTsd.h>`.



10.2 RTC5 Command Set

The commands are in alphabetical order.

Ctrl Command	acquire_rtc
Function	Reserves the desired RTC5 Board for a user program.
Call	NoOfAcquiredCard = acquire_rtc(CardNo)
Parameter	CardNo (DLL-internal) number of the desired board as an unsigned 32-bit value.
Result	The return value (unsigned 32-bit value) is CardNo, if the reservation was successful, or 0 if the board is currently reserved for another user program or if version check detects an error.
Comments	<ul style="list-style-type: none"> • The acquire_rtc command is also useful for single-board systems which need to coordinate the use of a board by multiple user programs. • The acquire_rtc command has no effect if CardNo exceeds the number of RTC5 Boards found during initialization (see rtc5_count_cards) or if CardNo = 0 (real boards begin with 1) (return value 0, get_last_error return code: RTC5_PARAM_ERROR). • Access rights to existing boards are assigned exclusively (always to only one user program at a time). The acquire_rtc command therefore has no effect if the requested board is already reserved by another user program (return value 0, get_last_error return code: RTC5_ACCESS_DENIED). Before an already-reserved board can be acquired by acquire_rtc, the user program with current access rights must explicitly release its rights by release_rtc or free_rtc5_dll. On the other hand, if the board was already freed prior to initialization of a user program, then initialization by init_rtc5_dll result in board management assigning board access rights for the user program. In this case, an explicit acquire_rtc call is not needed and has no effect. Nevertheless, the return value is the CardNo. • Assorted versions of the RTC5 DLL and the program files RTC5OUT.out, RTC5RBF.rbf and RTC5DAT.dat cannot be arbitrarily combined with another. The command acquire_rtc performs a version compatibility check. If no program files are loaded, then a version check cannot be explicitly executed (get_last_error return code: RTC5_TIMEOUT), but the check still regarded as successful and acquisition is not hindered. If program files have been loaded and the version check determines an error, then access is denied (return value 0, get_last_error return code: RTC5_ACCESS_DENIED RTC5_VERSION_MISMATCH). • A board successfully requested by acquire_rtc does not automatically become the active board. Activation is only achieved by select_rtc or init_rtc5_dll. • Running boards are neither halted nor initialized by acquire_rtc. • The acquire_rtc command is also available without explicit access rights to a particular RTC5 Board. • The acquire_rtc command is not available as a multi-board command. • See also chapter 6.7.1 "Board Acquisition by a User Program", page 96.



Ctrl Command	acquire_rtc
RTC4→ RTC5	New command.
References	init_RTC5_dll , select_rtc , free_RTC5_dll , release_rtc , RTC5_count_cards

Variable List Command	activate_fly_2d
Function	Activates a set_fly_2d Processing-on-the-fly application without encoder resets.
Restriction	If the Processing-on-the-fly option is not enabled, then the command terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	activate_fly_2d(ScaleX, ScaleY)
Parameters	ScaleX, Scaling factors as for set_fly_2d . ScaleY
Comments	<ul style="list-style-type: none"> If no Processing-on-the-fly correction is active, then activate_fly_2d activates set_fly_2d Processing-on-the-fly correction (see chapter 8.7.4, page 206). Unlike set_fly_2d, activate_fly_2d does not thereby reset the encoders, but instead recalculates the last Processing-on-the-fly-uncorrected coordinate values such that the Processing-on-the-fly-corrected output matches the current output. If the then-current recalculated coordinate values would have fallen outside the 24-bit virtual image field, then Processing-on-the-fly correction is not activated and an error bit is set that is queryable by get_marking_info (bit #9 = 1). If Processing-on-the-fly correction is <i>active</i>, then activate_fly_2d is a short list command without further effect and merely sets an error bit queryable by get_marking_info (bit #9 = 1). Therefore, activate_fly_2d cannot be used to modify the Processing-on-the-fly mode itself or to modify the scaling factors of the same mode. You can also query the error bit by the short list command if_notActivated in order to possibly jump to an appropriate error handling routine. Successful activation by activate_fly_2d <i>does not</i> reset any already-set error bit. It remains set for get_marking_info until get_marking_info is called. If unallowed parameter values are supplied (e.g. for ScaleX = 0), then activate_fly_2d is (already during loading) replaced by a list_nop (get_last_error return code RTC5_PARAM_ERROR). activate_fly_2d does not affect the laser signals ("laser active" laser control signals remain on/off if they were on/off).
RTC4→ RTC5	New command. RTC4 compatibility mode: see set_fly_2d .
Version info	Available as of version DLL 536, OUT 536.
References	set_fly_2d



Variable List Command	activate_fly_xy
Function	Activates a set_fly_x/set_fly_y Processing-on-the-fly application without encoder resets.
Restriction	If the Processing-on-the-fly option is not enabled, then the command terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>activate_fly_xy(ScaleX, ScaleY)</code>
Parameters	ScaleX, Scaling factors as for set_fly_x/set_fly_y . ScaleY
Comments	<ul style="list-style-type: none"> If no Processing-on-the-fly correction is active, then activate_fly_xy activates set_fly_x/set_fly_y Processing-on-the-fly correction. Unlike set_fly_x, and set_fly_y, activate_fly_xy does not thereby reset the encoders, but instead recalculates the last Processing-on-the-fly-uncorrected coordinate values such that the Processing-on-the-fly-corrected output matches the current output. If the then-current recalculated coordinate values would have fallen outside the 24-bit virtual image field, then Processing-on-the-fly correction is not activated and an error bit is set that is queryable by get_marking_info (bit #9 = 1). If Processing-on-the-fly correction is <i>active</i>, then activate_fly_xy is a short list command without further effect and merely sets an error bit queryable by get_marking_info (bit #9 = 1). Therefore, activate_fly_xy cannot be used to modify the Processing-on-the-fly mode or the scaling factors of the same mode You can also query the error bit by the short list command if_not_activated in order to possibly jump to an appropriate error handling routine. Successful activation by activate_fly_xy does not reset any already-set error bit. It remains set for get_marking_info until get_marking_info is called. If unallowed parameter values are supplied (e.g. for ScaleX = 0), then activate_fly_xy is (already during loading) replaced by a list_nop (get_last_error return code RTC5_PARAM_ERROR). activate_fly_xy does not affect the laser signals ("laser active" laser control signals remain on/off if they were on/off).
RTC4→ RTC5	New command. RTC4 compatibility mode: see set_fly_x/set_fly_y .
Version info	Available as of version DLL 536, OUT 536.
References	set_fly_x, set_fly_y



Ctrl Command	apply_mcbsp
Function	Fetches the most recent values fully transmitted over the McBSP/SPI interface for online positioning and defines offset and/or rotation matrix M_R or general transformation matrix M_T for subsequent coordinate transformations.
Call	apply_mcbsp(HeadNo, at_once)
Parameters	<p>HeadNo Number of the scan head connector as an unsigned 32-bit value. = 1: The definition only affects the <i>primary</i> scan head connector. = 2: The definition only affects the <i>secondary</i> scan head connector (activation required). = 0, 3: The definition affects <i>both</i> scan head connectors. Only the two least significant bits are evaluated.</p> <p>at_once This parameter (unsigned 32-bit value) determines when the defined transformation becomes effective. = 0: The new total transformation (total matrix and offset) is only calculated and applied to the current position when the next list command is executed. = 1: The new total transformation is calculated immediately (or before the next list command if a list is currently BUSY or the board is INTERNAL-BUSY) and applied to the current position. = 2: The new total transformation (total matrix and offset) is only calculated and applied to the current position when the next jump_abs, jump_rel, goto_xy or goto_xyz command is executed. > 2: As at_once = 2.</p>
Comments	<ul style="list-style-type: none"> Data acquisition by the McBSP/SPI interface for online positioning must be activated in advance by the commands set_mcbsp_x, set_mcbsp_y and/or set_mcbsp_rot or set_mcbsp_matrix (or with the corresponding list commands). Depending on the configuration, the command apply_mcbsp only defines (as with set_offset) an X and/or Y offset or also (as with set_angle) a rotation matrix or (as with set_matrix) a general matrix operation (see chapter 8.3 "Online Positioning", page 187). As with the commands described in chapter 8.2 "Coordinate Transformations", page 183, the parameter at_once determines when the newly defined total transformation becomes effective. Transformations previously defined by set_angle, set_offset or set_matrix might get overwritten by apply_mcbsp. In contrast, transformations and focus shifts previously defined by set_scale or set_defocus and Z offsets defined by set_offset_xyz is continued to be taken into account, when the total transformation gets recalculated. Any new definitions made with set_angle, set_offset or set_matrix overwrite coordinate transformations defined by the McBSP/SPI interface. The McBSP/SPI interface always ignores the first FrameSync signal after a load_program_file or mcbsp_init, so available data is not transmitted (see page 56).
RTC4→RTC5	New command.
Version info	Available as of version DLL 524, OUT 526.
References	apply_mcbsp_list , set_mcbsp_x , set_mcbsp_y , set_mcbsp_rot , set_mcbsp_matrix



Normal List Command	apply_mcbsp_list				
Function	Same as apply_mcbsp , but a list command.				
Call	<code>apply_mcbsp_list(HeadNo, at_once)</code>				
Parameters	<table> <tr> <td>HeadNo</td> <td>See apply_mcbsp.</td> </tr> <tr> <td>at_once</td> <td> <p>This parameter (unsigned 32-bit value) determines when the defined transformation becomes effective.</p> <ul style="list-style-type: none"> = 0: The transformation settings are only collected and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (e.g. by a list command with <code>at_once = 1</code> or a corresponding control command). = 1: The transformation is immediately calculated (including all transformation settings that were collected until then) and processed prior to the next list command. = 2: The transformation settings are only collected and intermediately stored (as with <code>at_once = 0</code>). However, The transformation is immediately calculated (including all transformation settings that were collected and intermediately stored until then) and applied to the current position when the next jump_abs, jump_rel, goto_xy or goto_xyz command is executed. > 2: As <code>at_once = 2</code>. </td> </tr> </table>	HeadNo	See apply_mcbsp .	at_once	<p>This parameter (unsigned 32-bit value) determines when the defined transformation becomes effective.</p> <ul style="list-style-type: none"> = 0: The transformation settings are only collected and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (e.g. by a list command with <code>at_once = 1</code> or a corresponding control command). = 1: The transformation is immediately calculated (including all transformation settings that were collected until then) and processed prior to the next list command. = 2: The transformation settings are only collected and intermediately stored (as with <code>at_once = 0</code>). However, The transformation is immediately calculated (including all transformation settings that were collected and intermediately stored until then) and applied to the current position when the next jump_abs, jump_rel, goto_xy or goto_xyz command is executed. > 2: As <code>at_once = 2</code>.
HeadNo	See apply_mcbsp .				
at_once	<p>This parameter (unsigned 32-bit value) determines when the defined transformation becomes effective.</p> <ul style="list-style-type: none"> = 0: The transformation settings are only collected and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (e.g. by a list command with <code>at_once = 1</code> or a corresponding control command). = 1: The transformation is immediately calculated (including all transformation settings that were collected until then) and processed prior to the next list command. = 2: The transformation settings are only collected and intermediately stored (as with <code>at_once = 0</code>). However, The transformation is immediately calculated (including all transformation settings that were collected and intermediately stored until then) and applied to the current position when the next jump_abs, jump_rel, goto_xy or goto_xyz command is executed. > 2: As <code>at_once = 2</code>. 				
Comments	<ul style="list-style-type: none"> • See apply_mcbsp. 				
RTC4→ RTC5	New command.				
Version info	Available as of version DLL 524, OUT 526.				
References	apply_mcbsp				



Normal List Command	arc_abs
Function	Moves the laser focus from the current position at marking speed along an arc with the specified angle and center point (absolute coordinate values) within a two-dimensional image field.
Call	<code>arc_abs(X, Y, Angle)</code>
Parameters	X, Y Absolute coordinates of the arc center in <i>bits</i> as signed 32-bit value. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.
	Angle Arc angle in ° as a 64-bit IEEE floating point value. Positive angle values correspond to clockwise angles. Allowed range: [-3600.0° ... +3600.0°] (± 10 full circles). Out-of-range values are edge-clipped.
Comments	<ul style="list-style-type: none"> If the marking speed has not been previously explicitly set by set_mark_speed or set_mark_speed_ctrl, then the marking is executed at a predefined marking speed of 1000 <i>bits per ms</i>. The “laser active” laser control signals are automatically turned on at the beginning of the command (or remain on after a directly preceding mark or arc command). The defined scanner and laser delays are thereby taken into account (see chapter 7.2 “Delay Settings for Synchronizing Scan Head and Laser Control”, page 111). Note that other delays are executed in Sky writing mode (see page 127). Exception: zero-length arc commands (see notes on page 115). During the conversion of specified coordinate values into scan system output values, any previously defined coordinate transformations, assigned correction tables etc. are taken into account after successful microvectorization (see page 141). The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, the RTC5 multiplies the specified arc center coordinate values by 16 (the allowed range of values is correspondingly reduced to [-524288 ... 524287]).
References	set_mark_speed , set_scanner_delays , arc_rel , timed_arc_abs , mark_abs , arc_abs_3d , mark_ellipse_abs



Normal List Command	arc_abs_3d
Function	Moves the laser focus at marking speed from the current position spirally around an axis parallel to the Z axis. The x and y components thereby characterize an arc with the specified angle around the specified axis, while the z component characterizes a linear motion from the current position to the specified end point. The position of the spiral axis and the z end coordinate are specifiable as absolute coordinate values.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as arc_abs .
Call	<code>arc_abs_3d(X, Y, Z, Angle)</code>
Parameters	<p>X, Y Position of the spiral axis (parallel to the Z axis) as absolute coordinates in <i>bits</i> (signed 32-bit values). Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</p> <p>Z Absolute Z end coordinate in <i>bits</i> as signed 32-bit values. Allowed range: [-32768 ... 32767]. An out-of-range value is edge-clipped.</p> <p>Angle Arc angle in ° as a 64-bit IEEE floating point value. Positive angle values correspond to clockwise angles. Allowed range: [-3600.0° ... +3600.0°] (± 10 full circles). An out-of-range value is edge-clipped.</p>
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, this command functions similarly to the arc_abs command (see the comments there). The X and Y axes can be controlled with 20-bit resolution, the Z axis with 16-bit resolution. The RTC5 (in RTC5 mode as well as in RTC4 compatibility mode) automatically upscales Z coordinate values internally to 20-bit values, so that the three dimensions of space can be handled equivalently in terms of the supplied mark speed value. During calculation of the Z output value to the scan system, any previously defined offset to the Z coordinate and focal length is taken into account (see page 141). The z motion is not taken into account during calculation of the number of microsteps.
RTC4→RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the values specified for X and Y by 16 (the allowed range of values is correspondingly reduced). The value range for Z is identical for the RTC5 and RTC4.
Version info	Available as of version DLL 517, OUT 516, RBF 512.
References	arc_abs , arc_rel_3d



Normal List Command	arc_rel
Function	Moves the laser focus from the current position at marking speed along an arc with the specified angle and center point (relative coordinate values) within a two-dimensional image field.
Call	<code>arc_rel(dx, dY, Angle)</code>
Parameters	<code>dx, dY</code> <i>Relative coordinates of the arc center in bits as signed 32-bit value.</i> Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.
	<code>Angle</code> Arc angle in ° as a 64-bit IEEE floating point value. Positive angle values correspond to clockwise angles. Allowed range: [-3600.0° ... +3600.0°] (± 10 full circles). Out-of-range values are edge-clipped.
Comments	<ul style="list-style-type: none"> The coordinates for the arc center are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to arc_abs (see the comments there). The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, the RTC5 multiplies the specified arc center coordinate values by 16 (the allowed range of values is correspondingly reduced to [-524288 ... 524287]).
References	set_mark_speed , set_scanner_delays , arc_abs , timed_arc_rel , mark_rel , arc_rel_3d , mark_ellipse_rel



Normal List Command	arc_rel_3d
Function	Moves the laser focus at marking speed from the current position spirally around an axis parallel to the Z axis. The x and y components thereby characterize an arc with the specified angle around the specified axis, while the z component characterizes a linear motion from the current position to the specified end point. The position of the spiral axis and the z end coordinate are specifiable as relative coordinate values.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as arc_rel .
Call	<code>arc_rel_3d(dx, dy, dz, Angle)</code>
Parameters	<p>dx, dy Position of the spiral axis (parallel to the Z axis) as relative coordinates in <i>bits</i> (signed 32-bit values). Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</p> <p>dz Relative Z end coordinate in <i>bits</i> as signed 32-bit values. Allowed range: [-32768 ... 32767]. An out-of-range value is edge-clipped.</p> <p>Angle Arc angle in ° as a 64-bit IEEE floating point value. Positive angle values correspond to clockwise angles. Allowed range: [-3600.0° ... +3600.0°] (± 10 full circles). An out-of-range value is edge-clipped.</p>
Comments	<ul style="list-style-type: none"> The position of the spiral axis (dx, dy) and the Z end coordinate (dz) are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to arc_abs_3d (see the comments there).
RTC4→ RTC5	New command. RTC4 compatibility mode: see arc_abs_3d
Version info	Available as of version DLL 517, OUT 516, RBF 512.
References	arc_abs_3d , arc_rel



Ctrl Command	auto_cal
Function	Controls the functions for (automatic self-) calibration of the scan system attached to the specified scan head connector.
Call	<code>ErrorCode = auto_cal(HeadNo, Command)</code>
Parameters	<p>HeadNo Number of the scan head connector as an unsigned 32-bit value. Allowed values: = 1: Primary scan head connector. = 2: Secondary scan head connector (activation required).</p> <p>Command Control parameter (as an unsigned 32-bit value, allowed value range: [0 ... 4]): = 0: The RTC5 detects the current Home-In positions, stores them in the DLL and in the RTC5 EEPROM as Home-In reference values and initializes the gain and offset values (Gain = 1.0, Offset = 0). = 1: The RTC5 detects the current Home-In positions, calculates and sets the new gain and offset values and thereby activates drift compensation. = 2: The RTC5 deactivates drift compensation by initializing the gain and offset values (Gain = 1.0, Offset = 0). = 3: The RTC5 detects the current Home-In positions (but – in comparison to <code>Command = 1</code> – leaves the gain and offset values unchanged and i.e. does not activate drift compensation) = 4: The RTC5 checks the ASC hardware (i.e. checks whether a scan system attached to the specified scan head connector is equipped with an internal sensor system for automatic self-calibration – Home-In sensors) and returns the type and status of the detected sensor system. The detected type is also stored in the RTC5 EEPROM.</p>
Result	<p>Error code or type of sensor system as an unsigned 32-bit value:</p> <p>3 The command cannot be executed because the board is currently BUSY or INTERNAL-BUSY.</p> <p>6 Parameter error.</p> <p>The following error codes are only returned after <code>Command = 0 ... 3</code>:</p> <p>0 No error.</p> <p>1, 10, 11 Home-In sensor not found (this could also mean a Home-In sensor is defective) (1: for X axis (galvanometer scanner 2) 10: for Y axis (galvanometer scanner 1) 11: for both axes).</p> <p>2, 20, 22 The spread in measured values during a measurement cycle is too high (2: for X axis / 20: for Y axis / 22: for both axes).</p> <p>4, 40, 44 Reference data not found (only for <code>Command = 1</code> and <code>3</code>) (4: for X axis / 40: for Y axis / 44: for both axes).</p>



Ctrl Command	auto_cal
Result (cont'd)	<p>5, 50, 55 Calibration error (Error during calibration or error in reference data) (5: for X axis / 50: for Y axis / 55: for both axes); (for DLL version 519 or lower) this return value could also mean the scan system contains no Home-In sensor(s).</p> <p>7 Error: automatic laser control activated in Mode 2.</p> <p>9, 90 Sensor for axis X or Y is defective. Only returned after Command = 0, 1 or 3.</p> <p>The following error code is only returned after Command = 0 or 4, and even then only if no other errors occurred:</p> <p>8 EEPROM write error (for this error, the get_last_error return code RTC5_EEPROM_ERROR is always generated).</p> <p>The following values are only returned after Command = 4:</p> <p>100 For both axes: a sensor system of type1 is included and is functioning.</p> <p>200 For both axes: a sensor system of type2 is included and is functioning.</p> <p>19 X axis (galvanometer scanner 2): a sensor system of type1 is included and is functioning. Y axis (galvanometer scanner 1): sensor system is defective.</p> <p>29 X axis (galvanometer scanner 2): a sensor system of type2 is included and is functioning. Y axis (galvanometer scanner 1): sensor system is defective.</p> <p>91 X axis (galvanometer scanner 2): sensor system is defective. Y axis (galvanometer scanner 1): a sensor system of type1 is included and is functioning.</p> <p>92 X axis (galvanometer scanner 2): sensor system is defective. Y axis (galvanometer scanner 1): a sensor system of type2 is included and is functioning.</p> <p>99 For both axes: sensor system is defective.</p> <p>255 For both axes: there is no sensor system included in the scan system.</p>
Comments	<ul style="list-style-type: none"> For information on using the command, see chapter 8.11 "Automatic Self-Calibration", page 224. At the end of this command's execution, the RTC5 always moves the galvanometer scanners back to the position held prior to the call, possibly with corrections attributable to changed gain and offset values. During determination of the current Home-In positions with auto_cal(Command = 0, 1 or 3), the current gain and offset corrections of the galvanometer scanners are not taken into account; neither are any head corrections or the current positions of the scanners. After first-time or renewed connecting a scan system, a reference value determination should be performed by auto_cal(Command = 0). Otherwise, the RTC5 uses the stored reference values of a previously operated (possibly different) scan system when later performing an automatic self-calibration. After initialization of the RTC5, or after a reset, drift compensation is turned off (gain = 1.0, offset = 0). However, previously determined reference values are still available.



Ctrl Command	auto_cal
Comments (cont'd)	<ul style="list-style-type: none"> If no appropriate Home-In reference values were stored or the scan system is not equipped with Home-In sensors, then the measurement routine for <code>auto_cal</code>(Command = 1) (axis-specific) automatically aborts and restores the prior state. If the values for HeadNo or Command are invalid, then <code>auto_cal</code> is not executed (return value 6, <code>get_last_error</code> return code: RTC5_PARAM_ERROR). This also applies for HeadNo = 2 if the "second scan head control" option has not been enabled (return value 6). The <code>auto_cal</code> command is likewise not be executed (return value 3, <code>get_last_error</code> return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or has been halted by <code>pause_list</code>) or the board's INTERNAL-BUSY status is currently set. In contrast, the command is executed when a list has been paused by <code>set_wait</code> (PAUSED status set). For RTC5_PARAM_ERROR, the BUSY status is not checked; therefore return codes RTC5_BUSY and RTC5_PARAM_ERROR do not arrive simultaneously. For Command = 0, 1 or 2, a valid correction table must be loaded and assigned. Otherwise, unexpected jumps can occur when gain/offset values are updated. Gain and offset correction can also be directly set by <code>set_hi</code> (even for systems <i>without</i> Home-In sensors). Reference values determined and stored by <code>auto_cal</code>(Command = 0, 1 or 3) can be queried by <code>get_hi_pos</code>. As of version DLL 520: ASC hardware checks are performed not just by <code>auto_cal</code>(Command = 4), but also automatically for <ul style="list-style-type: none"> – <code>auto_cal</code>(Command = 0) and – the first call of <code>auto_cal</code>(Command = 1) and <code>auto_cal</code>(Command = 3) if neither <code>auto_cal</code>(Command = 0) nor <code>auto_cal</code>(Command = 4) were previously executed. In each case, the detected ASC hardware type gets stored in the RTC5's EEPROM and can be subsequently queried by <code>get_auto_cal</code>. For automatic Command = 4 execution, however, no corresponding return value is generated. The return value is instead simply that of the primary Command call (see above). <p>When an error occurs, a corresponding error code gets saved to the RTC5 EEPROM (therefore, <code>get_auto_cal</code> does not return 100 or 200). As soon as hardware functionality is restored, you can clear the error from the RTC5 EEPROM by explicitly calling <code>auto_cal(command = 0)</code> or <code>auto_cal(command = 4)</code>. The error <i>cannot</i> be cleared from the RTC5 EEPROM by calling <code>auto_cal(command = 1)</code>.</p>
RTC4→RTC5	The functions for automatic self-calibration (Command = 0 ...2) are (largely) unchanged (see version info). New: Command = 3 and Command = 4.
Version info	Change with version DLL 520, OUT 519 (see page 693). Last change with version DLL 535: Return value 8 (see above).
References	<code>set_hi</code> , <code>get_hi_pos</code> , <code>get_auto_cal</code> , <code>write_hi_pos</code>



Ctrl Command	auto_change
Function	Activates a one-time automatic list change.
Call	auto_change()
Comments	<ul style="list-style-type: none">The auto_change command is synonymous with <code>auto_change_pos(0)</code>. The subsequent list then starts at its beginning.See also comments for auto_change_pos.
RTC4→ RTC5	Unchanged functionality.
Version info	Last change with version OUT 515.
References	auto_change_pos , get_status , read_status



Ctrl Command	auto_change_pos
Function	Activates a one-time automatic list change and simultaneously defines the list position at which execution continues.
Call	auto_change_pos(Pos)
Parameter	Pos Start position (list memory address) as an offset referenced to the beginning of the list to be started by the automatic list change (as an unsigned 32-bit value).
Comments	<ul style="list-style-type: none"> • The auto_change_pos command triggers a subsequent <i>one-time-only</i> list change or a list new start. For further list changes or list new starts, the command must be called again. • The command can be issued at any desired time point. • If the automatic list change is activated during processing of a list, then upon reaching set_end_of_list execution continues without delay at the supplied start position of the other list. If there is only <i>one</i> list ($\text{Mem2} = 0$, see config_list), then upon reaching set_end_of_list execution continues at the supplied start position of this list. • During processing of a list, the other list (and also the current list) can be newly loaded (see chapter 6.4.6 "Automatic List Changing", page 79). • So that auto_change_pos can function at all, any already active list must absolutely be finalized by set_end_of_list; the new list should already be loaded and the input pointer should be sufficiently ahead of the output pointer (otherwise, "old" commands are executed). If, during list execution, the end of the list is reached without encountering a set_end_of_list, then execution automatically continues at the beginning of the current list. • If an automatic list change is activated when no list is currently being processed, then checking takes place as to whether a list was already processed and the other list was started (at the supplied start position). If no list was previously executed, then "List 1" is regarded as already executed (initialization) and "List 2" is started. • If a list memory address outside the corresponding list area is supplied (depending on which list should be started: $\text{Pos} \geq \text{Mem1}$ or $\text{Pos} \geq \text{Mem2}$), then the start position is set to the beginning of the list ($\text{Pos} = 0$). • If, during processing of a list, the auto_change_pos(Pos >0) and start_loop commands are called, then upon the next set_end_of_list the command auto_change_pos(Pos >0) is executed; and at the next one the start_loop command is executed. • The current list and list execution statuses can be queried by the commands read_status and get_status. • The auto_change_pos command triggers a flush of the list input buffer (see page 75).
RTC4→ RTC5	Essentially unchanged, however: The list memory address (Pos) is supplied to the RTC5 as a relative memory address referenced to the beginning of the respective list, whereas the RTC4 is supplied an absolute memory address (0...7999). If no memory area is assigned to "List 2" by config_list ($\text{Mem2} = 0$), then the RTC5 command behaves like the RTC4 command.
References	auto_change , get_status , read_status



Ctrl Command	bounce_supp
Function	Debounces the external start signal.
Call	bounce_supp(Length)
Parameter	Length Debouncing time in <i>ms</i> as an unsigned 32-bit value. Allowed range: [0 ... 1023]. The 22 higher bits are ignored.
Comments	<ul style="list-style-type: none"> The command bounce_supp enables debouncing of start signals received at the /START, /START2 or /Slave-START inputs (see section "External List Start", page 240). Start signals occurring within the defined debouncing time after a successful start signal are thereby suppressed. Recommended procedure: Start a list, which operates more than one second. If /START, /START2 or /Slave-START bounces, then an additional trigger error signal is generated, which can be detected by get_marking_info (bit#8 = 1). Increase the debouncing time until this additional signal has vanished (bit#8 = 0). The debouncing time default value is 0 ms.
RTC4→ RTC5	New command.
References	get_startstop_info

Normal List Command	camming
Function	Enables camming functionality.
Call	camming (FirstPos, NPos, EncoderNo, Ctrl, Scale, Code)
Parameters	<p>FirstPos Starting address of the camming command list (absolute address in list memory) as an unsigned 32-bit value. Allowed range: [0 ... (2^{20}-1)].</p> <p>NPos Length of the camming command list (without terminating set_end_of_list or list_return) as an unsigned 32-bit value. Allowed range: [1 ... (2^{20}-FirstPos)].</p> <p>EncoderNo Number of the encoder counter, whose pulses is evaluated for controlling the camming process, as an unsigned 32-bit value. Allowed values: = 0: Encoder counter Encoder0. = 1: Encoder counter Encoder1. Higher bits are ignored.</p> <p>Ctrl Camming control mode as an unsigned 32-bit value. Allowed values: = 0: RTC5 controls laser similarly to a mark command, camming terminates automatically. = 1: User controls laser, camming terminates automatically. = 2: User controls laser, camming does not terminate automatically, the output index is clipped to [0, NPos-1]. = 3: User controls laser, camming does not terminate automatically, the camming list is continuously processed (output index modulus NPos).</p> <p>Scale Translation (conversion factor) between the encoder-counter value and the command index as a 64-bit IEEE floating point value. Allowed range: $2^{-60} < scale < 2^{60}$.</p> <p>Code Unsigned 32-bit value. Is not used.</p>
Comments	<ul style="list-style-type: none"> • See also chapter 8.12 "Camming", page 229. • If there are unallowed parameter values, camming is replaced by a list_nop already during loading (get_last_error return code RTC5_PARAM_ERROR). • Each time camming is called, the current encoder-counter value is ascertained for use as the new reference value. At a later time point, the corresponding command index of the camming command list is calculated for the then-current encoder-counter value (using the reference value and the Scale parameter). For each call of camming, the first command in the camming command list (index = 0) is always the first command to be processed. • camming waits for a scanner delay but sets no delay itself. • If Ctrl = 0, then the laser is (as with a normal mark command) switched on at the beginning of the camming process and switched off after it terminates (laser delay settings are taken into account). If Ctrl > 0, then the state of the laser is not changed; its control is the full responsibility of the user.



Normal List Command	camming
	<ul style="list-style-type: none"> • If <code>Ctrl = 0 or 1</code>, then the camming command terminates automatically (in the next cycle) as soon as the index first undershoots 0 or overshoots <code>NPos-1</code> (the final camming command to be executed is then be the one with an index of 0 or <code>NPos-1</code>). For these two modes (as always, if a list is BUSY), no external list starts are allowed as long as the camming command has not yet terminated. • In contrast, control modes <code>Ctrl = 2 and 3</code> are endless. Here, the camming process can only be terminated by stop_execution or an external list stop. However, in these two modes (as an exception) external list starts are allowed if the list (and the camming command) is still active. If <code>Ctrl = 2</code>, then the index is clipped to the boundaries 0 or <code>NPos - 1</code>; for <code>Ctrl = 3</code> it is set to modulus <code>NPos</code> (whereby the encoder speed is supposed not to be so high that a complete rotation is skipped over). Thus <code>Ctrl = 3</code> works like a ring buffer. • The camming command is a normal list command, but with a variable execution period. • camming functions even if the Processing-on-the-fly option is not activated. • While camming is executing, get_out_pointer always provides the position of the camming command, not the position of the current index.
RTC4→ RTC5	New command.
References	set_encoder_speed, set_auto_laser_control



Undelayed Short List Command	clear_fly_overflow
Function	Resets error bits (#4...7 from get_marking_info) for customer-defined monitoring of Processing-on-the-fly applications.
Call	<code>clear_fly_overflow(Mode)</code>
Parameters	<p>Mode This parameter (an unsigned 32-bit value) specifies which error bits to reset:</p> <ul style="list-style-type: none"> • Bit#0 = 1: error bit#4 (underflow X) is reset. • Bit#1 = 1: error bit#5 (overflow X) is reset. • Bit#2 = 1: error bit#6 (underflow Y) is reset. • Bit#3 = 1: error bit#7 (overflow Y) is reset. • Bit#4 = 1: error bit#24 (underflow Z) is reset. • Bit#5 = 1: error bit#25 (overflow Z) is reset. <p>Bits #0..5 can be combined as desired. Higher-order bits are ignored.</p>
Comments	<ul style="list-style-type: none"> • For command usage, see section "Customer-Defined Monitoring Area and Conditional Command Execution (as of Version DLL 525, OUT 527)", page 214. • For Mode = 0, all six error bits are reset as with Mode = 15.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 525, OUT 527. Last change (with version DLL531, OUT532): bits#4, 5.
References	get_marking_info



Undelayed Short List Command	clear_io_cond_list
Function	Clears the bits of the 16-bit digital output port on the EXTENSION 1 socket connector that are set in the parameter <code>MaskClear</code> , if the current <code>IOvalue</code> at the 16-bit digital <i>input</i> port on the EXTENSION 1 socket connector meets the following condition: $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } ((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0}$ (i.e. if the bits specified in <code>Mask1</code> are 1 and the bits specified in <code>Mask0</code> are 0).
Call	<code>clear_io_cond_list(Mask1, Mask0, MaskClear)</code>
Parameters	<code>Mask1</code> , 16-bit masks as unsigned 32-bit values. <code>Mask0</code> , Only the least significant 16 bits are evaluated. <code>MaskClear</code>
Comments	<ul style="list-style-type: none"> The command clears only those bits of the digital output port that are set in the parameter <code>MaskClear</code> and leaves the other bits unchanged. See also section "16-Bit Digital Input and Output", page 51 and chapter 9.3.2 "Conditional Command Execution", page 244.
Examples (Pascal)	<ul style="list-style-type: none"> Clear bit #4 of the output port (DIGITAL OUT4), if bit #0 of the input port (DIGITAL IN0) is set and bits #1 to #3 (DIGITAL IN1...3) of the input port are not set: <code>clear_io_cond_list(\$0001, \$000E, \$0010)</code> Always clear bit #15 of the output port (and leave the other bits unchanged): <code>clear_io_cond_list(0, 0, \$8000)</code>
RTC4→ RTC5	Unchanged functionality.
References	set_io_cond_list , write_io_port , write_io_port_mask , get_io_status , read_io_port



Ctrl Command	config_laser_signals
Function	Configures the laser signal types to be outputted on pin 1 (LASER1), pin 2 (LASERON) and pin 9 (LASER2) of the LASER connector.
Call	config_laser_signals(Config)
Parameter	<p>Config Desired signal configuration as an unsigned 32-bit value. Thereby the following bits configure:</p> <ul style="list-style-type: none"> Bits # 0-1: Pin 2 (LASERON channel) Bits # 2-3: Pin 1 (LASER1 channel) Bits # 4-5: Pin 9 (LASER2 channel) Bits # 6-31 are ignored <p>To the following signal type:</p> <ul style="list-style-type: none"> = 0 = 00_b: LASERON signal = 1 = 01_b: LASER1 signal = 2 = 10_b: LASER2 signal = 3 = 11_b: FirstPulseKiller signal <p>The default setting (after load_program_file) is: Config = 100100_b = 0x24 = 36</p>
Comments	<ul style="list-style-type: none"> • The specified configuration takes effect the next time the laser is switched on. Therefore, the command should not be issued during an active marking procedure.
Examples	<ul style="list-style-type: none"> • Config = 000111_b: FirstPulseKiller signal on LASERON channel, LASER1 signal on LASER1 channel, LASERON signal on LASER2 channel. In this configuration, LASER1 signals synchronously generated with the LASERON signal can be immediately outputted, whereas the laser itself switches on only after a delay by the FirstPulseKiller signal. • Config = 100100_b (default setting): LASERON signal on the LASERON channel, LASER1 signal on the LASER1 channel, LASER2 signal on the LASER2 channel. In this configuration, LASER1 signals can only be switched on after the laser, not before it (here the LASERON signal is the laser start signal; LASER1 signals cannot be outputted before the laser start, because the RTC5 only generates LASER1 signals if the LaserON signal is on).
RTC4→ RTC5	New command.
Version info	Available as of version DLL 533, OUT 534, RBF 524.

Delayed short list command	config_laser_signals_list
Function	Same as config_laser_signals , but a list command.
Call	config_laser_signals_list(Config)
Parameters	Config Siehe config_laser_signals .
Comments	<ul style="list-style-type: none"> • See config_laser_signals.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 537, OUT 537.
References	config_laser_signals

Ctrl Command	config_list				
Function	Configures list memory, therefore allocates memory areas for lists.				
Call	<code>config_list(Mem1, Mem2)</code>				
Parameters	<table border="0"> <tr> <td>Mem1</td> <td>Desired memory size for "List 1" (as an unsigned 32-bit value).</td> </tr> <tr> <td>Mem2</td> <td>Desired memory size for "List 2" (as an unsigned 32-bit value).</td> </tr> </table>	Mem1	Desired memory size for "List 1" (as an unsigned 32-bit value).	Mem2	Desired memory size for "List 2" (as an unsigned 32-bit value).
Mem1	Desired memory size for "List 1" (as an unsigned 32-bit value).				
Mem2	Desired memory size for "List 2" (as an unsigned 32-bit value).				
Comments	<ul style="list-style-type: none"> • The RTC5's list memory contains 2^{20} storage positions that can be divided into three areas (lists) by config_list. The sizes of "List 1" and "List 2" are specified through parameters <code>Mem1</code> and <code>Mem2</code>. config_list automatically assigns to the protected area ("List 3") all remaining memory not assigned to "List 1" or "List 2". • The following rules apply to the parameters <code>Mem1</code> and <code>Mem2</code> (invalid values are automatically corrected in the specified order): <ul style="list-style-type: none"> – $\text{Mem1} > 0$ ("List 1" cannot be empty) <code>Mem1 = 0</code> is corrected to <code>Mem1 = 1</code>. – $\text{Mem1} \leq 2^{20}$ ("List 1" can contain a maximum of 2^{20} storage positions) <code>Mem1 > 2^{20}</code> is corrected to <code>Mem1 = 2^{20}</code> – $\text{Mem1} = -1$ is interpreted as $\text{Mem1} = (2^{32}-1)$ and corrected to <code>Mem1 = 2^{20}</code>. Example: With <code>config_list(-1, x)</code> where <code>x</code> is any value (also <code>x = -1</code>), "List 1" is automatically assigned the entire list memory (<code>Mem1 = 2^{20}</code>, <code>Mem2 = 0</code>, no memory for "List 3"). – $\text{Mem2} \leq 2^{20} - \text{Mem1}$ ("List 2" can maximally receive the "rest" of list memory) <code>Mem2 = 0</code> is allowed. <code>Mem2 > 2^{20} - \text{Mem1}</code> is corrected to <code>Mem2 = 2^{20} - \text{Mem1}</code>. – $\text{Mem2} = -1$ is interpreted as $\text{Mem2} = (2^{32}-1)$ and corrected to <code>Mem2 = 2^{20} - \text{Mem1}</code>. Example: With <code>config_list(\text{Mem1}, -1)</code>, "List 2" is automatically assigned with the "rest" of list memory (<code>Mem2 = 2^{20} - \text{Mem1}</code>, no memory for "List 3"). – Storage positions for "List 3": $2^{20} - \text{Mem1} - \text{Mem2}$ • By default, the RTC5's list buffer area is preconfigured so that "List 1" and "List 2" can each accept 4000 list commands (<code>Mem1 = Mem2 = 4000</code>). The protected "List 3" then owns the remaining 1040576 of the 2^{20} storage positions. • The config_list command is ignored (<code>get_error</code> return value = <code>RTC5_BUSY</code>) if the board's BUSY status or PAUSED status is currently set (list is being processed or has been halted by <code>pause_list</code> or <code>set_wait</code>). • Configuration by config_list does not alter the contents of list memory. Repeating the call with differing parameters is therefore nondestructive. However, after a configuration change, previously loaded list commands are processed in accordance with the new configuration. Moreover, a configuration change could in some circumstances affect the input pointer (if, prior to the reconfiguration, it pointed to a memory position which was assigned to "List 3" by the configuration change, then it is shifted to the beginning of "List 1") or affect a previously started automatic list change. This should be taken into account when further loading or executing command lists. <p>Also observe the notes in chapter 6.3.2 "Configuring the List Memory", page 74.</p>				



Ctrl Command	config_list
Comments (cont'd)	<ul style="list-style-type: none">If you do not know the current configuration data for list memory (Mem1 and Mem2), you can find out after load_list(ListNo, 0) or set_start_list_pos(ListNo, 0) by using the command get_list_space (if the board changed "ownership" previously call get_config_list).
RTC4→ RTC5	New command.
References	get_config_list

Ctrl Command	control_command	
Function	Sends a control command to an iDRI ^E scan system (intelliSCAN, intelliSCAN _{de} , intelliDRILL, intellicube, intelliWELD, varioSCAN _{de}).	
Call	control_command(Head, Axis, Data)	
Parameter	Head	Number of the scan head connector as an unsigned 32-bit value. Allowed values: = 1: Primary scan head connector. = 2: Secondary scan head connector (activation required).
	Axis	Number of the axis as an unsigned 32-bit value. Allowed values: = 1: X axis (STATUS channel, galvanometer scanner 2). = 2: Y axis (STATUS1 channel, galvanometer scanner 1).
	Data	Command code with optional parameter as unsigned 32-bit value. The upper part (bits#16-31) of this parameter is <i>not</i> evaluated, only the lower part (bits#0-15) is evaluated. The more significant data byte of the lower part Code_H (bits#8-15) represents a command code and the less significant data byte Code_L (bits#0-7) an optional parameter. For each command code the corresponding command and its allowed parameter values are described below. Example: With Data = 0501 _H , i.e. Code _H = 05 _H (command SetMode) and Code _L = 01 _H , the actual position is selected to be returned from the scan system.
Code_H	Command and Parameter Values (Code_L)	
05 _H	<p>SetMode: This command selects the data signal to be returned from the scan system for the specified axis by the respective status channel. See also chapter 8.1.2 "Configuring Status Return Behavior", page 173. Each Code_L parameter value corresponds to a particular data type.</p> <ul style="list-style-type: none"> • Default setting: Code_L = 00_H (XY2-100 status word) • The data types marked in grey are only relevant for the intelliWELD. <p>iDRI^E scan systems with SL2-100 interface return signed 20-bit values. Scan systems with XY2-100 Enhanced interface return signed 16-bit values; here the XY2-100 converter converts (by multiplying by 16) the returned values to signed 20-bit values.</p>	
Code_L	Data Signal Type returned from the Scan System	
00 _H	XY2-100 status word Bit #19 (MSB), 11 = 1: Internal voltages normal (Power OK). Bit #18, 10 = 1: Galvanometer scanner temperature within normal range (TempOK). Bit #16, 15, 8, 7 = 1: X- and Y-axis position error within normal range (PosAck-Signal). Bit #17, 12, 9, 4 = 1. Bit #14, 6 = 1 (For intelliWELD and for scan systems with sensors for automatic self calibration: Reserved). Bit #13, 5 = 0. Bit #0...3 = 0.	
01 _H	Actual (angular) position / bit [-2 ¹⁹ ... 2 ¹⁹ -1].	
02 _H	Set (angular) position / bit [-2 ¹⁹ ... 2 ¹⁹ -1].	
03 _H	Position error (= set position – actual position) / bit [-2 ¹⁹ ... 2 ¹⁹ -1].	
04 _H	Actual current (output stage current) / 16 ⁻¹ mA [-2 ¹⁹ ... 2 ¹⁹ -1].	
05 _H	Relative galvanometer scanner control [-16000 ... 16000] (not with intellicube) (the return value 16 corresponds to 1%).	
06 _H	Actual (angular) velocity / (bit/ms) [-2 ¹⁹ ... 2 ¹⁹ -1].	

Ctrl Command	control_command		
	(05 _H)	Code _L	Returned Data Signal Type
		12 _H	Actual Z axis position / bit [-2 ¹⁹ ... 2 ¹⁹ -1] (only for intelliWELD with varioSCAN FC; not with varioSCAN FC i). This data signal can only be read by the channel of the scan head connector to which the Z axis is connected.
		14 _H	Galvanometer scanner temperature / 160 ⁻¹ °C [-2 ¹⁹ ... 2 ¹⁹ -1].
		15 _H	Servo board temperature / 160 ⁻¹ °C [-2 ¹⁹ ... 2 ¹⁹ -1].
		16 _H	AGC voltage (PD supply voltage) / 1600 ⁻¹ V [-2 ¹⁹ ... 2 ¹⁹ -1].
		17 _H	DSP core supply voltage (1.8 V) / 1600 ⁻¹ V [-2 ¹⁹ ... 2 ¹⁹ -1].
		18 _H	DSP IO voltage (3.3 V) / 1600 ⁻¹ V [-2 ¹⁹ ... 2 ¹⁹ -1].
		19 _H	Analog section voltage (9 V) / 1600 ⁻¹ V [-2 ¹⁹ ... 2 ¹⁹ -1].
		1A _H	AD converter supply voltage (5 V) / 1600 ⁻¹ V [-2 ¹⁹ ... 2 ¹⁹ -1].
		1B _H	AGC current (PD supply current) / 16 ⁻¹ mA [-2 ¹⁹ ... 2 ¹⁹ -1].
		1D _H	Relative heating output of the corresponding galvanometer scanner heater (not relevant for intellicube, dynAXIS® XS and scan systems with water-cooled scanners). Bits #4...19 Relative heating output / % [0 ... 1000]. Bits #0...3 = 0.
		1E _H	Serial number (lower 16 bits) Bits #4...19 Serial number (lower 16 bits) [0 ... 2 ¹⁶ -1]. Bits #0...3 = 0.
		1F _H	Serial number (higher 16 bits) Bits #4...19 Serial number (higher 16 bits) [0 ... 2 ¹⁶ -1]. Bits #0...3 = 0.
		20 _H	Article number (lower 16 bits) Bits #4...19 Article number (lower 16 bits) [0 ... 2 ¹⁶ -1]. Bits #0...3 = 0.
		21 _H	Article number (higher 16 bits) Bits #4...19 Article number (higher 16 bits) [0 ... 2 ¹⁶ -1]. Bits #0...3 = 0.
		22 _H	Firmware version number Bits #4...19 Version number [0 ... 2 ¹⁶ -1]. Bits #0...3 = 0.
		23 _H	Calibration angle (mechanical scan angle (\pm) for 96% of the maximum and minimum control value, i.e. for ± 503316 bit) Bits #4...19 Calibration angle / 10 ⁻³ ° [0 ... 2 ¹⁶ -1]. Bits #0...3 = 0.
		24 _H	Aperture Bits #4...19 Aperture / mm [0 ... 2 ¹⁶ -1]. Bits #0...3 = 0.



Ctrl Command	control_command		
	(05 _H)	Code _L	Returned Data Signal Type
		25 _H	<p>Wavelength</p> <p>Bits #4...19 Wavelength / nm [0 ... 2¹⁶-1].</p> <p>Bits #0...3 = 0.</p>
		26 _H	<p>Tuning number (as of Firmware Version 2078)</p> <p>Bits #12...19 Start setting.</p> <p>Bits #4...11 Current setting.</p> <p>Bits #0...3 = 0.</p>
		27 _H	<p>only after control_command(Data = 17FF_H): Number of the internally (e.g. by Code = 17FF_H for reinstatement by Code = 1700_H) buffered, specified to be returned data type (as of Firmware Version 2078)</p> <p>Bits #12...19 = 0.</p> <p>Bits #4...11 Temporarily stored setting.</p> <p>Bits #0...3 = 0.</p>
		28 _H	<p>Current operational state (flags B15 ... B0):</p> <p>Bit #19 (MSB) (B15) = 1: Galvanometer scanner's output stage is on.</p> <p>Bit #18 (B14) = 1: Galvanometer scanner heater's output stage is on (always 0 for intellicube, dynAXIS® XS and scan systems with water-cooled scanners).</p> <p>Bit #17 (B13) = 1: Internal voltages normal.</p> <p>Bit #16 (B12) = 1: Position error within normal range.</p> <p>Bit #15 (B11) = 1: Galvanometer scanner and servo board temperature within normal range.</p> <p>Bit #14 (B10) = 1: Booting process completed.</p> <p>Bit #13 (B9) = 0: A critical error occurred. The system was switched into a permanent error state.</p> <p>Bit #12 (B8) = 0: The external power supply voltages have – at least temporarily – dropped below the allowed value.</p> <p>Bit #11 (B7) = 0: The temperature in the scan system exceeds the maximum allowed value. The system was switched into a temporary error state.</p> <p>Bit #10 (B6) = 1: The AD converter was successfully initialized.</p> <p>Bit #9 (B5) = 0: The galvanometer scanner has reached a critical edge position.</p> <p>Bit #8 (B4) = 1: All control parameters valid.</p> <p>Bit #7 (B3) Reserved (for intelliWELD: 1 = currently no INTERLOCK error).</p> <p>Bit #6 (B2) Reserved (for intelliWELD: 1 = The sensor board was recognized).</p> <p>Bit #5 (B1) Reserved.</p> <p>Bit #4 (B0) = 1: The control is activated, as soon as all necessary flags are set.</p> <p>Bit #0...3 = 0.</p>

Ctrl Command	control_command																				
	(05 _H)	Code _L	Returned Data Signal Type																		
		29 _H	<p>Current operational state (flags B31 ... B16):</p> <table> <tr><td>Bit #19(MSB) (B31)</td><td>= 1: AD converter supply voltage (5 V) OK.</td></tr> <tr><td>Bit #18 (B30)</td><td>= 1: Analog section voltage (9 V) OK.</td></tr> <tr><td>Bit #17 (B29)</td><td>= 1: DSP IO voltage (3.3 V) OK.</td></tr> <tr><td>Bit #16 (B28)</td><td>= 1: DSP core supply voltage (1.8 V) OK.</td></tr> <tr><td>Bit #15 (B27)</td><td>= 1: AGC voltage (PD supply voltage) OK.</td></tr> <tr><td>Bit #14 (B26)</td><td>= 1: Servo board operation temperature within normal range.</td></tr> <tr><td>Bit #13 (B25)</td><td>= 1: Galvanometer scanner operation temperature within normal range.</td></tr> <tr><td>Bits #4...12 (B16-24)</td><td>Reserved.</td></tr> <tr><td>Bits #0...3</td><td>= 0.</td></tr> </table>	Bit #19(MSB) (B31)	= 1: AD converter supply voltage (5 V) OK.	Bit #18 (B30)	= 1: Analog section voltage (9 V) OK.	Bit #17 (B29)	= 1: DSP IO voltage (3.3 V) OK.	Bit #16 (B28)	= 1: DSP core supply voltage (1.8 V) OK.	Bit #15 (B27)	= 1: AGC voltage (PD supply voltage) OK.	Bit #14 (B26)	= 1: Servo board operation temperature within normal range.	Bit #13 (B25)	= 1: Galvanometer scanner operation temperature within normal range.	Bits #4...12 (B16-24)	Reserved.	Bits #0...3	= 0.
Bit #19(MSB) (B31)	= 1: AD converter supply voltage (5 V) OK.																				
Bit #18 (B30)	= 1: Analog section voltage (9 V) OK.																				
Bit #17 (B29)	= 1: DSP IO voltage (3.3 V) OK.																				
Bit #16 (B28)	= 1: DSP core supply voltage (1.8 V) OK.																				
Bit #15 (B27)	= 1: AGC voltage (PD supply voltage) OK.																				
Bit #14 (B26)	= 1: Servo board operation temperature within normal range.																				
Bit #13 (B25)	= 1: Galvanometer scanner operation temperature within normal range.																				
Bits #4...12 (B16-24)	Reserved.																				
Bits #0...3	= 0.																				
		2A _H	<p>Stop Event Code</p> <table> <tr><td>= 00010_H:</td><td>The galvanometer scanner has reached a critical edge position.</td></tr> <tr><td>= 00020_H:</td><td>AD converter error.</td></tr> <tr><td>= 00030_H:</td><td>Temperature in scan system above max. allowed value.</td></tr> <tr><td>= 00040_H:</td><td>External power supply voltages have dropped below the allowed value.</td></tr> <tr><td>= 00050_H:</td><td>Flags are not valid.</td></tr> <tr><td>= 00060_H - 000C0_H:</td><td>Reserved.</td></tr> <tr><td>= 000D0_H:</td><td>Watchdog 10 µs time out (loop time exceeded).</td></tr> <tr><td>= 000E0_H:</td><td>Position Acknowledge time out (set position not reached for long time).</td></tr> <tr><td>= 000F0_H:</td><td>Reserved.</td></tr> </table>	= 00010 _H :	The galvanometer scanner has reached a critical edge position.	= 00020 _H :	AD converter error.	= 00030 _H :	Temperature in scan system above max. allowed value.	= 00040 _H :	External power supply voltages have dropped below the allowed value.	= 00050 _H :	Flags are not valid.	= 00060 _H - 000C0 _H :	Reserved.	= 000D0 _H :	Watchdog 10 µs time out (loop time exceeded).	= 000E0 _H :	Position Acknowledge time out (set position not reached for long time).	= 000F0 _H :	Reserved.
= 00010 _H :	The galvanometer scanner has reached a critical edge position.																				
= 00020 _H :	AD converter error.																				
= 00030 _H :	Temperature in scan system above max. allowed value.																				
= 00040 _H :	External power supply voltages have dropped below the allowed value.																				
= 00050 _H :	Flags are not valid.																				
= 00060 _H - 000C0 _H :	Reserved.																				
= 000D0 _H :	Watchdog 10 µs time out (loop time exceeded).																				
= 000E0 _H :	Position Acknowledge time out (set position not reached for long time).																				
= 000F0 _H :	Reserved.																				
		2B _H	Operational state at the moment of the most recently occurred operation interruption (flags B15 ... B0, see Data = 0528 _H)																		
		2C _H	Operational state at the moment of the most recently occurred operation interruption (flags B31 ... B16, see Data = 0529 _H)																		
		2F _H	<p>Running time (seconds) (as of Firmware Version 2061)</p> <table> <tr><td>Bits #4...19</td><td>Running time (seconds) / s [0 ... 59].</td></tr> <tr><td>Bits #0...3</td><td>= 0.</td></tr> </table>	Bits #4...19	Running time (seconds) / s [0 ... 59].	Bits #0...3	= 0.														
Bits #4...19	Running time (seconds) / s [0 ... 59].																				
Bits #0...3	= 0.																				
		30 _H	<p>Running time (minutes) (as of Firmware Version 2061)</p> <table> <tr><td>Bits #4...19</td><td>Running time (minutes) / min [0 ... 59].</td></tr> <tr><td>Bits #0...3</td><td>= 0.</td></tr> </table>	Bits #4...19	Running time (minutes) / min [0 ... 59].	Bits #0...3	= 0.														
Bits #4...19	Running time (minutes) / min [0 ... 59].																				
Bits #0...3	= 0.																				
		31 _H	<p>Running time (hours) (as of Firmware Version 2061)</p> <table> <tr><td>Bits #4...19</td><td>Running time (hours) / h [0 ... 23].</td></tr> <tr><td>Bits #0...3</td><td>= 0.</td></tr> </table>	Bits #4...19	Running time (hours) / h [0 ... 23].	Bits #0...3	= 0.														
Bits #4...19	Running time (hours) / h [0 ... 23].																				
Bits #0...3	= 0.																				
		32 _H	<p>Running time (days) (as of Firmware Version 2061)</p> <table> <tr><td>Bits #4...19</td><td>Running time (days) / d [0 ... 2¹⁶-1].</td></tr> <tr><td>Bits #0...3</td><td>= 0.</td></tr> </table>	Bits #4...19	Running time (days) / d [0 ... 2 ¹⁶ -1].	Bits #0...3	= 0.														
Bits #4...19	Running time (days) / d [0 ... 2 ¹⁶ -1].																				
Bits #0...3	= 0.																				
The data types marked in grey are only relevant for the intelliWELD.																					
		33 _H	<p>3.3 V sensor board operating voltage</p> <table> <tr><td>Bits #4...19</td><td>Operating voltage / 10⁻³ V [0 ... 7000].</td></tr> <tr><td>Bits #0...3</td><td>= 0.</td></tr> </table>	Bits #4...19	Operating voltage / 10 ⁻³ V [0 ... 7000].	Bits #0...3	= 0.														
Bits #4...19	Operating voltage / 10 ⁻³ V [0 ... 7000].																				
Bits #0...3	= 0.																				

Ctrl Command	control_command		
	(05 _H)	Code _L	Returned Data Signal Type
		34 _H	Sensor board operating temperature Bits #4...19 Operating temperature / 0.1°C [0 ... 65535]. Bits #0...3 = 0.
		35 _H	Emerging-beam-opening temperature Bits #4...19 Temperature / 0.05°C [0 ... 2250]. Bits #0...3 = 0.
		36 _H	Temperature of mirror 2 Bits #4...19 (Temperature + 26.6°C) / 0.05°C [0 ... 1992]. Bits #0...3 = 0.
		37 _H	Temperature of mirror 1 Bits #4...19 (Temperature + 26.6°C) / 0.05°C [0 ... 1992]. Bits #0...3 = 0.
		38 _H	Protective-window temperature Bits #4...19 Temperature / 0.044°C [0 ... 2250]. Bits #0...3 = 0.
		39 _H	Collimator temperature Bits #4...19 Temperature / 0.05°C [0 ... 2250]. Bits #0...3 = 0.
		3A _H	Galvanometer-mount temperature Bits #4...19 Temperature / 0.05°C [0 ... 2250]. Bits #0...3 = 0.
		3B _H	Coolant-flow rate Bits #4...19 (Coolant-flow rate + 0.93 l × min ⁻¹) / (0.0052 l × min ⁻¹) [0 ... 2250]. Bits #0...3 = 0.
		3C _H	Protective-window scattered light value Bits #4...19 Scattered light value / 0.00444 V [0 ... 2250]. Bits #0...3 = 0.
		3D _H	Flags sensor board (*) if "1" then an INTERLOCK error is initiated Bit #19 (MSB) = 1: Protective-window temperature value not in [0 ... 1882]. Bit #18 = 1: Temperature of mirror 1 value not in [0 ... 2250]. Bit #17 = 1: Temperature of mirror 2 value not in [0 ... 2250]. Bit #14 (*) = 1: Emerging-beam-opening temperature value not in [0 ... 1200]. Bit #7 = 1: Protective-window scattered light value not in [0 ... 2250]. Bit #6 = 1: Coolant-flow rate value not in [750 ... 2250]. Bit #5 (*) = 1: Galvanometer-mount temperature value not in [0 ... 1600]. Bit #4 (*) = 1: Collimator temperature value not in [0 ... 2000]. Bits #0...3 = 0.
		3F _H	Position value scale factor setting (as of Firmware Version 2072) Bits #6...19 Reserved. Bits #4, 5 Scale factor = 1/2 ⁿ with n = 2 × bit #5 + bit #4. Bits #0...3 = 0.

Ctrl Command	control_command							
	Code _H	Command and Parameter Values (Code _L)						
	0A _H	<p>UpdatePermanentMemory (as of Firmware Version 2078): This command (with its only allowed parameter value of Code_L = 00_H) causes the currently set servo behavior to also be the default start behavior following subsequent resets. See also chapter 8.1.8 "Configuring the Start Behavior", page 182.</p>						
	0E _H	<p>SetControlDefinitionMode This command specifies which information about a particular tuning (i.e. the corresponding servo algorithm) is to be returned from the scan system by the selected status channel. Code_L [0...3] thereby defines the tuning number. As with the SetMode command, tuning information are returned as signed 20-bit values.</p>						
		Code _L Data Signal Returned from Scan System						
	00 _H	<table> <tr> <td>Bit #19 (MSB)</td> <td>= 0: Tuning available. = 1: Tuning not available.</td> </tr> <tr> <td>...</td> <td></td> </tr> </table>	Bit #19 (MSB)	= 0: Tuning available. = 1: Tuning not available.	...			
Bit #19 (MSB)	= 0: Tuning available. = 1: Tuning not available.							
...								
	03 _H	<table> <tr> <td>Bits #8...18</td> <td>Reserved.</td> </tr> <tr> <td>Bits #4...7</td> <td>Tuning type: = 0: Vector tuning (microvectorization). = 1: Jump tuning. = 2,3: Reserved.</td> </tr> <tr> <td>Bits #0...3</td> <td>= 0.</td> </tr> </table>	Bits #8...18	Reserved.	Bits #4...7	Tuning type: = 0: Vector tuning (microvectorization). = 1: Jump tuning. = 2,3: Reserved.	Bits #0...3	= 0.
Bits #8...18	Reserved.							
Bits #4...7	Tuning type: = 0: Vector tuning (microvectorization). = 1: Jump tuning. = 2,3: Reserved.							
Bits #0...3	= 0.							
	11 _H	<p>SelectControlDefinition (as of Firmware Version 2078): For scan systems equipped with multiple tunings (i.e. servo algorithms), this command selects a desired tuning. See also chapter 8.1.4 "Configuring Dynamics Settings (Tuning)", page 176. Code_L thereby defines the tuning number. Default setting: Code_L = 00_H.</p>						
	12 _H	<p>SetPositionScale (as of Firmware Version 2072): This command can be used to set a (down) scale factor for the position values (the servo electronic scales the position values received from the RTC5 by the specified factor, see also chapter 8.1.7 "Configuring the Effective Calibration", page 181). The command must be issued initially with the parameter Code_L = 83_H and a second time with the desired parameter value (see the following list). In the default setting (Code_L = 00_H), the scale factor is 1 (no scaling).</p>						
		Code _L Effect on the scale factor						
	00 _H	The scale factor is set to the value 1 (but first execute the command code Data = 1283 _H !)						
	01 _H	The scale factor is set to the value 1/2 (but first execute the command code Data = 1283 _H !)						
	02 _H	The scale factor is set to the value 1/4 (but first execute the command code Data = 1283 _H !)						
	03 _H	The scale factor is set to the value 1/8 (but first execute the command code Data = 1283 _H !)						



Ctrl Command	control_command	
	Code _H	Command and Parameter Values (Code _L)
	15 _H	<p>SetPosAcknowledgelevel (as of Firmware Version 2066): This command sets the PosAcknowledge threshold value. See also chapter 8.1.6 "Configuring the PosAcknowledge Threshold Value", page 181. The parameter value Code_L is the desired PosAcknowledge threshold value in bits [00_H ... FF_H]. See also comments page 292.</p>
	17 _H	<p>Store/RestoreTransmissionMode (as of Firmware Version 2078): This command allows the currently used status return data type to be temporarily stored (Code_L = FF_H) and then reinstated at a later time (Code_L = 00_H). See also Code = 0527_H.</p>
	21 _H	<p>SetEchoMode (as of Firmware Version 2078): This command verifies whether data transfer is intact. See also chapter 8.1.9 "Fault Diagnosis and Functional Test", page 182. It transfers an 8-bit value (Code_L) to the scan system and causes a 20-bit value to be returned on the corresponding status channel: if data transfer is error-free, then the upper 8 bits of the returned 20-bit value are identical with Code_L and the next lower 8 bits are identical with the complement of Code_L (NOT Code_L). Example: For <code>control_command(1, 1, 0x210A)</code> and if data transfer is error-free, (<code>get_value(1) AND 0xFFFF0</code>) returns 0xAF50.</p>
Comments	<p><i>General comments:</i></p> <ul style="list-style-type: none"> The control_command command can only be used in conjunction with <i>iDRI^E</i> scan systems (<i>intelliSCAN</i>, <i>intelliSCAN_{de}</i>, <i>intelliDRILL</i>, <i>intellicube</i>, <i>intelliWELD</i>, <i>varioSCAN_{de}</i>). Conventional scan systems without <i>iDRI^E</i> technology ignore the command. With invalid values of Head and/or Axis, the command is <i>not</i> executed (get_last_error return code: RTC5_PARAM_ERROR). This also applies for Head = 2 if the “second scan head control” option has not been enabled. Command code Data is transmitted to the scan system instead of the usual position data for Head and Axis. Therefore, the corresponding galvanometer scanner microstep is omitted, if the command is called during execution of a list. Some parameters of control_command are not usable with older firmware versions of “intelli” scan systems. These parameters are specifically noted. All other parameters are usable with any firmware version. Under some circumstances, control_command might be unavailable at the primary scan head connector if speed-dependent laser control has been activated by set_auto_laser_control (Mode = 2). 	



Ctrl Command	control_command
Comments (cont'd)	<p><i>Comments regarding the commands SetMode (Code_H = 05_H), SetControlDefinitionMode (Code_H = 0E_H), SetEchoMode (Code_H = 21_H) and RestoreTransmissionMode (Data = 1700_H):</i></p> <ul style="list-style-type: none"> The data type selected by the control_command command (Code_H = 05_H, 0E_H or 21_H or Data = 1700_H) is transmitted until another data type is selected. Data returned to the RTC5 can be queried by the commands get_value, get_values, set_trigger/set_trigger4 and get_waveform. Switching to a different data source causes a short (serial transmission-related) delay before transmission of the first data. After switching data sources, therefore, a delay time of up to 60 µs can occur before reading the data. As of DLL version 520, control_command always automatically inserts a waiting time of 60 µs after the data source is switched (so that the previously mentioned commands now always return correct values). All data returned from the scan system are transmitted to the RTC5 as signed 20-bit values. This applies even if the DLL is set to RTC4 compatibility mode and even for scan systems without SL2-100 interface, controlled by an XY2-100 converter. Queried data returned by the commands get_value, get_values or get_waveform are nevertheless generally transferred to the PC as 32-bit signed values (for data evaluation, see comments for get_value). For scan systems with integrated SL2-100 interface, get_head_status queries the XY2-100 status word regardless of settings made by control_command (Code_H = 05_H, 0E_H or 21_H or Data = 1700_H). It is returned in addition to the selected data. In contrast, if iDRIVE scan systems (<i>without</i> an integrated SL2-100 interface) are controlled by an XY2-100 converter, get_head_status returns the XY2-100 status word only if this signal was previously selected to be returned from the scan system by control_command (see also page 173). After a reset or power-up of the scan system, it can take around 5 seconds for data to be returned from the scan system (see also get_value). After a reset or power-up of the scan system, always the XY2-100 status word is returned. control_command (Data = 1700_H) has no effect if a power-up or reset was executed after the most recent execution of the StoreTransmissionMode command (Data = 17FF_H).



Ctrl Command	<code>control_command</code>
Comments (cont'd)	<p><i>Comments regarding the SetMode command (Code_H = 05_H):</i></p> <ul style="list-style-type: none"> Set (angular) position values returned by the scan system correspond to the effective output values Sample<AX..BY>_Out with set_trigger/set_trigger4 and take into account any defined wobbel and Processing-on-the-fly corrections, coordinate transformations, image field correction as well as offset and gain compensations for automatic self-calibration of the scan system. Additionally, the 20-bit set position values returned from a Z axis (and in RTC4 compatibility mode from the X and Y axes, too) are scaled by a factor 16 relating to the therefore specified 16-bit coordinate values. During a temporary error state after the maximum allowed temperature was exceeded (flag bit#7 = 0), the scan head's output stages of the affected axis are at least temporarily deactivated. The scan system returns to normal operation as soon as the temperature drops again below the maximum allowed temperature. If a critical error occurs (flag bit#9 = 0), the scan system automatically enters a permanent error state, in which the output stages of the affected axis remain deactivated – even if the critical error was only temporarily present. Normal operation is <i>not</i> resumed. Flag bit#9 is only reset by a hardware reset. Critical errors are for instance improper internal voltages (flag bit#13 = 0), external power supply interruption (flag bit#8 = 0) or reaching a critical edge position (flag bit#5 = 0). During both temporary and permanent error states, the scan system continues to transmit data to the control board. Even in these states, switching or selection of data signals for diagnostic purposes is still possible. The flags indicate the scan system's operational state. The scan module can return one of two flag blocks indicating the current operational state (Data = 0528_H, 0529_H) or alternatively one of two further flag blocks indicating the operational state at the moment of the most recently occurred operation interruption (Data = 052B_H, 052C_H). After every successful restart – and, as long as no error has occurred – all status information of the two latter blocks (Data = 052B_H, 052C_H) are irrelevant. Only, as soon as an error causes a switch into a temporary or permanent error state, the current status values are saved into these two blocks. In this case, also an event code is simultaneously set, indicating which particular event caused the error state. This event code can be read out separately (Data = 052A_H). The angle bit-values (actual position, set position, position error) or angle bit/ms-values (actual speed) returned to the RTC5 can be converted into °-values or °/ms-values by the scan system's calibration angle. The calibration angle can be read out by Data = 0523_H. Exact values for the internal voltages referred to in the table can vary for different versions of the scan system. intelliWELD scan systems can be supplied with various number of sensors. Reference the scan system's user manual. This RTC5 manual lists the command codes for all its theoretically possible sensors.



Ctrl Command	control_command
Comments (cont'd)	<p><i>Notes on the SetPositionScale command (Code_H = 12_H):</i></p> <ul style="list-style-type: none"> If the scale factor for scaling the position values is changed, then the user program calibration factor for calculating RTC5 control values (in bits) from the desired image field position values (in mm) (see chapter 7.3.2 "Image Field Size and Calibration", page 134) needs to be subjected to an inverse proportional scaling. In addition, the jump and marking speed should be adjusted. The currently set scaling factor can be queried with Data = 053F_H. <p><i>Note on the SetPosAcknowledgelevel command (Code_H = 15_H):</i></p> <ul style="list-style-type: none"> The threshold value must be specified related to a 16-bit position range. This value is converted by the RTC5 to a value related to a 20-bit position range by multiplying by 16 (and if applicable reconverted by the XY2-100 converter). The default start behavior is for the scan system to set the threshold value to 0.28% of the full position range after every power-up or reset (for iDRI^E scan systems with SL2-100 interface, this corresponds to 0.28% of 2²⁰ bits, for iDRI^E scan systems with XY2-100 Enhanced interface to 0.28% of 2¹⁶ bits i.e. 183 bits; in both cases, this corresponds to Code_L = 183 = B7_H). If other threshold values are desired, they must be separately set for each axis (Code_H = 15_H). SCANLAB recommends setting only threshold values (Code_L) above 14_H (i.e. 0.03% of the full position range). Lower values can lead to frequent system safety shutdowns due to Position Acknowledge time outs (set position not reached for an excessive time).
RTC4 → RTC5	Unchanged functionality. Note that all returned data selected by control_command are always in the RTC5's 20-bit range, even in RTC4 compatibility mode (see also comments above and comments for get_value).
Version info	Last change with version DLL 520 (see page 693).
References	get_value , get_values , get_head_status , set_trigger , set_trigger4 , get_waveform



Ctrl Command	copy_dst_src
Function	Creates entries in the internal management table for an indexed character, text string or subroutine with the specified index (<code>Dst</code>) by copying the table entries of another index (<code>Src</code>).
Call	<code>copy_dst_src(Dst, Src, Mode)</code>
Parameters	<p>Dst Index (as an unsigned 32-bit value) of the indexed character, text string or subroutine whose entries should be copied from <code>Src</code>. Allowed range: [0 ... 1023] for indexed characters or subroutines, [1024+0 ... 1024+41] for indexed text strings.</p> <p>Src Index (as an unsigned 32-bit value) of the indexed character, text string or subroutine whose entries should be copied to <code>Dst</code>. Allowed range: [0 ... 1023] for indexed characters or subroutines, [1024+0 ... 1024+41] for indexed text strings.</p> <p>Mode This parameter (unsigned 32-bit value) determines which internal management tables should be changed: Bit #0 = 0:Dst is the index of an indexed character or text string. = 1:Dst is the index of an indexed subroutine. Bit #1 = 0:Src is the index of an indexed character or text string. = 1:Src is the index of an indexed subroutine. Bits #2-31:Are not evaluated.</p>
Comments	<ul style="list-style-type: none"> If an index value (<code>Dst</code> and/or <code>Src</code>) is invalid, then the command is ignored (get_last_error return code: <code>RTC5_PARAM_ERROR</code>). <code>copy_dst_src</code> creates an additional reference (index) to an indexed character, text string or subroutine (that can also be called with this new index). The command only alters the corresponding entry in the internal management table and does not modify the list buffer's memory contents. This allows copying, renumbering or converting between indexed characters, text strings or subroutines without having to reload each time. A real copy of an indexed character, text string or subroutine in the protected buffer area can be created (after the <code>copy_dst_src</code> command) with save_disk/load_disk. Characters, text strings and/or subroutines with multiple references are thereby written several times to the list memory. Keep this in mind in order to prevent unintended buffer overflow of the protected buffer area. See also section "Management of Indexed Characters and Text Strings", page 89.
RTC4→ RTC5	New command.
References	save_disk , load_disk

Ctrl Command	disable_laser
Function	Disables the laser control signals for "laser active" operation.
Call	<code>disable_laser()</code>
Comments	<ul style="list-style-type: none"> The command disable_laser disables the laser control signals for "laser active" operation at the output ports LASER1, LASER2 and LASERON (then the output ports are in the high impedance tristate mode). Pin (2) of the 15-pin laser connector is then at a fixed level. However, standby signals activated with set_standby or set_standby_list continue to be outputted by the LASER1 and LASER2 output ports. If the standby signals are deactivated, also the pins (1) and (9) of the 15-pin laser connector and pins (19) and (22) of the (on-board) EXTENSION 2 socket connector are at a fixed level after disable_laser. The laser control signals for "laser active" operation can also be disabled by set_laser_control and reenabled by set_laser_control or enable_laser. After initialization of the RTC5 with load_program_file, the laser control is deactivated and absolutely requires set_laser_control for activation. If the command results in an RTC5_TIMEOUT error (e.g. if no program file was loaded), the LASER1, LASER2 and LASERON output ports can only be reactivated with set_laser_control after a load_program_file command. The command get_startstop_info (Bit #9) queries the current status of the laser control signals.
RTC4→ RTC5	Unchanged.
References	enable_laser, set_laser_control, get_startstop_info

Ctrl Command	enable_laser
Function	Enables the laser control signals for "laser active" operation.
Call	<code>enable_laser()</code>
Comments	<ul style="list-style-type: none"> After a hardware reset (and after load_program_file), the command set_laser_control must be called to activate the LASER1, LASER2 and LASERON output ports and define the signal level (see chapter 7.4 "Laser Control", page 144). Otherwise, the command enable_laser has no effect. After initialization of the RTC5 with load_program_file, the laser control signals are deactivated. For first-time activation, set_laser_control must be called (see above). After a subsequent disabling by disable_laser (or set_laser_control), the enable_laser (or set_laser_control) command can be used for reenabling. The command get_startstop_info (Bit #9) queries the current status of the laser control signals. Even if the laser control signals have been enabled with enable_laser or set_laser_control, they are not outputted without further commands (see chapter 7.4 "Laser Control", page 144).
RTC4→ RTC5	Essentially unchanged functionality. In some circumstances, the command set_laser_control must be called before enable_laser (see above).
References	disable_laser, set_laser_control, get_startstop_info, set_laser_mode



Ctrl Command	execute_at_pointer
Function	Starts list execution ("List 1" or "List 2") at the specified address in the RTC5 list buffer.
Call	<code>execute_at_pointer(Pos)</code>
Parameter	Pos Absolute address of the first list command to be executed as an unsigned 32-bit value. Allowed range: [0 ... (2^{20} -1)].
Comments	<ul style="list-style-type: none"> The <code>execute_at_pointer</code> command essentially functions like the <code>execute_list_pos</code> command (see the comments there). However, <code>execute_at_pointer</code> requires a start address specified as an <i>absolute</i> memory address, whereas <code>execute_list_pos</code> requires specification of the list number and a <i>relative</i> memory address. For $\text{Pos} \geq \text{Mem1} + \text{Mem2}$ (see <code>config_list</code>), Pos is set to 0.
RTC4→ RTC5	Unchanged functionality.
References	execute_list_pos , get_out_pointer

Ctrl Command	execute_list
Function	Starts execution at the beginning of the specified list ("List 1" or "List 2").
Call	<code>execute_list(ListNo)</code>
Parameter	ListNo Number of the list to be executed as an unsigned 32-bit value. Allowed values: [uneven: "List 1", even: "List 2"].
Comments	<ul style="list-style-type: none"> The command <code>execute_list</code> is synonymous with <code>execute_list_pos</code> with $\text{Pos} = 0$. The commands <code>execute_list_1</code> and <code>execute_list_2</code> (with no parameters) can be used alternatively.
RTC4→ RTC5	Unchanged functionality.
References	get_status , execute_at_pointer , execute_list_pos

Ctrl Command	execute_list_1
Function	See execute_list .
Call	<code>execute_list_1</code>
RTC4→ RTC5	Unchanged functionality.
References	execute_list

Ctrl Command	execute_list_2
Function	See execute_list .
Call	<code>execute_list_2</code>
RTC4→ RTC5	Unchanged functionality.
References	execute_list



Ctrl Command	execute_list_pos
Function	Starts list execution ("List 1" or "List 2") at the specified position.
Call	<code>execute_list_pos(ListNo, Pos)</code>
Parameters	<p>ListNo Number of the list to be executed as an unsigned 32-bit value. Allowed values: [uneven: "List 1", even: "List 2"].</p> <p>Pos Address of the first list command to be executed (offset relative to the start of the respective list) as an unsigned 32-bit value. Allowed range: [0 ... (2^{32}-1)].</p>
Comments	<ul style="list-style-type: none"> The command is ignored (<code>get_error</code> return value = RTC5_BUSY) if the board's BUSY status or PAUSED status is currently set (list is being processed or has been halted by <code>pause_list</code> or <code>set_wait</code>). If the board's INTERNAL-BUSY status is currently set, <code>execute_list_pos</code> is only executed with a delay (after INTERNAL-BUSY has been reset again). No checks are performed to determine if a list is currently being loaded. During list processing, the other list (or even the same list) can be simultaneously reloaded (see also "List Handling", page 75). Programs in the protected area ("List 3") cannot be directly executed by <code>execute_list_pos</code>. They can only be called from a list ("List 1" or "List 2") as a subroutine. Alternatively, the corresponding area can be assigned by <code>config_list</code> to "List 1" or "List 2". Uneven <code>ListNo</code> values cause "List 1" to be executed; otherwise "List 2" is executed. This allows automatically generated continuous list changing by an incremented count. If "List 2" has not been assigned memory (<code>Mem2 = 0</code>, see <code>config_list</code>) then "List 1" is opened. If <code>Pos</code> is specified as being larger than the memory area of the respective list (<code>Pos > Mem1</code> or <code>Pos > Mem2</code>), then <code>Pos</code> is set to 0. The BUSY list status of the selected list is set and the BUSY list status of the other corresponding list is reset (see <code>read_status</code>). The BUSY list execution status (see <code>get_status</code>) is set. Execution stops when a <code>set_end_of_list</code> command is encountered. If the end of a list area is reached without encountering a <code>set_end_of_list</code> command, then execution continues at the beginning of the same list area instead of with the next list. The output pointer remains in the active list area unless a <code>set_end_of_list</code> command was encountered and an <code>auto_change_pos</code> or <code>start_loop</code> command was previously issued. For both lists to be treated as a single list, you must set the configuration appropriately: e.g. (<code>Mem1+Mem2, 0</code>). If a home jump (defined with <code>home_position</code> or <code>home_position_xyz</code>) was executed by <code>set_end_of_list</code>, then <code>execute_list_pos</code> leads to a corresponding home return (the INTERNAL-BUSY status is set while the home return is executed). The <code>execute_list_pos</code> command triggers a flush of the list input buffer (see page 75) even if the start was unsuccessful. <code>execute_list_pos</code> also covers the specialized variants <code>execute_list_1</code>, <code>execute_list_2</code>, <code>execute_list</code> and <code>execute_at_pointer</code>.



Ctrl Command	execute_list_pos
RTC4→ RTC5	New command.
References	execute_list , execute_at_pointer , set_start_list_pos

Normal List Command	fly_return
Function	Deactivates the previously set Processing-on-the-fly correction (for multiple directions in space, if necessary) and subsequently executes a jump to the defined position.
Restriction	If the Processing-on-the-fly option is not enabled, the command only executes the jump to the defined new output position.
Call	<code>fly_return(X, Y)</code>
Parameter	X, Y Absolute coordinates of the new output position in <i>bits</i> as signed 32-bit values. Allowed range: [-524288 ... 524287]. Out-of-range values are edge-clipped.
Comments	<ul style="list-style-type: none"> The jump to the new output position is executed as with jump_abs (see comments there). If Processing-on-the-fly-correction was activated within a subroutine called by an "AbsCall" and subsequently gets deactivated by fly_return, then the coordinate values specified with fly_return receive an offset (based on the current coordinates at the time of the call, see also section "AbsCalls", page 83). See also chapter 8.7 "Processing-on-the-fly (Optional)", page 199.
RTC4→ RTC5	Unchanged functionality (except for the extended range of values and "AbsCall", see above). The coordinates of the new output position are specified as 20-bit values; in RTC4 compatibility mode they are specified as 16-bit values (as with the RTC4) and the RTC5 multiplies the specified coordinate values by 16. The allowed range of values is correspondingly reduced to [-32768 ... 32767].
References	set_fly_x , set_fly_y , set_fly_rot , set_fly_x_pos , set_fly_y_pos , set_fly_rot_pos



Normal List Command	fly_return_z
Function	Deactivates the previously set Processing-on-the-fly correction (for multiple directions in space, if necessary) and subsequently executes a jump to the defined position.
Restriction	If the Processing-on-the-fly option is not enabled, the command only executes the jump to the defined new output position.
Call	<code>fly_return_z(X, Y, Z)</code>
Parameter	X, Y, Z Absolute coordinates of the new output position in <i>bits</i> as signed 32-bit values. Allowed range: <ul style="list-style-type: none"> • For X and Y: [-524288 ... 524287]. • For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped.
Comments	<ul style="list-style-type: none"> • The jump to the new output position is executed as with jump_abs_3d (see comments there). • If Processing-on-the-fly-correction was activated within a subroutine called by an "AbsCall" and subsequently gets deactivated by fly_return_z, then the coordinate values specified with fly_return_z receive an offset (based on the current coordinates at the time of the call, see also chapter 6.5.1 "Subroutines", section ""AbsCalls"", page 83). • See also chapter 8.7 "Processing-on-the-fly (Optional)", page 199.
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced to [-32768 ... 32767]). The value range for Z coordinates is identical for the RTC5 and RTC4.
Version info	Available as of version DLL 531, OUT 532.
References	set_fly_z



Ctrl Command	free_RTC5_dll
Function	Frees up all resources allocated by the DLL for a user program.
Call	<code>free_RTC5_dll()</code>
Comments	<ul style="list-style-type: none"> DLL-allocated resources particularly include board-management memory in the DLL allocated as the result of the init_RTC5_dll command. <code>free_RTC5_dll</code> deletes DLL board management. Afterward, the user program has no access to boards (get_last_error return code: <code>RTC5_ACCESS_DENIED</code>). The <code>free_RTC5_dll</code> command does not cause RTC5 Board resets. Board resets can only be initiated by the command load_program_file. The calling of <code>free_RTC5_dll</code> is not absolutely necessary, because DLL-assigned resources are automatically freed up when the user program terminates and the DLL is thereby unloaded by Microsoft Windows. However, some user program development environments (in debug mode) issue "memory leaks detected" warnings even though the DLL is unloaded. The calling of <code>free_RTC5_dll</code> eliminates this annoyance.
RTC4→RTC5	New command.
References	init_RTC5_dll , release_RTC

Ctrl Command	get_auto_cal
Function	Returns the attached scan system's type of ASC hardware previously detected by auto_cal .
Call	<code>ASCTYPE = get_auto_cal(HeadNo)</code>
Parameters	<code>HeadNo</code> Number of the scan head connector as an unsigned 32-bit value. Allowed values: = 1: Primary scan head connector. = 2: Secondary scan head connector (activation required).
Result	ASC hardware type as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> If the ASC hardware type was previously detected by auto_cal, then <code>get_auto_cal</code> returns the same value as auto_cal(Command = 4) – see comments there. If the ASC hardware type was not previously detected by auto_cal, then <code>get_auto_cal</code> returns the value 255 (initialized value).
RTC4→RTC5	New command.
Version info	Available as of version DLL 520, OUT 519.
References	auto_cal



Ctrl Command	get_char_pointer
Function	Returns the absolute start address of an indexed character.
Call	CharPointer = get_char_pointer(Char)
Parameter	Char Index of the indexed character as an unsigned 32-bit value. Allowed range: [0 ... 1023].
Result	Absolute start address as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The get_char_pointer command reads from the internal management table the start address of the indexed character with the specified index. Whether the read address resides in a protected or the unprotected memory area depends on whether the character was loaded into the protected memory area or an unprotected subroutine was only subsequently referenced. If Index > 1023 or if no character was referenced with the specified index, then the command returns the value “-1” (e.g. $2^{32}-1$). This command is useful for checking if a character has already been defined or for calling an indexed character by an absolute memory address as if it were a non-indexed subroutine, e.g. for conditional execution with list_call_cond. Be aware, though, that a subsequent save_disk/load_disk might alter the absolute memory address. And you should ensure that get_char_pointer does not return “-1”; otherwise the list_call_cond command is ignored.
RTC4→ RTC5	New command.
References	get_sub_pointer, get_text_table_pointer

Ctrl Command	get_config_list
Function	Passes the parameters of the current list memory configuration (Mem1, Mem2) to the DLL's board management and initializes it as if the config_list command was called.
Call	get_config_list()
Comments	<ul style="list-style-type: none"> The get_config_list command is useful when a board changes “ownership” and the new board management is not aware of the memory configuration (at the start of each user program, the board and board management each independently initialize Mem1 = 4000 and Mem2 = 4000; the board by load_program_file and board management when starting the corresponding user program). See also “Board Acquisition by a User Program”, page 96. The get_config_list command does not return a value to the user program. The user program can, however, read the list-memory configuration data after load_list(ListNo, 0) or set_start_list_pos(ListNo, 0) by using the get_list_space command. The get_config_list command is executed regardless of the board's BUSY status.
RTC4→ RTC5	New command.
References	config_list



Ctrl Command	get_counts
Function	Reads the current number of successful external list starts.
Call	Counts = get_counts()
Result	Number of successful external list starts as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The number is read from an internal counter, which is incremented each time a list is started by an external start signal. The counter can be reset to zero by set_control_mode.
RTC4→ RTC5	Unchanged.
References	set_max_counts , set_control_mode , get_startstop_info

Ctrl Command	get_dll_version
Function	Returns the version number of the RTC5 DLL.
Call	DLLVersion = get_dll_version()
Result	DLL version number as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The RTC5 DLL version numbers are in the range 500-599. The command get_dll_version is also available without explicit access rights to a specific RTC5 Board. get_dll_version is not available as a multi-board command. The board-specific error variables <code>LastError</code> and <code>AccError</code> (see "Error Handling", page 98) are neither generated nor altered by get_dll_version.
RTC4→ RTC5	Unchanged functionality.
References	get_hex_version , get_RTC_version

Ctrl Command	get_encoder
Function	Returns the current counts of the two internal encoder counters.
Call	get_encoder(&Encoder0, &Encoder1)
Returned parameter values	Encoder0, current counts as pointers to signed 32-bit values. Encoder1
Comments	<ul style="list-style-type: none"> For usage of this command see "Processing-on-the-fly (Optional)", page 199 and "Synchronization by Encoder Signals", page 246. If incremental encoders are used to detect the motion of the parts to be processed, encoder counter Encoder0 is triggered by the signals at encoder input ENCODER X and encoder counter Encoder1 by the signals at encoder input ENCODER Y. In contrast, if an encoder simulation has been started by simulate_encoder, both encoder counters are triggered by an internal periodic 1 MHz clock signal.
RTC4→ RTC5	Unchanged functionality (except for the extended range of values).
References	store_encoder , read_encoder , set_fly_x , set_fly_y , set_fly_rot , wait_for_encoder



Ctrl Command	get_error																									
Function	Returns the cumulative error code (i.e. a list of error types occurring since the last reset or error reset).																									
Call	AccError = get_error()																									
Result	<p>Error code as an unsigned 32-bit value. If multiple errors occurred, then multiple bits are set. Error constants are predefined for the specific errors.</p> <table> <thead> <tr> <th>Bit</th> <th>Error type</th> <th>Error constant</th> </tr> </thead> <tbody> <tr> <td></td> <td>No error.</td> <td>RTC5_NO_ERROR = 0</td> </tr> <tr> <td>Bit #0 (LSB)</td> <td>= 1: No board found (this error can only occur by init_RTC5_dll).</td> <td>RTC5_NO_CARD = 1</td> </tr> <tr> <td>Bit #1</td> <td>= 1: Access denied (this error can occur by init_RTC5_dll, select_RTC, acquire_RTC or any multi-board command).</td> <td>RTC5_ACCESS_DENIED = 2</td> </tr> <tr> <td>Bit #2</td> <td>= 1: Command not forwarded (this error implies an internal, PCI or driver error, e.g. caused by a hardware defect or an incorrect connection).</td> <td>RTC5_SEND_ERROR = 4</td> </tr> <tr> <td>Bit #3</td> <td>= 1: No response from board (it is likely that no program has been loaded onto the RTC5; this error can especially occur in connection with control commands that expect a response, e.g. get_hex_version).</td> <td>RTC5_TIMEOUT = 8</td> </tr> <tr> <td>Bit #4</td> <td>= 1: Invalid parameter (this error can occur through all commands for which invalid parameters are not automatically corrected to valid values, e.g. parameters with limited choices such as get_head_para; if this error occurs for a list command, it is replaced with list_nop; if this error occurs for a control command, it is not executed).</td> <td>RTC5_PARAM_ERROR = 16</td> </tr> <tr> <td>Bit #5</td> <td>= 1: List processing is (not) active (e.g. for execute_list, if a list is currently being processed e.g. for stop_execution, if no list is currently being processed e.g. for restart_list, if pause_list was not previously called).</td> <td>RTC5_BUSY = 32</td> </tr> </tbody> </table>		Bit	Error type	Error constant		No error.	RTC5_NO_ERROR = 0	Bit #0 (LSB)	= 1: No board found (this error can only occur by init_RTC5_dll).	RTC5_NO_CARD = 1	Bit #1	= 1: Access denied (this error can occur by init_RTC5_dll , select_RTC , acquire_RTC or any multi-board command).	RTC5_ACCESS_DENIED = 2	Bit #2	= 1: Command not forwarded (this error implies an internal, PCI or driver error, e.g. caused by a hardware defect or an incorrect connection).	RTC5_SEND_ERROR = 4	Bit #3	= 1: No response from board (it is likely that no program has been loaded onto the RTC5; this error can especially occur in connection with control commands that expect a response, e.g. get_hex_version).	RTC5_TIMEOUT = 8	Bit #4	= 1: Invalid parameter (this error can occur through all commands for which invalid parameters are not automatically corrected to valid values, e.g. parameters with limited choices such as get_head_para ; if this error occurs for a list command, it is replaced with list_nop ; if this error occurs for a control command, it is not executed).	RTC5_PARAM_ERROR = 16	Bit #5	= 1: List processing is (not) active (e.g. for execute_list , if a list is currently being processed e.g. for stop_execution , if no list is currently being processed e.g. for restart_list , if pause_list was not previously called).	RTC5_BUSY = 32
Bit	Error type	Error constant																								
	No error.	RTC5_NO_ERROR = 0																								
Bit #0 (LSB)	= 1: No board found (this error can only occur by init_RTC5_dll).	RTC5_NO_CARD = 1																								
Bit #1	= 1: Access denied (this error can occur by init_RTC5_dll , select_RTC , acquire_RTC or any multi-board command).	RTC5_ACCESS_DENIED = 2																								
Bit #2	= 1: Command not forwarded (this error implies an internal, PCI or driver error, e.g. caused by a hardware defect or an incorrect connection).	RTC5_SEND_ERROR = 4																								
Bit #3	= 1: No response from board (it is likely that no program has been loaded onto the RTC5; this error can especially occur in connection with control commands that expect a response, e.g. get_hex_version).	RTC5_TIMEOUT = 8																								
Bit #4	= 1: Invalid parameter (this error can occur through all commands for which invalid parameters are not automatically corrected to valid values, e.g. parameters with limited choices such as get_head_para ; if this error occurs for a list command, it is replaced with list_nop ; if this error occurs for a control command, it is not executed).	RTC5_PARAM_ERROR = 16																								
Bit #5	= 1: List processing is (not) active (e.g. for execute_list , if a list is currently being processed e.g. for stop_execution , if no list is currently being processed e.g. for restart_list , if pause_list was not previously called).	RTC5_BUSY = 32																								



Ctrl Command	get_error		
	<p>Bit #6 = 1: List command rejected, illegal input pointer (e.g. for any list command directly after load_char + list_return: the list command is then not loaded).</p> <p>Bit #7 = 1: List command was converted to a list_nop RTC5_IGNORED = 128 (e.g. set_end_of_list in a protected subroutine).</p> <p>Bit #8 = 1: Version error: DLL version (RTC5 DLL), RTC version (firmware file) and HEX version (DSP program file) not compatible (see also load_program_file).</p> <p>Bit #9 = 1: Verify error: The download verification (see page 99) has detected an incorrect download.</p> <p>Bit #10 = 1: DSP version error: DSP version too old (this error only occurs with older RTC5 Boards – see get_RTC_version bits #16-23 – and only through a few commands such as mark_ellipse_abs; the corresponding command description's "Version info" section contains related comments; Commands that generate the error are neither executed nor replaced by list_nop).</p> <p>Bit #11 = 1: A DLL-internal Windows memory request failed. RTC5_OUT_OF_MEMORY = 2048</p> <p>Bit #12 = 1: EEPROM read or write error (can occur during initialization or auto_cal). RTC5_EEPROM_ERROR = 4096</p> <p>Bits #13...#15 Reserved.</p> <p>Bit #16 = 1: Error reading PCI configuration register (can only occur during init_RTC5_dll). RTC5_CONFIG_ERROR = 65536</p> <p>Bits #17...#31 Reserved.</p>		
Comments	<ul style="list-style-type: none"> For error handling see chapter 6.8 "Error Handling", page 98. The commands get_error and n_get_error are also available without explicit access rights to a specific RTC5 Board. The board-specific error variables LastError and AccError (see chapter 6.8 "Error Handling", page 98) are neither generated nor altered by get_error. 		



Ctrl Command	get_error
Example (C/C++)	<p>Creates an array for specifying which board has no existing access rights and resets the cumulative error code.</p> <pre>UINT NoAccess[MaxCount+1]; // MaxCount is a user-defined constant UINT Error = init_RTC5_dll(); // Searches for all installed RTC5 Boards if (Error & RTC5_ACCESS_DENIED) { // at least one board is inaccessible UINT Count = rtc5_count_cards(); // number of boards found for (UINT Num = 1; Num <= Count; Num++) { NoAccess[Num] = n_get_last_error(Num) & RTC5_ACCESS_DENIED; n_reset_error(Num, RTC5_ACCESS_DENIED); } }</pre>
RTC4→RTC5	New command.
Version info	<p>Change with version DLL 518: bit#10.</p> <p>Last change with version DLL 535: bits #11, 12, 16.</p>
References	get_last_error , reset_error , set_verify

Ctrl Command	get_fly_2d_offset
Function	Returns the current reference values (offset values) for 2D encoder compensation.
Call	<code>get_fly_2d_offset(&OffsetX, &OffsetY)</code>
Returned parameter values	OffsetX, Reference values as pointers to signed 32-bit values. OffsetY
Comments	<ul style="list-style-type: none"> For 2D encoder compensation, see section "2D Encoder Compensation for XY Stages", page 206.
RTC4→RTC5	New command.
Version info	Available as of version DLL 536, OUT 536.
References	init_fly_2d , set_fly_2d

Ctrl Command	get_free_variable
Function	Returns the current value of a free variable.
Call	<code>VariableValue = get_free_variable(No)</code>
Parameter	No Number of the free variable to be queried as an unsigned 32-bit value. Allowed range: [0 ... 7]. Only the two least significant bits are evaluated.
Result	The value currently stored in the free variable (as an unsigned 32-bit value).
Comments	<ul style="list-style-type: none"> See also chapter 6.9.1 "Free Variables", page 102.
RTC4→RTC5	New command.
Version info	Available as of version DLL 531, OUT 532. Last change (version DLL 539, OUT 539): value range of parameter No increased to [0 ... 7].
References	set_free_variable , set_free_variable_list



Ctrl Command	get_galvo_controls
Function	Returns the corresponding control values for given input values.
Restriction	The command can only be executed, if no list is currently being processed (get_last_error -return code: RTC5_BUSY).
Call	<code>get_galvo_controls(InPtr, OutPtr)</code>
Parameter	<p>InPtr Pointer (data type ULONG_PTR in C and C++, that is, as an unsigned 32-bit or 64-bit value) to an array of five unsigned 32-bit values, where the desired settings are specified: X, Y, Z, Defocus, Zoom. Out-of-range values are edge-clipped.</p> <p>OutPtr Pointer (data type ULONG_PTR in C and C++, that is, as an unsigned 32-bit or 64-bit value) to an array of four unsigned 32-bit values, where the corresponding control values are to be stored: XA, YA, XB, YB. Range of values in each case is [-524288 ... 524287].</p>
Returned parameter values	XA, YA, XB, YB denote the control values for the XY axes of scan heads A and B.
Comments	<ul style="list-style-type: none"> • get_galvo_controls carries-out a virtual jump to the specified coordinates. Though the galvanometer scanners are <i>not</i> moved. • All other settings which are not specified within the command get_galvo_controls (for example, matrix, offset, angle, etc.; also stretch table) are considered as they are set at the moment. Users are solely responsible to apply these by <code>at_once > 0</code> before the command get_galvo_controls is called. The settings are not used, if they just only have been saved by <code>at_once = 0</code>. • Control values are calculated only for channels where a correction table has been previously assigned by select_cor_table, see the following examples. <pre>select_cor_table(0,0): XA = YA = XB = YB = 0 2D correction files: inputs Z, Defocus, Zoom are ignored select_cor_table(1,0): XA, YA calculated, XB, YB = 0 select_cor_table(0,1): XA, YA = 0, XB, YB calculated select_cor_table(1,1): XA, YA, XB, YB calculated (Standard-)3D correction file: input Zoom is ignored select_cor_table(1,0): XA, YA calculated, XB = YB = Zout calculated select_cor_table(0,1): XA= YA = Zout calculated, XB, YB calculated select_cor_table(1,1): same as select_cor_table(0,0) 3D zoom correction file (only for intelliWELD II with zoom axis): select_cor_table(1,0): XA, YA calculated, XB = Zout, YB = ZoomOut calculated select_cor_table(0,1): XA= Zout, YA = ZoomOut calculated, XB, YB calculated </pre> <ul style="list-style-type: none"> • The return values are 0, if a get_last_error return code RTC5_BUSY was generated.



Ctrl Command	get_galvo_controls
RTC4→ RTC5	<p>New command.</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified X and Y values by 16 (the allowed range of values is correspondingly reduced).</p> <p>Even in RTC4 compatibility mode, all returned values are in the RTC5 20-bit range.</p>
Version info	Available as of version DLL 539, OUT 539.
References	select_cor_table

Ctrl Command	get_head_para				
Function	Returns the value of the requested parameter in the correction table assigned to the specified scan head.				
Call	<code>HeadPara = get_head_para(HeadNo, ParaNo)</code>				
Parameters	<table> <tr> <td>HeadNo</td> <td>Number of the scan head connector as an unsigned 32-bit value. Allowed values: = 1: Primary scan head connector. = 2: Secondary scan head connector (activation required).</td> </tr> <tr> <td>ParaNo</td> <td>Number of the parameter as an unsigned 32-bit value. Allowed values: 0-15 (meaning: see page 140)</td> </tr> </table>	HeadNo	Number of the scan head connector as an unsigned 32-bit value. Allowed values: = 1: Primary scan head connector. = 2: Secondary scan head connector (activation required).	ParaNo	Number of the parameter as an unsigned 32-bit value. Allowed values: 0-15 (meaning: see page 140)
HeadNo	Number of the scan head connector as an unsigned 32-bit value. Allowed values: = 1: Primary scan head connector. = 2: Secondary scan head connector (activation required).				
ParaNo	Number of the parameter as an unsigned 32-bit value. Allowed values: 0-15 (meaning: see page 140)				
Result	Parameter value as a 64-bit IEEE floating point value (see page 140).				
Comments	<ul style="list-style-type: none"> The parameter values can be read out by get_table_para from a currently loaded correction table and by get_head_para from an assigned correction table and thus directly incorporated into a user program (see page 140). If the parameters HeadNo and ParaNo are out of range, then the return value is 0 (get_last_error return code: RTC5_PARAM_ERROR). The return value is also 0 (no get_last_error return code) if no correction table has been assigned to the specified head (e.g. for HeadNo = 2 if the “second scan head control” option has not been enabled) and no 3D table has been assigned to the other head. If a 3D table has been assigned to a head, then this 3D table’s parameter is returned regardless of HeadNo (two 3D tables cannot be simultaneously assigned). HeadNo must nevertheless be 1 or 2 (see preceding comment). 				
RTC4→ RTC5	New command.				
Version info	Last change with version DLL 516, OUT 514.				



Ctrl Command	get_head_status																														
Function	Returns the XY2-100 status word from the specified scan head connector.																														
Call	<code>get_head_status(Head)</code>																														
Parameter	<p>Head = 1: Returns the status of the primary scan head connector (Byte #1 = Byte #0).</p> <p>= 2: Returns the status of the secondary scan head connector (activation required) (Byte #1 = Byte #0).</p> <p>Else: Returns the status of the primary scan head connector (Byte #0) and of the secondary scan head connector (Byte #1).</p>																														
Result	<p>XY2-100 status word (as an unsigned 32-bit value):</p> <table> <tr> <td>Byte #0 (LSB)</td> <td>Bit #0 (LSB)</td> <td>1.</td> </tr> <tr> <td></td> <td>Bit #1</td> <td>0.</td> </tr> <tr> <td></td> <td>Bit #2</td> <td>1 (reserved).</td> </tr> <tr> <td></td> <td>Bit #3</td> <td>Position Acknowledge of X axis, 1 = OK.</td> </tr> <tr> <td></td> <td>Bit #4</td> <td>Position Acknowledge of Y axis, 1 = OK.</td> </tr> <tr> <td></td> <td>Bit #5</td> <td>1 (reserved).</td> </tr> <tr> <td></td> <td>Bit #6</td> <td>Temperature Status, 1 = OK.</td> </tr> <tr> <td></td> <td>Bit #7</td> <td>Power Status, 1 = OK.</td> </tr> <tr> <td>Byte #1</td> <td>Bits #8...# 15</td> <td>Bit assignments as with Byte #0.</td> </tr> <tr> <td>Bytes #2...#3</td> <td>Bits #16...#31</td> <td>0.</td> </tr> </table>	Byte #0 (LSB)	Bit #0 (LSB)	1.		Bit #1	0.		Bit #2	1 (reserved).		Bit #3	Position Acknowledge of X axis, 1 = OK.		Bit #4	Position Acknowledge of Y axis, 1 = OK.		Bit #5	1 (reserved).		Bit #6	Temperature Status, 1 = OK.		Bit #7	Power Status, 1 = OK.	Byte #1	Bits #8...# 15	Bit assignments as with Byte #0.	Bytes #2...#3	Bits #16...#31	0.
Byte #0 (LSB)	Bit #0 (LSB)	1.																													
	Bit #1	0.																													
	Bit #2	1 (reserved).																													
	Bit #3	Position Acknowledge of X axis, 1 = OK.																													
	Bit #4	Position Acknowledge of Y axis, 1 = OK.																													
	Bit #5	1 (reserved).																													
	Bit #6	Temperature Status, 1 = OK.																													
	Bit #7	Power Status, 1 = OK.																													
Byte #1	Bits #8...# 15	Bit assignments as with Byte #0.																													
Bytes #2...#3	Bits #16...#31	0.																													
Comments	<ul style="list-style-type: none"> The get_head_status command does not read the "normal" 20-bit status word, but rather the permanently transmitted additional 6 status bits of the SL2-100 protocol (also by the XY2-100 converter) (see page 143). The status bits are returned by get_head_status in bits#2-7 and/or bits#10-15. Independently of the scan system's current state, bits#0 and 8 are returned by get_head_status as 1, while bits#1 and 9 are returned as 0. If no scan system is currently connected or is not switched on, then the status of the most recently connected system is returned. get_startstop_info (bit #17 and/or 25) can be used for distinguishing. If a scan system is still not connected after a reset (power-on or load_program_file) (or for <code>Head = 2</code>, if the "second scan head control" option has not been enabled), then get_head_status returns the value 0. If an XY2-100 converter version 1.0 is plugged-in solitarily, or a connected scan system is not switched on, then get_head_status returns a 0xFD byte ("everything OK"). The PowerOK signal is an electronically generated signal. Therefore, it cannot be used to check whether a connected scan head is switched on at all. With an XY2-100 converter version 2.0 the value 0 is returned, if no scan system is connected or not switched on. The Power Status and Temperature Status signals deliver combined status information of both axes. In any case also obey the status signal information described in the manual of your scan system. 																														

Ctrl Command	get_head_status
Comments (cont'd)	<ul style="list-style-type: none"> When using an iDRI/E scan system (intelliSCAN, intelliSCAN_{de}, intelliDRILL, intellicube, intelliWELD, varioSCAN_{de}), <ul style="list-style-type: none"> Without the SL2-100 interface, get_head_status only returns meaningful return information if the XY2-100 status word was selected for return transmission. the Position Acknowledge signals of the X- and Y-axis are logically AND-connected and only returned as a common signal (e.g. at bit#3,4). Here, in addition, the Position Acknowledge signals of the X- and Y-axis can be separately read out by the command control_command with data = 0528_H at flag bit#12. after a reset or power-up of the scan system, it can take around 5 seconds for data to be returned from the scan system. Status signals can also be queried by get_value, get_values, set_trigger and set_trigger4. See also chapter 8.5 "Using Several Different Correction Tables", page 192 for information about using two scan heads.
RTC4→ RTC5	<p>Essentially unchanged functionality, however:</p> <ul style="list-style-type: none"> The RTC5 allows (unlike the RTC4) the simultaneous reading of both scan head connectors' status words, even (for iDRI/E scan systems) independently of the signal to be returned selected by the command control_command. Even for iDRI/E scan systems, bits #0 and 1 (unlike with the RTC4) do not return information about the scan system's operational readiness (see get_value).
References	get_value , get_values , set_trigger , set_trigger4 , get_waveform

Ctrl Command	get_hex_version
Function	Returns the version number of the currently loaded RTC5 software (DSP program file RTC5OUT.out).
Call	<code>HexVersion = get_hex_version()</code>
Result	RTC5 software version number as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The version numbers of program files are in the range 500 ... 599. get_hex_version returns the following values: <ul style="list-style-type: none"> if the 3D option is <i>not</i> enabled values in the range 2500 ... 2599 (version number + 2000) if the 3D option is enabled values in the range 3500 ... 3599 (version number + 3000) The file name extension for RTC5 program files is no longer *.hex (as with the RTC4, RTC3 and RTC2), but instead *.out (see also load_program_file). For ≥ DLL 535: The software version number can also be returned after an RTC5_VERSION_MISMATCH or RTC5_ACCESS_DENIED error. The return value is 0 if no program has yet been loaded. Here, an RTC5_TIMEOUT error is not generated.
RTC4→ RTC5	Unchanged functionality.
Version info	Last change with version DLL 535 (see comment).
References	get_dll_version , get_RTC_version

Ctrl Command	get_hi_data
Function	Returns the Home-In positions, last determined (by auto_cal) of the scan system attached to the primary scan head connector.
Call	<code>get_hi_data(&X1, &X2, &Y1, &Y2)</code>
Returned parameter values	X1, X2, Coordinates of the currently stored Home-In positions in <i>bits</i> as pointers to Y1, Y2 signed 32-bit values.
Comments	<ul style="list-style-type: none"> The command is synonymous with the get_hi_pos command with HeadNo = 1 (see comments there).
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). The returned values are in the RTC5's 20-bit range.
References	get_hi_pos , write_hi_pos

Ctrl Command	get_hi_pos
Function	Returns the Home-In positions, last determined (by auto_cal) of the scan system attached to the specified scan head connector.
Call	<code>get_hi_pos(HeadNo, &X1, &X2, &Y1, &Y2)</code>
Parameter	HeadNo Number of the scan head connector as an unsigned 32-bit value, allowed values. = 1: Primary scan head connector. = 2: Secondary scan head connector (activation required).
Returned parameter values	X1, X2, Coordinates of the currently stored (last determined) Home-In positions in <i>bits</i> as pointers to signed 32-bit values. Y1, Y2
Comments	<ul style="list-style-type: none"> For information on using this command, see section "Customer-Specific Calibration", page 227. It is up to users to ensure that the scan system currently attached to the specified scan head connector is the same scan system which was used to determine the returned Home-In positions. For determination of Home-In position values, this scan system should be equipped with an internal sensor system for automatic self-calibration (Home-In sensors). The returned values are 0 if no scan system equipped with automatic self-calibration (Home-In sensors) is attached to the specified scan head connector or if an error has occurred during determination of the Home-In values for such a system. For HeadNo=2, the returned values are also 0, if the "second scan head control" option has not been enabled. Directly after initialization (init_rtc5_dll), particularly prior to a first call of the command auto_cal(Command = 0, 1 or 3), the returned values are the Home-In reference values stored in the im EEPROM of the RTC5 Board. If such reference values have not been successfully determined at least once by auto_cal(Command = 0), get_hi_pos returns 0. The command get_hi_pos is also available without explicit access rights to a specific RTC5 Board. If parameter values are invalid, then all returned coordinates are 0 (get_last_error return code: RTC5_PARAM_ERROR).
RTC4→ RTC5	New command.
References	get_hi_data , auto_cal , set_hi , write_hi_pos



Ctrl Command	get_input_pointer
Function	Returns the present (absolute) position of the input pointer.
Call	InputPointer = get_input_pointer()
Result	position of the input pointer [0 ... (2^{20} -1)] as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The position of the input pointer corresponds to the position in RTC5 list buffer (also in the protected "List 3" area), where the next list command is stored. The number of still-available storage positions there can be queried by get_list_space. get_input_pointer returns the absolute list memory address (offset relative to the start of "List 1"). The relative position referenced to the start of the respective list area can be queried by get_list_pointer. Before loading a non-indexed subroutine or character set, you should use get_input_pointer to obtain the start address if subsequent referencing is to be performed by set_sub_pointer or set_char_pointer. The absolute position of the output pointer can be queried by get_status or get_out_pointer. The board-specific error variables <code>LastErrorHandler</code> and <code>AccErrorHandler</code> (see "Error Handling", page 98) are neither generated nor altered by get_input_pointer.
RTC4→RTC5	Unchanged functionality.
References	get_list_pointer , set_input_pointer , get_list_space , get_status , get_out_pointer

Ctrl Command	get_io_status
Function	Returns the current state of the 16-bit digital output port on the EXTENSION 1 socket connector.
Call	IOStatus = get_io_status()
Result	16-bit value (DIGITAL OUT0 ... DIGITAL OUT15) as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> This command is conceived of for use in combination with the commands set_io_cond_list and clear_io_cond_list (see also "Programming Examples", page 245). See also section "16-Bit Digital Input and Output", page 51.
RTC4→RTC5	Unchanged functionality.
References	write_io_port , write_io_port_mask , set_io_cond_list , clear_io_cond_list



Control Command	get_jump_table
Function	Retrieves the board's currently stored jump delay table and copies the 1024 corresponding unsigned 16-bit values to the supplied PC address.
Call	<code>ErrorCode = get_jump_table(Addr)</code>
Parameters	Addr PC Address of a 2048-byte area of PC main memory.
Result	Error code as unsigned 32-bit value. 0 No error. 11 Driver error.
Notes	<ul style="list-style-type: none"> Do not call this command during processing of a list. The data format is "1024 16-bit values" representing the delay values for a piecewise linear interpolation at the sampling points (=jump lengths) $N \times 1024$ with $0 \leq N < 1024$. The values can thus also be directly generated or modified by users.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 525.
Reference	set_jump_table , load_jump_table_offset

Ctrl Command	get_lap_time
Function	Returns the <i>current</i> RTC5 timer value (without resetting it to zero).
Call	<code>TimerValue = get_lap_time()</code>
Result	RTC5 timer value in seconds since the last call of save_and_restart_timer . As a 64-bit IEEE floating point value.
Comments	<ul style="list-style-type: none"> The command get_lap_time serves to query the elapsed time of a time consuming marking during processing.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 541, OUT 541.
References	get_time , save_and_restart_timer

Ctrl Command	get_laser_pin_in
Function	Returns the current status of the two digital inputs at the laser connector.
Call	<code>LaserPinIn = get_laser_pin_in()</code>
Result	Unsigned 32-bit value. Bit #0 DIGITAL IN1. (LSB) Bit #1 DIGITAL IN2. Bits #2- Reserved. 31
Comments	<ul style="list-style-type: none"> Both inputs can be used as desired by users.
RTC4→ RTC5	New command.
References	set_laser_pin_out



Ctrl Command	get_last_error
Function	Returns an error code listing any errors which occurred during execution of the most recent command.
Call	<code>LastError = get_last_error()</code>
Result	Error code as an unsigned 32-bit value. If multiple errors occurred simultaneously, then multiple bits are set. The meanings of bit numbers, error types and error constants is identical to those for the command get_error .
Comments	<ul style="list-style-type: none"> For error handling see chapter 6.8 "Error Handling", page 98. The commands <code>get_last_error</code> and <code>n_get_last_error</code> are also available without explicit access rights to a specific RTC5 Board. The board-specific error variables <code>LastError</code> and <code>AccError</code> (see "Error Handling", page 98) are neither generated nor altered by <code>get_last_error</code>.
RTC4→ RTC5	New command.
References	get_error , reset_error , set_verify

Ctrl Command	get_list_pointer				
Function	Provides the input pointer's current (relative) position and the list number.				
Call	<code>get_list_pointer(&ListNo, &Pos)</code>				
Returned parameter values	<table> <tr> <td>ListNo</td> <td>Number of the list in which the input pointer is currently located; as a pointer to an unsigned 32-bit value [1...3].</td> </tr> <tr> <td>Pos</td> <td>Current position of the input pointer (offset relative to the start of the respective list) as a pointer to an unsigned 32-bit value.</td> </tr> </table>	ListNo	Number of the list in which the input pointer is currently located; as a pointer to an unsigned 32-bit value [1...3].	Pos	Current position of the input pointer (offset relative to the start of the respective list) as a pointer to an unsigned 32-bit value.
ListNo	Number of the list in which the input pointer is currently located; as a pointer to an unsigned 32-bit value [1...3].				
Pos	Current position of the input pointer (offset relative to the start of the respective list) as a pointer to an unsigned 32-bit value.				
Comments	<ul style="list-style-type: none"> The input pointer's absolute list buffer address (offset relative to the start of "List 1") can be queried by <code>get_input_pointer</code> (see also comments there). The number of list positions until the end of the respective list (from the input pointer) can be queried by <code>get_list_space</code>. The board-specific error variables <code>LastError</code> and <code>AccError</code> (see chapter 6.8 "Error Handling", page 98) are neither generated nor altered by <code>get_list_pointer</code>. 				
RTC4→ RTC5	New command.				
References	get_input_pointer , get_list_space				



Ctrl Command	get_list_serial
Function	Returns the number of the serial-number-set most recently selected by select_serial_set_list (or of serial-number-set 0 after load_program_file) and the current serial number of this serial-number-set.
Call	<code>LastMarkedSerialNo = get_list_serial(&Set)</code>
Result	Serial number as 64-bit IEEE floating point value.
Returned parameter value	Set Number of the selected serial-number-set as a pointer to an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The serial number queried by <code>get_list_serial</code> is typically the one most recently marked by mark_serial or mark_serial_abs. For command usage, see chapter 7.5.2 "Marking Serial Numbers", page 170.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 537, OUT 537.
References	select_serial_set_list , get_serial

Ctrl Command	get_list_space
Function	Returns the amount of free list memory, hence the number of list commands that can still be loaded from the input pointer's current position to the last position in the respective list.
Call	<code>ListSpace = get_list_space()</code>
Result	number of free list positions as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> If an indexed subroutine or indexed character set is currently being loaded into the protected buffer area ("List 3"), then <code>get_list_space</code> returns the amount of still-available protected memory (otherwise the input pointer is not located in the protected area).
RTC4→ RTC5	The command <code>get_list_space</code> was available on the RTC3/RTC4 to support their circular queue mode and returned the distance between the input and output pointers. The RTC5 does not have a circular queue mode. The input pointer's position can be queried by get_input_pointer or get_list_pointer and the output pointer's position can be queried by get_status or get_out_pointer .
References	get_input_pointer , get_status , get_out_pointer , get_list_pointer



Ctrl Command	get_marking_info
Function	Returns information about any boundary exceedances during Processing-on-the-fly correction as well as improper encoder signals. The function also returns the error bits of laser-signal auto-suppression.
Call	<code>MarkingInfo = get_marking_info()</code>
Result	<p>Error code as an unsigned 32-bit value.</p> <p>Bit #0 = 1: Processing-on-the-fly underflow in X direction ($X < -524288$). (LSB)</p> <p>Bit #1 = 1: Processing-on-the-fly overflow in X direction ($X > +524287$).</p> <p>Bit #2 = 1: Processing-on-the-fly underflow in Y direction ($Y < -524288$).</p> <p>Bit #3 = 1: Processing-on-the-fly overflow in Y direction ($Y > +524287$).</p> <p>Bit #4 = 1: Processing-on-the-fly underflow in X direction ($X < X_{min}$).</p> <p>Bit #5 = 1: Processing-on-the-fly overflow in X direction ($X > X_{max}$).</p> <p>Bit #6 = 1: Processing-on-the-fly underflow in Y direction ($Y < Y_{min}$).</p> <p>Bit #7 = 1: Processing-on-the-fly overflow in Y direction ($Y > Y_{max}$).</p> <p>Bit #8 = 1: TriggerError: an enabled external trigger or simulated trigger occurred during execution of a list.</p> <p>Bit #9 = 1: ActivateFlyError: an error has occurred during activation of Processing-on-the-fly correction by activate_fly_2d or activate_fly_xy (see also chapter 8.7.7 "Synchronization of Processing-on-the-fly Applications", page 211).</p> <p>Bit #10 PosAck error bit of head A's X axis.</p> <p>Bit #11 TempOK error bit of head A's X axis.</p> <p>Bit #12 PowerOK error bit of head A's X axis.</p> <p>Bit #13 PosAck error bit of head A's Y axis.</p> <p>Bit #14 TempOK error bit of head A's Y axis.</p> <p>Bit #15 PowerOK error bit of head A's Y axis.</p> <p>Bit #16 = 1: Signal 1 of encoder input ENCODER X has too-short spacing.</p> <p>Bit #17 = 1: Signal 2 of encoder input ENCODER X has too-short spacing.</p> <p>Bit #18 = 1: Signal 1 of encoder input ENCODER Y has too-short spacing.</p> <p>Bit #19 = 1: Signal 2 of encoder input ENCODER Y has too-short spacing.</p> <p>Bit #20 = 1: Improper signal sequence at encoder input ENCODER X.</p> <p>Bit #21 = 1: Improper signal sequence at encoder input ENCODER Y.</p> <p>Bit #22 = 1: Processing-on-the-fly underflow in Z direction ($Z < -32.768$).</p> <p>Bit #23 = 1: Processing-on-the-fly overflow in Z direction ($Z > +32.767$).</p> <p>Bit #24 = 1: Processing-on-the-fly underflow in Z direction ($Z < Z_{min}$).</p> <p>Bit #25 = 1: Processing-on-the-fly overflow in Z direction ($Z > Z_{max}$).</p>



Ctrl Command	get_marking_info
Result (cont'd)	<p>Bit #26 PosAck error bit of head B's X axis.</p> <p>Bit #27 TempOK error bit of head B's X axis.</p> <p>Bit #28 PowerOK error bit of head B's X axis.</p> <p>Bit #29 PosAck error bit of head B's Y axis.</p> <p>Bit #30 TempOK error bit of head B's Y axis.</p> <p>Bit #31 PowerOK error bit of head B's Y axis.</p> <p>The remaining bits are reserved.</p>
Comments	<ul style="list-style-type: none"> For usage of this command and of the error bits#0...7, see chapter 8.7.9 "Monitoring Processing-on-the-fly Corrections", page 213. The boundary limits Xmin, Xmax, Ymin, Ymax for the customer-defined monitoring area (bits#4...7) can be specified by the command set_fly_limits. Encoder-signal spacing could be too short if interfering signals are present, a rapid directional change occurs or the frequency is essentially too high. An improper encoder signal sequence occurs if both signals 1 and 2 change simultaneously, thus hindering determination of the counting direction. If no external encoder is used, the corresponding inputs on the MARKING ON THE FLY socket connector have undefined signal levels. Therefore signal errors may be permanently detected for these inputs (upper 16 bit of the return value), even if simulated encoders are used. If this disturbs the application, then this pins should be set to a defined signal level by pull-up resistors. The error bits#10...15 and 26...31 are only set in case of an error if laser-signal auto-suppression has been activated (see section "Laser-Signal Auto-Suppression Triggered by Scan-System-Errors", page 146). Any time an error occurs, all error bits corresponding to an error-indicating status signal are (cumulatively) set. If applicable, even error bits are set, which have not been selected to be used for laser-signal auto-suppression by set_laser_control. All error bits are reset by get_marking_info. The boundary limits Zmin, Zmax for the customer-defined monitoring area (bits#24...25) can be specified by the command set_fly_limits_z. All Error bits are reset during initialization (by load_program_file). Error bits #22...23 are also reset by get_marking_info. Error bits #24...25 are not reset by get_marking_info. If applicable, error bits #24...25 get implicitly reset by the conditional commands. They can also be explicitly reset by the command clear_fly_overflow.
RTC4→ RTC5	Unchanged functionality for info bits that are also used on the RTC4.
Version info	<p>Last changes:</p> <ul style="list-style-type: none"> with version DLL 525, OUT 527 (Bit #4...7). with version DLL 527, OUT 529, RBF 519 (Bit #10...15 and 26...31). with version DLL 531, OUT 532 (Bit #22...25). with version DLL 536, OUT 536 (Bit #9).
References	set_fly_x , set_fly_y , set_fly_z , set_fly_rot , set_fly_x_pos , set_fly_y_pos , set_fly_rot_pos , set_fly_limits , set_fly_limits_z , clear_fly_overflow , if_not_activated



Ctrl Command	get_master_slave
Function	Returns the master/slave status of the addressed RTC5 Board.
Call	MasterSlaveStatus = get_master_slave()
Result	<p>Master/slave status as an unsigned 32-bit value, i.e. information, whether a further RTC5 Board is connected to the MASTER or SLAVE connector of the addressed RTC5 Board:</p> <p>Bit #0 = 1: A board is connected to the SLAVE connector. (LSB)</p> <p>Bit #1 = 1: A board is connected to the MASTER connector.</p> <p>Bit #2-31 = 0.</p> <p>and information, whether the addressed board is operated as a master, slave or single board:</p> <p>= 0 The board is operated as single board.</p> <p>= 1 The board is operated as slave (without any further downstream slave board).</p> <p>= 2 The board is operated as master.</p> <p>= 3 The board is operated as slave (together with a downstream slave board).</p>
Comments	<ul style="list-style-type: none"> For usage of this command, see chapter 6.6.3 "Master/Slave Operation", page 93. This command only returns the status if the addressed RTC5 Board was previously initialized by load_program_file. Otherwise, 0 is returned.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 515, OUT 514, RBF 512.
References	sync_slaves

Ctrl Command	get_mcbsp
Function	Returns the most recent input value that was fully transferred by the McBSP/SPI interface to the memory location for Processing-on-the-fly applications.
Call	<code>mcbsp_value = get_mcbsp()</code>
Result	Input value as a signed 32-bit value.
Comments	<ul style="list-style-type: none"> • get_mcbsp is equivalent to read_mcbsp(0) (see notes there).
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> As of OUT 526, data transfer stagnation can no longer occur at the McBSP/SPI interface. Therefore, get_mcbsp no longer needs to be called to resolve such stagnation. Up to OUT 525, no more than two complete transmissions are processed per 10 µs cycle (also at 16 MHz). Each third data word might be overwritten with further data words (the transfer stagnates). Up to OUT 525, any data transfer stagnation needs to first be resolved by an additional get_mcbsp command before the actual current values can be read by a further call of get_mcbsp. get_mcbsp_list can also achieve this within execution of a list, so that a subsequent Processing-on-the-fly-application (controlled by McBSP/SPI signals) is supplied right from the beginning with current data. As of OUT 526, data available at the McBSP/SPI interface after a load_program_file is directly and permanently acquired (see also page 56), whereas up to OUT 525 this only applies if a Processing-on-the-fly-application that uses the McBSP/SPI interface has been activated and a list is currently running.
References	read_mcbsp

Undelayed Short List Command	get_mcbsp_list
Function	Up to OUT 525: reads queued data (input value) from the buffer of the McBSP/SPI interface (Multi channel Buffered Serial Port, see also page 54) and stores it internally. As of OUT 526: no function, see "Version info".
Call	<code>get_mcbsp_list()</code>
Comments	<ul style="list-style-type: none"> • This command has no effect on the user program.
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> As of OUT 526, get_mcbsp_list is no longer needed. For compatibility with existing applications, the command continues to be available and can be executed. As of OUT 525, the data read from the McBSP/SPI interface and internally stored is available internally for Processing-on-the-fly applications that use the McBSP/SPI interface (nothing gets returned to the user program). See also version info at get_mcbsp.
References	get_mcbsp

Ctrl Command	get_out_pointer
Function	Returns the current (or most recent) position of the output pointer as offset relative to the start of the respective list and the list number.
Call	<code>get_out_pointer(&ListNo, &Pos)</code>
Returned parameter values	ListNo Number of the list ("List 1" or "List 2") in which the output pointer is currently located (or in which it most recently resided) as a pointer to an unsigned 32-bit value.
	Pos Current (or most recent) position of the output pointer (relative memory address) as a pointer to an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> This command calls get_status (see the comments there, particularly with respect to "List 3") and uses the command's returned absolute output pointer position to determine the list number and the relative position within the list. The relative position and list number returned by get_out_pointer simplify comparing the output pointer position to the current input pointer position (particularly with respect to list consistency) during an alternative list change.
RTC4→ RTC5	New command.
References	get_status, get_input_pointer, get_list_pointer

Ctrl Command	get_overrun
Function	Returns the number of overruns of the 10 µs clock period since the last call and resets the overrun counter.
Call	<code>NumberOfOverruns = get_overrun()</code>
Result	Number of overruns of the 10 µs clock period since the last call of get_overrun as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> See section "Clock Overruns", page 142.
RTC4→ RTC5	New command.

Ctrl Command	get_RTC_mode
Function	Returns the currently set operation mode of the RTC5 DLL.
Call	<code>DLLMode = get_RTC_mode()</code>
Result	DLL operation mode as a 32-bit value. = 4: RTC4 compatibility mode. = 5: RTC5 mode (default setting).
Comments	<ul style="list-style-type: none"> The DLL operation mode can be set by set_RTC4_mode or set_RTC5_mode. The default setting is RTC5 mode. The get_RTC_mode command is also available without explicit access rights to a particular RTC5 Board. get_RTC_mode is not available as a multi-board command. The board-specific error variables <code>LastError</code> and <code>AccError</code> (see "Error Handling", page 98) are neither generated nor altered by get_RTC_mode.
RTC4→ RTC5	New command.
References	set_RTC4_mode, set_RTC5_mode



Ctrl Command	get_RTC_version																
Function	Returns the version number of the RTC5 firmware (<code>RTC5RBF.rbf</code>) and additional information about enabled options of the RTC5 Board.																
Call	<code>RTCVersion = get_RTC_version()</code>																
Result	<p>RTC5 version number as an unsigned 32-bit value:</p> <table> <tr> <td>Bit #0 (LSB) ...</td> <td>Firmware version of the RTC5 Board.</td> </tr> <tr> <td>Bit #7</td> <td></td> </tr> <tr> <td>Bit #8</td> <td>= 1: The Processing-on-the-fly option is enabled (see page 199).</td> </tr> <tr> <td>Bit #9</td> <td>= 1: The “second scan head control” option is enabled (see page 43).</td> </tr> <tr> <td>Bit #10</td> <td>= 1: The 3D option is enabled (see page 193).</td> </tr> <tr> <td>Bits #11...#15</td> <td>Reserved.</td> </tr> <tr> <td>Bits #16...#23</td> <td>DSP version number.</td> </tr> <tr> <td>Bits #24...#31</td> <td>Firmware subversion of the RTC5 Board.</td> </tr> </table>	Bit #0 (LSB) ...	Firmware version of the RTC5 Board.	Bit #7		Bit #8	= 1: The Processing-on-the-fly option is enabled (see page 199).	Bit #9	= 1: The “second scan head control” option is enabled (see page 43).	Bit #10	= 1: The 3D option is enabled (see page 193).	Bits #11...#15	Reserved.	Bits #16...#23	DSP version number.	Bits #24...#31	Firmware subversion of the RTC5 Board.
Bit #0 (LSB) ...	Firmware version of the RTC5 Board.																
Bit #7																	
Bit #8	= 1: The Processing-on-the-fly option is enabled (see page 199).																
Bit #9	= 1: The “second scan head control” option is enabled (see page 43).																
Bit #10	= 1: The 3D option is enabled (see page 193).																
Bits #11...#15	Reserved.																
Bits #16...#23	DSP version number.																
Bits #24...#31	Firmware subversion of the RTC5 Board.																
Comments	<ul style="list-style-type: none"> The RTC5 firmware version numbers are in the range 500 ... 599. <code>get_RTC_version</code> (bit#0...7) returns values in the range 0 ... 99 (version number - 500). The current DSP version number is 2 (0 and 1 are older versions). The current firmware subversion is 0. For \geq DLL 535: The firmware version can also be returned after an <code>RTC5_VERSION_MISMATCH</code> or <code>RTC5_ACCESS_DENIED</code> error. The return value is 0 if no program has yet been loaded. Here, an <code>RTC5_TIMEOUT</code> error is not generated. 																
RTC4→RTC5	Unchanged functionality.																
Version info	Change with version DLL 518, OUT 517: bits#16...23. Last change with version DLL 535: bits#24...31 (see also comment).																
References	get_hex_version , get_dll_version																

Ctrl Command	get_serial
Function	Returns the current serial number of the serial-number-set selected by select_serial_set (or of serial-number-set 0 after load_program_file).
Call	<code>CurrentSerialNo = get_serial()</code>
Result	Serial number (as 64-bit IEEE floating point value).
Comments	<ul style="list-style-type: none"> For command usage, see chapter 7.5.2 “Marking Serial Numbers”, page 170. This command should not be confused with get_serial_number, which returns the product serial number of the RTC5 Board.
RTC4→RTC5	New command.
References	select_serial_set



Ctrl Command	get_serial_number
Function	Returns the individual serial number of the active RTC5 Board.
Call	<code>RTCSerialNumber = get_serial_number()</code>
Result	RTC5 serial number as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> Serial numbers of installed boards are ascertained by init_RTC5_dll and cached in the DLL, from where you can query them by get_serial_number. The command get_serial_number is helpful when using several RTC5 Boards in one computer (see chapter 6.6, page 92). The associated multi-board command n_get_serial_number can be used for determining the relationship between the installed boards and the DLL-internal numbers assigned to them during initialization. The DLL-internal numbers are newly assigned during each initialization of a user program (see init_RTC5_dll) and must be supplied for a variety of commands (in particular, all multi-board commands). The number of boards found during initialization can be queried by rtc5_count_cards. The get_serial_number and n_get_serial_number commands are also available without explicit access rights to a specific RTC5 Board. The board-specific error variables <code>LastError</code> and <code>AccError</code> (see chapter 6.8 "Error Handling", page 98) are neither generated nor altered by get_serial_number.
RTC4→RTC5	Unchanged functionality.
References	rtc5_count_cards

Ctrl Command	get_standby	
Function	Returns the currently set standby parameters.	
Call	<code>get_standby(&HalfPeriod, &PulseLength)</code>	
Returned parameter values	HalfPeriod	Half of the currently set standby output period of the standby pulses as a pointer to an unsigned 32-bit value. <i>1 bit equals 1/64 µs.</i>
	PulseLength	Currently set pulse length of the standby pulses as a pointer to an unsigned 32-bit value. <i>1 bit equals 1/64 µs.</i>
Comments	<ul style="list-style-type: none"> For usage of this command, see "Laser Standby" Signals, Seite 145. 	
RTC4→RTC5	New command. In RTC4 compatibility mode, the RTC5 divides the parameter values by 8. <i>1 bit equals 1/8 µs.</i>	
Version info	Available as of version DLL 515, OUT 514, RBF 512.	
References	set_standby	



Ctrl Command	get_startstop_info
Function	Provides information about internal and external list starts and stops since the last time the command was called. Also provided are the current external start and stop levels, the status and signal level of the laser control signals, and possible transmission errors to and from the attached scan system.
Call	StartStopInfo = get_startstop_info()
Result	<p>Info signal as an unsigned 32-bit value:</p> <p>Bit #0 = 1: An <i>internal</i> list start has been executed (by execute_list or similar) since the last get_startstop_info command was called.</p> <p>Bit #1 = 1: An <i>external</i> list start has been executed (by /START, /START2, /Slave-START, simulate_ext_start or simulate_ext_start_ctrl) since the last get_startstop_info command was called.</p> <p>Bit #2 = 1: An <i>internal</i> list stop has been executed (by stop_execution) since the last get_startstop_info command was called.</p> <p>Bit #3 = 1: An <i>external</i> list stop has been executed (by /STOP, /STOP2, /Slave-STOP or simulate_ext_stop) since the last get_startstop_info command was called.</p> <p>Bit #4 Ext-Stop level (logical AND operation of the signals /STOP, /STOP2, /Slave-STOP and simulate_ext_stop): = 1: No stop signals are currently present at the inputs or the inputs are not connected. = 0: There is a stop signal at at least one input.</p> <p>Bits #5... 8 Reserved.</p> <p>Bit #9 = 1: The laser control signals for "laser active" operation are enabled (see also set_laser_control, enable_laser and disable_laser).</p> <p>Bit #10 = 1: The TTL laser control signals at the LASER1 and LASER2 output ports are <i>active-low</i> (the signal level can be defined by set_laser_control).</p> <p>Bit #11 = 1: Since the last call of get_startstop_info, at least one external list start has failed (more external starts were triggered than could be simultaneously held in the 8-start wait loop).</p> <p>Bit #12 Ext-Start level (logical AND operation of the signals /START, /START2 and /Slave-START): = 1: No start signals are currently present at the inputs or the inputs are not connected. = 0: A start signal is present at at least one input.</p> <p>Bit #13 = 1: The TTL laser control signal at the LASERON output port is <i>active-low</i> (the signal level can be defined by set_laser_control).</p> <p>Bits #14 ...15 Reserved.</p>



Ctrl Command	get_startstop_info
	<p>Bits #16 ... 31 Error bits that get set when an error occurs during data transmission from the scan system (bits #16...23 for the primary scan head connector, bits #24...31 for the secondary scan head connector):</p> <ul style="list-style-type: none"> Bits #16,24: Incorrect number of frames within a data block. Bits #17,25: Incorrect pulse length of signal received from scan system, maybe no scan system is connected. Bits #18,26: Preamble sequence incorrect. Bits #19,27: Bit count within a subframe incorrect. Bits #20,28: Parity error when reading data received from scan system. Bits #21,29: The present data is invalid (old). Bits #22,30: Reserved. Bits #23,31: Reserved.
Comments	<ul style="list-style-type: none"> • The info bits #0 ... #3 and the error bits #16 ... #31 are reset after the command is executed. • See also "External List Stop", page 239 and "External List Start", page 240.
RTC4→ RTC5	Unchanged functionality for the info bits that are also used on the RTC4.
References	get_counts , get_status , set_control_mode



Ctrl Command	get_status													
Function	Returns the current list execution status (BUSY, INTERNAL-BUSY and PAUSED) and the current (or most recent) position of the output pointer.													
Call	get_status(&Status, &Pos)													
Returned parameter values	Status	<p>Status value as a pointer to an unsigned 32-bit value.</p> <table> <tr> <td>Bit #0 (LSB)</td><td>= 1: BUSY status set.</td></tr> <tr> <td>Bit #1 ... Bit #6</td><td>Reserved.</td></tr> <tr> <td>Bit #7</td><td>= 1: INTERNAL-BUSY status set.</td></tr> <tr> <td>Bit #8 ... Bit #14</td><td>Reserved.</td></tr> <tr> <td>Bit #15</td><td>= 1: PAUSED status set.</td></tr> <tr> <td>Bit #16 ... Bit #31</td><td>0.</td></tr> </table>	Bit #0 (LSB)	= 1: BUSY status set.	Bit #1 ... Bit #6	Reserved.	Bit #7	= 1: INTERNAL-BUSY status set.	Bit #8 ... Bit #14	Reserved.	Bit #15	= 1: PAUSED status set.	Bit #16 ... Bit #31	0.
Bit #0 (LSB)	= 1: BUSY status set.													
Bit #1 ... Bit #6	Reserved.													
Bit #7	= 1: INTERNAL-BUSY status set.													
Bit #8 ... Bit #14	Reserved.													
Bit #15	= 1: PAUSED status set.													
Bit #16 ... Bit #31	0.													
Pos	Current (or most recent) position of the output pointer (absolute memory address) as a pointer to an unsigned 32-bit value.													
Comments	<ul style="list-style-type: none"> For a description of when the BUSY, INTERNAL-BUSY or PAUSED status values are set or not set, see chapter 6.4.3 "List Execution Status", page 77. (BUSY and PAUSED set) requires restart_list for continuation, (BUSY not set and PAUSED set) requires release_wait and (both BUSY and PAUSED not set) requires execute_list_pos. "Continuation" is not allowed with (BUSY set and PAUSED not set) and a currently running list. An improper continuation generates the get_last_error return code RTC5_BUSY. With (INTERNAL-BUSY set), release_wait and execute_list_pos are only executed with a delay (after INTERNAL-BUSY has been reset again). The output pointer points to the command (in "List 1" or "List 2") currently being executed or most recently executed. If, during processing of a subroutine in the protected buffer area ("List 3"), the output pointer's position Pos is queried, then the position is returned of the list command in the list area ("List 1" or "List 2") in which the output pointer most recently resided (typically from where the subroutine was called (e.g. with list_call). The commands pause_list and set_wait leave Pos unchanged. get_status returns the output pointer's position as an absolute memory address (offset relative to the start of "List 1"). The relative position referenced to the start of the respective list area can be queried by get_out_pointer. The current input pointer position can be queried by get_input_pointer or get_list_pointer. List status values for individual lists (see chapter 6.4.2 "List Status", page 76) can be queried by read_status. The commands get_status, get_input_pointer and read_status can be used during loading of a list to ensure that no list is overwritten that has still not been processed (see also the load_list command). As long as no program has been loaded (by load_program_file), get_status returns undefined values. The command get_head_status is available for querying the status signals of the scan heads. 													



Ctrl Command	get_status
RTC4→ RTC5	Essentially unchanged functionality, however: The Parameter <code>Status</code> returns the BUSY, INTERNAL-BUSY and PAUSED status with an RTC5, whereas only the BUSY status with an RTC4.
References	read_status , get_input_pointer , get_list_pointer , get_out_pointer

Ctrl Command	get stepper_status																				
Function	Returns the following status information for both stepper motor outputs: the current statuses of the stepper motor signals, the currently defined CLOCK pulse period, the Busy and Init statuses, and the current values of the internal position variables.																				
Call	<code>get stepper_status(&Status1, &Pos1, &Status2, &Pos2)</code>																				
Returned parameter values	<p>Status1, Current statuses of stepper motor outputs 1 and 2 as pointers to unsigned Status2 32-bit values.</p> <table> <tr> <td>Bit #0</td> <td>ENABLE signal. (LSB)</td> </tr> <tr> <td>Bit #1</td> <td>DIRECTION signal.</td> </tr> <tr> <td>Bit #2</td> <td>CLOCK signal.</td> </tr> <tr> <td>Bit #3</td> <td>SWITCH signal.</td> </tr> <tr> <td>Bit #4</td> <td>Busy status.</td> </tr> <tr> <td>Bit #5</td> <td>Init status.</td> </tr> <tr> <td>Bits #6...</td> <td>Reserved.</td> </tr> <tr> <td>7</td> <td></td> </tr> <tr> <td>Bits #8...</td> <td>CLOCK pulse period (24-bit value).</td> </tr> <tr> <td>31</td> <td></td> </tr> </table> <p>Pos1, Current values of the internal position variables for stepper motor outputs 1 Pos2 and 2 as pointers to signed 32-bit values.</p>	Bit #0	ENABLE signal. (LSB)	Bit #1	DIRECTION signal.	Bit #2	CLOCK signal.	Bit #3	SWITCH signal.	Bit #4	Busy status.	Bit #5	Init status.	Bits #6...	Reserved.	7		Bits #8...	CLOCK pulse period (24-bit value).	31	
Bit #0	ENABLE signal. (LSB)																				
Bit #1	DIRECTION signal.																				
Bit #2	CLOCK signal.																				
Bit #3	SWITCH signal.																				
Bit #4	Busy status.																				
Bit #5	Init status.																				
Bits #6...	Reserved.																				
7																					
Bits #8...	CLOCK pulse period (24-bit value).																				
31																					
Comments	<ul style="list-style-type: none"> For programming the stepper motor signals, see chapter 9.1.5 "Stepper Motor Control", page 234. If the SWITCH signal (limit switch signal) is set to (LOW), no more CLOCK pulses are generated. The Busy status indicates that a previously initiated (by <code>stepper_abs</code>, <code>stepper_rel</code>, etc.) set-position movement has not yet completed. The Init status indicates that a previously initiated (by <code>stepper_init</code>) reference movement has not yet completed. 																				
RTC4→ RTC5	New command.																				
Version info	<ul style="list-style-type: none"> Available as of version DLL 527, OUT 529, RBF 519. For older RTC5 Boards with DSP version numbers < 2 (<code>get rtc_version</code> bits #16-23), the command is not executed (<code>get last error</code> return code: <code>RTC5_TYPE_REJECTED</code>). 																				



Ctrl Command	get_sub_pointer
Function	Returns the absolute start address of an indexed subroutine.
Call	SubPointer = get_sub_pointer(Index)
Parameter	Index Index of the indexed subroutine as an unsigned 32-bit value. Allowed range: [0 ... 1023].
Result	Absolute start address as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The get_sub_pointer command reads from the internal management table the start address of the indexed subroutine with the specified index. Whether the read address resides in a protected or the unprotected memory area depends on whether the subroutine was loaded into the protected memory area or an unprotected subroutine was only subsequently referenced. If <code>Index > 1023</code> or if no subroutine was referenced with the specified index, then the command returns the value “-1” (e.g. $2^{32}-1$). This command is useful for checking if a subroutine has already been defined or for calling an indexed subroutine by an absolute memory address as if it were a non-indexed subroutine. Be aware, though, that a subsequent save_disk/load_disk might alter the absolute memory address.
RTC4→ RTC5	New command.
References	get_char_pointer, get_text_table_pointer



Ctrl Command	get_sync_status
Function	Returns the master/slave synchronization status of the addressed RTC5 Board.
Call	<code>MasterSlaveSyncStatus = get_sync_status()</code>
Result	<p>Master/slave synchronization status [0 ... 640] as an unsigned 32-bit value.</p> <p>< 11: The board is synchronized to the master board (or to the preceding board in the master/slave chain).</p> <p>= 640: The addressed board is operated as master.</p>
Comments	<ul style="list-style-type: none"> For usage of this command, see chapter 6.6.3 "Master/Slave Operation", page 93. If the addressed board has not been initialized previously by load_program_file, the get_sync_status returns 0. First synchronize the boards of the master/slave chain by the sync_slaves command before using get_sync_status. To ensure that get_sync_status actually returns the current master/slave synchronization status, you should first have already triggered an external list start for the master board by an external start signal or a command (see section "External List Start", page 240). This list start then initiates measurement of the time differences between each board's /SLAVE start pulse and the corresponding 10 µs clock. This time difference gets stored on each board as the master/slave synchronization status (in units of 15 ns) and remains stored there until the a new external list start is triggered for the master board. This gives you the flexibility to query the synchronization status by get_sync_status even at a later point in time. In the synchronized state, measured time differences are shorter than the transit time difference for synchronization between the boards themselves (see page 93): master/slave synchronization status < 11. The master/slave synchronization status is typically 7. In a not-explicitly-synchronized state (prior to sync_slaves), the master/slave synchronization status might coincidentally be < 11. If so, then the boards behave as if synchronized.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 531, OUT 532, RBF 523.
References	sync_slaves , get_master_slave



Ctrl Command	get_table_para
Function	Returns the value of the specified parameter from a currently loaded correction table.
Call	<code>TablePara = get_table_para(TableNo, ParaNo)</code>
Parameters	<p>TableNo Number of the currently loaded correction table as an unsigned 32-bit value. Allowed values: [1...4].</p> <p>ParaNo Number of the parameter as an unsigned 32-bit value. Allowed values: 0-15 (meaning: see section "Correction File Header", page 140).</p>
Result	Parameter value as a 64-bit IEEE floating point value (see section "Correction File Header", page 140).
Comments	<ul style="list-style-type: none"> The parameter values can be read out by get_table_para from a currently loaded correction table and by get_head_para from an assigned correction table and thus directly incorporated into a user program (see section "Correction File Header", page 140). If the parameters <code>TableNo</code> and <code>ParaNo</code> are out of range, then the return value is 0 (get_last_error return code: <code>RTC5_PARAM_ERROR</code>). If no correction table with the specified number has been loaded, then the parameter's value is undefined.
RTC4→ RTC5	New command.
Version info	Last change (with version DLL 533, OUT 534): the <code>TableNo</code> parameter's value range has been increased to [0...4] (load_correction_file lets you load up to 4 correction files).



Ctrl Command	get_text_table_pointer
Function	Returns the absolute start address of an indexed text string.
Call	TextTablePointer = get_text_table_pointer(Index)
Parameter	Index Index of the indexed text string as an unsigned 32-bit value. Allowed range: [0 ... 41].
Result	Absolute start address as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The get_text_table_pointer command reads from the internal management table the start address of the indexed text string with the specified index. Whether the read address resides in a protected or the unprotected memory area depends on whether the text string was loaded into the protected memory area or an unprotected subroutine was only subsequently referenced. If <code>Index > 41</code> or if no text string was referenced with the specified index, then the command returns the value “-1” (e.g. $2^{32}-1$). This command is useful for checking if a text string has already been defined or for calling an indexed text string by an absolute memory address as if it were a non-indexed subroutine, e.g. for conditional execution with list_call_cond. Be aware, though, that a subsequent save_disk/load_disk might alter the absolute memory address. And you should ensure that get_text_table_pointer does not return “-1”; otherwise the list_call_cond command is ignored.
RTC4→ RTC5	New command.
References	get_char_pointer, get_sub_pointer

Ctrl Command	get_time
Function	Returns the RTC5 timer value (without resetting it to zero). It was stored during the most recent call of save_and_restart_timer .
Call	TimerValue = get_time()
Result	RTC5 timer value in seconds. As a 64-bit IEEE floating point value.
Comments	<ul style="list-style-type: none"> See save_and_restart_timer. The number of elapsed list-command clock cycles since the reset to zero can be queried by get_lap_time.
RTC4→ RTC5	Unchanged.
References	get_lap_time, save_and_restart_timer

Ctrl Command	get_transform																					
Function	Transfers to the PC the position values that were recorded by set_trigger and stored on the RTC5, and applies backward transformation to these values.																					
Call	<code>get_transform(Number, Ptr1, Ptr2, Ptr, Code)</code>																					
Parameters	<p>Number Number [1...2²⁰] of to-be-backward-transformed position values as an unsigned 32-bit value. The measured values with indices 0 to (Number-1) are backward transformed.</p> <p>Ptr1, Pointers (in C and C++ data type ULONG_PTR, i.e. unsigned 32-bit or 64-bit values) to the two areas of PC main memory to which the backward transformed values should be transferred (see also Code).</p> <p>Ptr Pointer (in C and C++ data type ULONG_PTR, i.e. an unsigned 32-bit or 64-bit value) to the area of PC main memory to which the correction and transformation settings for backward transformation were previously transferred by upload_transform.</p> <p>Code This parameter (an unsigned 32-bit value) controls aspects of backward transformation, particularly which partial transformations should be performed: If a partial transformation should <i>not</i> be performed, then its corresponding bit (#2...#5) must be set to 1. This parameter has the same meaning as in transform (Ptr1 corresponds to Sig1 and Ptr2 to Sig2). For bit #0 = 0, the measurement value pairs recorded by set_trigger are backward transformed as XY coordinates:</p> <table> <tr> <td>Bit #1</td> <td>= 0:</td> <td>Values recorded by measurement channel 1 (Signal1) are transferred to the PC using Ptr1 and then backward transformed as X coordinates. Values recorded by measurement channel 2 (Signal2) are transferred to the PC using Ptr2 and then backward transformed as Y coordinates.</td> </tr> <tr> <td></td> <td>= 1:</td> <td>Values recorded by measurement channel 1 (Signal1) are transferred to the PC using Ptr1 and then backward transformed as Y coordinates. Values recorded by measurement channel 2 (Signal2) are transferred to the PC using Ptr2 and then backward transformed as X coordinates.</td> </tr> <tr> <td>Bit #2</td> <td>= 0:</td> <td>Gain/offset correction of automatic self-calibration is backward transformed.</td> </tr> <tr> <td>Bit #3</td> <td>= 0:</td> <td>The image field correction is backward transformed.</td> </tr> <tr> <td>Bit #4</td> <td>= 0:</td> <td>The offset of the defined coordinate transformation is backward transformed.</td> </tr> <tr> <td>Bit #5</td> <td>= 0:</td> <td>The total matrix of the defined coordinate transformation is backward transformed.</td> </tr> <tr> <td>Bits #6...#31</td> <td></td> <td>Reserved.</td> </tr> </table>	Bit #1	= 0:	Values recorded by measurement channel 1 (Signal1) are transferred to the PC using Ptr1 and then backward transformed as X coordinates. Values recorded by measurement channel 2 (Signal2) are transferred to the PC using Ptr2 and then backward transformed as Y coordinates.		= 1:	Values recorded by measurement channel 1 (Signal1) are transferred to the PC using Ptr1 and then backward transformed as Y coordinates. Values recorded by measurement channel 2 (Signal2) are transferred to the PC using Ptr2 and then backward transformed as X coordinates.	Bit #2	= 0:	Gain/offset correction of automatic self-calibration is backward transformed.	Bit #3	= 0:	The image field correction is backward transformed.	Bit #4	= 0:	The offset of the defined coordinate transformation is backward transformed.	Bit #5	= 0:	The total matrix of the defined coordinate transformation is backward transformed.	Bits #6...#31		Reserved.
Bit #1	= 0:	Values recorded by measurement channel 1 (Signal1) are transferred to the PC using Ptr1 and then backward transformed as X coordinates. Values recorded by measurement channel 2 (Signal2) are transferred to the PC using Ptr2 and then backward transformed as Y coordinates.																				
	= 1:	Values recorded by measurement channel 1 (Signal1) are transferred to the PC using Ptr1 and then backward transformed as Y coordinates. Values recorded by measurement channel 2 (Signal2) are transferred to the PC using Ptr2 and then backward transformed as X coordinates.																				
Bit #2	= 0:	Gain/offset correction of automatic self-calibration is backward transformed.																				
Bit #3	= 0:	The image field correction is backward transformed.																				
Bit #4	= 0:	The offset of the defined coordinate transformation is backward transformed.																				
Bit #5	= 0:	The total matrix of the defined coordinate transformation is backward transformed.																				
Bits #6...#31		Reserved.																				



Ctrl Command	get_transform
Parameters (cont'd)	<p>Code If bit #0 = 1, then (only) the values recorded with set_trigger by one of the two measurement channels (either channel 1 or 2) are backward transformed as Z coordinates:</p> <p>Bit #1 = 0: Values recorded by measurement channel 1 (Signal1) are transferred to the PC using <code>Ptr1</code> and then backward transformed as Z coordinates. Values recorded by measurement channel 2 (Signal2) are transferred untransformed to the PC using <code>Ptr2</code>.</p> <p> = 1: Values recorded by measurement channel 2 (Signal2) are transferred to the PC using <code>Ptr1</code> and then backward transformed as Z coordinates. Values recorded by measurement channel 1 (Signal1) are transferred untransformed to the PC using <code>Ptr2</code>.</p> <p>Bit #2 = 0: The offset to the focal length defined by set_defocus or set_defocus_list is backward transformed.</p> <p>Bit #3 = 0: The ABC correction is backward transformed.</p> <p>Bit #4 = 0: The offset to the Z coordinate defined by set_offset_xyz or set_offset_xyz_list is backward transformed.</p> <p>Bits #5... #31 Reserved.</p>
Comments	<ul style="list-style-type: none"> For backward transformation of position values see chapter 8.1.3 "Position Monitoring", page 174. get_transform(Number, <code>Ptr1</code>, <code>Ptr2</code>, <code>Ptr</code>, <code>Code</code>) transfers to the PC the data pairs recorded by set_trigger by executing get_waveform(1, Number, <code>Ptr1</code>) and get_waveform(2, Number, <code>Ptr2</code>) and overwrites the data pairwise by using transform(<code>Sig1</code>, <code>Sig2</code>, <code>Ptr</code>, <code>Code</code>). Prior to calling get_transform, you must call upload_transform. Additionally, position values should have been recorded by set_trigger. Before calling get_transform (as with get_waveform), you can check by measurement_status whether a measurement session is currently running that was started with set_trigger. We recommend not to read any data when data recording is currently active. The number <code>Pos</code> for the last (or current) data pair of the measurement session can be queried by measurement_status. No more than <code>Pos+1</code> data elements should be read. Any further elements are from earlier recordings or the initialization. The PC main memory areas pointed to by <code>Ptr1</code> and <code>Ptr2</code> must have been sufficiently allocated by users (<code>Number</code> × 4 bytes per channel). If only Z position values are to be backward transformed (<code>Code</code> bit #0 = 1), then you can set the pointer (<code>Ptr1</code> or <code>Ptr2</code>) for the unused return channel to NULL. This channel does then not transfer and backward transform data to the PC. Ensure here that <code>Code</code> bit #1 is appropriately set.



Ctrl Command	get_transform
Comments (Forts.)	<ul style="list-style-type: none"> If the command call is made with <code>Ptr1 = NULL and Ptr2 = NULL</code>, then neither channel transfers data to the PC and likewise no backward transformations are performed (<code>get_last_error</code> return code <code>RTC5_PARAM_ERROR</code>). The same applies for <code>Number = 0</code> or <code>Number > 2²⁰</code>. If <code>Ptr = NULL</code>, then no backward transformation is performed. The data recorded by <code>set_trigger</code> is then transferred untransformed to the PC, as with <code>get_waveform(1, Number, Ptr1)</code> and <code>get_waveform(2, Number, Ptr2)</code>. The same applies for <code>Ptr ≠ NULL</code> if an error occurred during execution of <code>get_transform</code> (e.g. when data referenced by <code>Ptr</code> are invalid or erroneous or when Z-axis inversion is not possible). Here, however, a <code>get_last_error</code> return code of <code>RTC5_PARAM_ERROR</code> is additionally generated. If needed, the values recorded with <code>set_trigger</code> and backward-transformed transferred to the PC by <code>get_transform</code> can additionally be untransformed transferred to the PC by <code>get_waveform</code>. If backward transformation of Z position values is requested (Code bit #0 = 1), but only a 2D correction table was assigned at the timepoint of the prior successful call to <code>upload_transform</code>, then the offsets to the focal length and Z coordinates are initialized with 0 and the values A, B, C with are initialized 0, 1, 0 (1-to-1 backward transformation). For backward transformation of XY position values (Code bit #0 = 0), only the Z = 0 plane are transformed. XY stretching and Z defocus due to Z deviations (particularly with non-F-Theta systems) are not taken into account. PCI transmission errors generate the <code>get_last_error</code> return code <code>RTC5_SEND_ERROR</code>.
RTC4 → RTC5	<p>New command.</p> <p>Even in the RTC5's RTC4 compatibility mode, all values transferred to the PC (including Z values) are in the 20-bit range and must, when necessary, be converted (by dividing by 16) by users.</p>
Version info	<ul style="list-style-type: none"> Available as of version DLL 516, OUT 515, RBF 512. Change with version DLL 517: parameter Code changed.
References	<code>upload_transform</code> , <code>transform</code> , <code>set_trigger</code> , <code>get_waveform</code>



Ctrl Command	get_value
Function	Returns the current value of the specified signal.
Call	<code>Value = get_value(Signal)</code>
Parameter	Signal desired signal type as an unsigned 32-bit value.
Result	Current value of the specified signal as a signed 32-bit value.
Comments	<ul style="list-style-type: none"> The selectable signal types are identical to those of the set_trigger command (refer to the comments there for the allowed value range, signal types and other information). If the value for <code>Signal</code> is unallowed, then get_value does not read out a signal and returns 0 (get_last_error return code <code>RTC5_PARAM_ERROR</code>). To observe the specified signal over a long time period, use set_trigger to start a corresponding measurement session. When using an <i>iDRI/VE</i> scan system (<i>intelliSCAN</i>, <i>intelliSCAN_{de}</i>, <i>intelliDRILL</i>, <i>intellicube</i>, <i>intelliWELD</i>, <i>varioSCAN_{de}</i>), after a reset or power-up of the scan system, it can take around 5 seconds before valid data starts being returned from the scan system. If the XY2-100 status word was selected for return transmission, then the operational readiness can be checked by monitoring bits 4 and 5: bit#4 = 1 and bit#5 = 0 (only) if returned-data transmission is valid. For <code>Signal = 0</code>, get_value returns the current laser status (<code>LASERON</code> signal) even when list execution has already been finished. When you query data returned as status signals from the scan system to the RTC5 (<code>Status<AX...BY></code>), then be mindful of the returned data type's value range when evaluating it (see the command description of control_command): <ul style="list-style-type: none"> Data types originally generated in the scan system as unsigned 16-bit values (e.g. the XY2-100 status word or the serial number) and returned to the RTC5 as unsigned 20-bit values (whereby bits #0...3 = 0) contain the relevant information in bits #4...19. Here, only bits #4...19 should be evaluated (see code example below). For bits #20...31 of this data type, get_value returns to the PC not only zero, but sometimes (depending on bit #19 of the underlying 16-bit status value) even the value one. So only evaluate bits #4...19. In contrast, data types returned to the RTC5 as signed 20-bit values (e.g. actual positions or actual speeds) can be safely evaluated as a complete signed 32-bit value returned by get_value (see also code example below). Although Z values must be supplied as a 16 bit values in 3D command parameters, SCANLAB Z axes and 3-axis scan systems (and correspondingly the get_value command) return Z values always as signed 20-bit values. If necessary, these returned Z values must be converted (divided by 16) by the user program.



Ctrl Command	get_value
Example (C/C++)	<p>Querying diverse data types (primary scan head connector, X axis):</p> <p>a) XY2-100 status word, PowerOK status</p> <pre>UINT statusword, powerOK; control_command (1, 1, 0x0500); // only applicable for iDRIVE systems statusword = (get_value(1) & 0x000FFFF0) >> 4; powerOK = (statusword & 0x00000080);</pre> <p>b) Serial number (only applicable for iDRIVE systems)</p> <pre>UINT SN_low, SN_high, SN; // the serial number's lower 16 bits are selected for return // and queried by get_value: control_command (1, 1, 0x051E); SN_low = (get_value(1) & 0x000FFFF0)>>4; // the serial number's upper 16 bits are selected for return // and queried by get_value: control_command (1, 1, 0x051F); SN_high = (get_value(1) & 0x000FFFF0)>>4; //Complete serial number: SN = (SN_high << 16) + SN_low;</pre> <p>c) Actual position (only applicable for iDRIVE systems)</p> <pre>long real_position; control_command (1, 1, 0x0501); real_position = get_value(1);</pre>
RTC4→ RTC5	Essentially unchanged functionality, however: Even in RTC4 compatibility mode, all returned values are in the RTC5's 20-bit range, but get transferred to the PC as 32-bit data. Therefore, you must evaluate the values accordingly (see above).
Version info	Last change (with version DLL 536, OUT 536): Additional selectable data signals (Signal = 43 and 44, see set_trigger).
References	get_values , set_trigger , get_waveform , get_head_status



Ctrl Command	get_values
Function	Returns the current values of up to four specified signals.
Call	<code>get_values(SignalPtr, ResultPtr)</code>
Parameters	<p>SignalPtr Pointer (in C and C++ data type <code>ULONG_PTR</code>, i.e. an unsigned 32-bit or 64-bit value) to an array of four unsigned 32-bit values, where the desired signal types are specified.</p> <p>ResultPtr Pointer (in C and C++ data type <code>ULONG_PTR</code>, i.e. an unsigned 32-bit or 64-bit value) to an array of four signed 32-bit values, where the current values of the up to four specified signals should be stored.</p>
Comments	<ul style="list-style-type: none"> Up to four desired signals can be simultaneously queried. The selectable signal types are identical to those of the <code>set_trigger</code> command (refer to the comments there for the allowed value range, signal types and other information). The desired signal types must be specified by <code>SignalPtr</code>. The corresponding signal values are then stored by <code>ResultPtr</code>. For storage of each queried data set, the user program must make available (at the address specified by <code>ResultPtr</code>) 4×4 bytes of PC memory. <code>get_values</code> functions similarly to the <code>get_value</code> command (see comments there). <code>get_values</code> returns 0 and performs no query on channels for which an invalid signal type was specified by <code>SignalPtr</code>. Nevertheless, the <code>get_last_error</code> return code <code>RTC5_PARAM_ERROR</code> is only generated if all four specified signal types are invalid. If any of the pointer parameters are NULL, then the command is not executed (all return values are 0) and a <code>get_last_error</code> return code of <code>RTC5_PARAM_ERROR</code> is generated.
RTC4→RTC5	New command. Even in RTC4 compatibility mode, the four returned values are in the RTC5's 20-bit range, but get transferred to the PC as 32-bit data. Therefore, you must evaluate the values accordingly (see comments for <code>get_value</code>).
Version info	<ul style="list-style-type: none"> Available as of version DLL 516, OUT 515, RBF 512. Last change (with version DLL 536, OUT 536): Additional selectable data signals (see <code>set_trigger</code>).
References	get_value , set_trigger , transform



Ctrl Command	get_wait_status
Function	Returns the wait state of the RTC5.
Call	WaitStatus = get_wait_status()
Result	Wait state as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none">• If processing has stopped at a wait marker, the command get_wait_status returns the number of this marker. See set_wait.• If no wait marker was reached, the command get_wait_status returns zero.• Processing of the list can be resumed by calling the command release_wait.
RTC4→ RTC5	Unchanged functionality.
References	set_wait, release_wait



Ctrl Command	get_waveform
Function	Transfers to the PC the data that was measured and stored onto the RTC5 by set_trigger or set_trigger4 .
Call	<code>get_waveform(Channel, Number, Ptr)</code>
Parameters	<p>Channel Measurement channel [1 or 2; if recordings started by set_trigger4, then also 3 or 4]; specified as an unsigned 32-bit value.</p> <p>Number Number [1...2^{20} for set_trigger, 1...2^{19} for set_trigger4] of measured values to transfer; specified as an unsigned 32-bit value. Values of measurement positions 0 to (Number-1) are transferred.</p> <p>Ptr Pointer (data type <code>ULONG_PTR</code> in C and C++, that is, as an unsigned 32-bit or 64-bit value) to a location in the PC memory to where the measured values should be transferred.</p>
Comments	<ul style="list-style-type: none"> In the following cases, no data is transferred (get_last_error return code <code>RTC5_PARAM_ERROR</code>): <ul style="list-style-type: none"> For Number = 0. For Number > 2^{20}. For Number > 2^{19} when Channel = 3 or 4 was selected or if a file was loaded as correction table No=3 or No=4 with load_correction_file. For Ptr = NULL. PCI transmission errors generate the get_last_error return code <code>RTC5_SEND_ERROR</code>. Before calling get_waveform, you can check by measurement_status whether a measurement session is currently running that was started with set_trigger or set_trigger4. We recommend not reading any data when data recording is currently active. The number <code>Pos</code> for the last (or current) data pair of the measurement session can be queried by measurement_status. No more than <code>Pos+1</code> data elements should be read. Any further elements are from earlier recordings or the initialization.
RTC4→RTC5	Essentially unchanged functionality, however: Even in RTC4 compatibility mode, all values are in the RTC5's 20-bit range, but get transferred to the PC as 32-bit data. Therefore, you must evaluate the values accordingly (see comments for get_value).
Version info	Last change with version DLL 536, OUT 536: Channel = 3 and 4.
References	set_trigger , set_trigger4 , get_value , get_values



Ctrl Command	get_z_distance
Function	Returns the focus length value l for the specified point within the working volume.
Restriction	If the 3D option has not been enabled or if no 3D correction table has been assigned (see select_cor_table), then the command returns 0 and otherwise has no effect.
Call	<code>ZDistance = get_z_distance(X, Y, Z)</code>
Parameters	X, Y, Z Absolute coordinates of the point $(x y z)$ in the working volume in bits as signed 32-bit values. Allowed range: <ul style="list-style-type: none">• For X and Y: [-524288 ... 524287].• For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped.
Result	Focus length value as a signed 32-bit value [-32768 ... 32767].
Comments	<ul style="list-style-type: none"> • This command is only needed for re-calibrating the Z-axis in a 3-axis scan system (see page 196). The (unit-free) focus length value l corresponds to the focus length difference between the specified point $(x y z)$ and the point $(0 0 0)$. The focus length value can be positive or negative. • The command first performs a (virtual) jump to the point $(x y z)$ and then returns the focus length value. • If a list is currently executed, then the command has no effect and returns 0 (get_last_error return code: RTC5_BUSY). • The command has no effect and returns 0 (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list) or the board's INTERNAL-BUSY status is currently set. In contrast, the command is normally executed when a list has been paused by set_wait (PAUSED status set). • For 3D image field calibration, see chapter 8.6.3 "3D Marking Commands", page 194.
RTC4→ RTC5	Unchanged functionality (except for the extended range of X and Y values). In RTC4 compatibility mode, the RTC5 multiplies the specified X and Y coordinate values by 16 (the allowed range of values is correspondingly reduced to [-32768 ... 32767]). The value range for Z coordinates is identical for the RTC5 and RTC4.
References	load_z_table



Ctrl Command	goto_xy
Function	Moves the output point (of the laser focus) along a 2D vector at jump speed from the current position to the specified position (absolute coordinate values) within a two-dimensional image field.
Call	<code>goto_xy(X, Y)</code>
Parameters	X, Y Absolute coordinates of the jump vector end point in <i>bits</i> as signed 32-bit values. Allowed range: [-524288 ... 524287]. Out-of-range values are edge-clipped.
Comments	<ul style="list-style-type: none"> If the jump speed has not been previously explicitly set by <code>set_jump_speed</code> or <code>set_jump_speed_ctrl</code>, then the jump is executed at a predefined jump speed of 10000 <i>bits per ms</i>. <code>goto_xy</code> (unlike the list commands <code>jump_abs</code> and <code>jump_rel</code>) has no effect on the laser control signals and also does not set a jump delay. During the conversion of specified coordinate values into scan system output values, any previously defined coordinate transformations, assigned correction tables etc. are taken into account after successful microvectorization (see page 141). As of version DLL 520, OUT 519, coordinate transformation settings that were only collected until then possibly take effect by <code>goto_xy</code> (see page 185, <code>at_once = 2</code>). The command is ignored (<code>get_last_error</code> return code: <code>RTC5_BUSY</code>) if the board's BUSY status is currently set (list is being processed or has been halted by <code>pause_list</code>) or the board's INTERNAL-BUSY status is currently set. In contrast, the command is executed when a list has been paused by <code>set_wait</code> (PAUSED status set). The command is not executed (<code>get_last_error</code> return code: <code>RTC5_BUSY</code>), if an external stop signal (/STOP, /STOP2 or /Slave STOP) is present. The INTERNAL-BUSY status is set while <code>goto_xy</code> is executed. As of version DLL 527, OUT 529, the <code>goto_xy</code> command only returns to the user program after microvectorization has been executed. Therefore, you cannot execute any control command, while a <code>goto_xy</code>-induced jump is being executing.
RTC4→ RTC5	<ul style="list-style-type: none"> Extended range of values. The command returns only after microvectorization has been executed (see comments above). The image field coordinates for the X and Y axes are specified as 20-bit values; in RTC4 compatibility mode they are specified as 16-bit values (as with the RTC4) and the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced to [-32768 ... 32767]).
Version info	Last changes: <ul style="list-style-type: none"> with version DLL 520, OUT 519: behavior for coordinate transformations with <code>at_once = 2</code> (see page 694). with version DLL 527, OUT 529: The command returns only after the motion has completed.
References	set_jump_speed , jump_abs , jump_rel , goto_xyz , get_status



Ctrl Command	goto_xyz
Function	Moves the output point (of the laser focus) along a 3D vector at jump speed from the current position to the specified position (absolute coordinate values) within a three-dimensional process volume.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as goto_xy . However, microvectorization is calculated like a 3D command and hence influences the effective jump speed in the XY plane.
Call	<code>goto_xyz(X, Y, Z)</code>
Parameters	X, Y, Z Absolute coordinates of the jump vector end point in <i>bits</i> as signed 32-bit values. Allowed range: <ul style="list-style-type: none">• For X and Y: [-524288 ... 524287].• For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped.
Comments	<ul style="list-style-type: none">• Except for the additional motion in the third dimension, this command functions similarly to the goto_xy command (see the comments there).• The X and Y axes can be controlled with 20-bit resolution, the Z axis with 16-bit resolution. The RTC5 (in RTC5 mode as well as in RTC4 compatibility mode) automatically upscales Z coordinate values internally to 20-bit values, so that the three dimensions of space can be handled equivalently in terms of the supplied jump speed value.• The DirectMove3D parameter of the set_delay_mode command determines the type of Z-axis motion (linear or with stepwise correction).• During calculation of the Z output value to the scan system, any previously defined offset to the Z coordinate and focal length is taken into account (see chapter 7.3.5 "Output Values to the Scan System", page 141).
RTC4→ RTC5	<ul style="list-style-type: none">• Extended range of values.• The command returns only after the motion has completed (see comments for goto_xy).• RTC4 compatibility mode: see goto_xy, the value range for Z coordinates is identical for the RTC5 and RTC4.
Version info	Last changes: <ul style="list-style-type: none">• with version DLL 520, OUT 519: behavior for coordinate transformations with <code>at_once = 2</code> (see page 694).• with version DLL 527, OUT 529: The command returns only after the motion has completed.
References	goto_xy , jump_abs_3d , jump_rel_3d



Ctrl Command	home_position
Function	Activates the home jump mode (for the X and Y axes) and defines the home position.
Call	<code>home_position(XHome, YHome)</code>
Parameters	<p>XHome, Absolute coordinates of the home position in <i>bits</i> as signed 32-bit values. YHome Allowed range: [-524288...524287]. Larger values are clipped.</p>
Comments	<ul style="list-style-type: none"> This command defines the coordinates of a home jump to be executed, for example, upon reaching the end of a list. Analogously, a list starts with a home return to the most recently valid position. Home jump and home return are executed at jump speed. During the conversion of specified coordinate values into scan system output values, any previously defined coordinate transformations, assigned correction tables etc. are taken into account after successful microvectorization (see chapter 7.3.5 "Output Values to the Scan System", page 141). This command is intended for a laser system that does not allow fast switching of the laser. After calling the command, the laser focus moves to the specified home position whenever no list is executing or when a list has been paused by set_wait. A home jump is also executed, if list execution is stopped by stop_execution or by an external stop signal. The home jump itself (in contrary to a home return) cannot be stopped. While a home jump or a home return is executed, the INTERNAL-BUSY status is set. A <i>beam dump</i> should be placed in the home position. The home jump mode is deactivated with the command <code>home_position(0, 0)</code>, even if it was activated by home_position_xyz.
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). The image field coordinates for the X and Y axes are specified as 20-bit values; in RTC4 compatibility mode they are specified as 16-bit values (as with the RTC4) and the RTC5 multiplies the specified coordinate values for the home position by 16 (the allowed range of values is correspondingly reduced to [-32768 ... 32767]).
References	home_position_xyz



Ctrl Command	home_position_xyz
Function	Activates the home jump mode (for the X, Y and Z axes) and defines the home position.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as home_position . However, microvectorization is calculated like a 3D command and hence influences the effective jump speed in the XY plane.
Call	<code>home_position_xyz(XHome, YHome, ZHome)</code>
Parameters	<p>XHome, Absolute coordinates of the home position in <i>bits</i> as signed 32-bit values. YHome, Allowed range: ZHome • for XHome and YHome: [-524288 ... 524287] • for ZHome: [-32768 ... 32767] Out-of-range values are edge-clipped.</p>
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, this command functions similarly to the home_position command (see the comments there). The home jump mode is deactivated with the command <code>home_position_xyz(0, 0, 0)</code> or <code>home_position_xy(0, 0)</code>. The X and Y axes can be controlled with 20-bit resolution, the Z axis with 16-bit resolution. The RTC5 (in RTC5 mode as well as in RTC4 compatibility mode) automatically upscales Z coordinate values internally to 20-bit values, so that the three dimensions of space can be handled equivalently in terms of the supplied jump speed value. The <code>DirectMove3D</code> parameter of the set_delay_mode command determines the type of Z-axis motion (linear or with stepwise correction). During calculation of the Z output value to the scan system, any previously defined offset to the Z coordinate and focal length is taken into account (see chapter 7.3.5 "Output Values to the Scan System", page 141).
RTC4→ RTC5	New command. RTC4 compatibility mode: see home_position , the value range for Z coordinates is identical for the RTC5 and RTC4.
References	home_position



Undelayed Short List Command	if_cond
Function	<p><i>Conditional command execution:</i> This command immediately executes the directly following list command, if the current IOvalue at the EXTENSION 1 socket connector's 16-bit digital input port meets the following condition:</p> $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } ((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0}$ <p>(i.e. if the bits specified in Mask1 are 1 and the bits specified in Mask0 are 0). Otherwise, the next list command is skipped over.</p>
Call	<code>if_cond(Mask1, Mask0)</code>
Parameters	<p>Mask1, 16-bit masks as unsigned 32-bit values. Mask0 Only the least significant 16 bits are evaluated.</p>
Comments	<ul style="list-style-type: none"> • See also chapter 9.3.2 "Conditional Command Execution", page 244.
RTC4→ RTC5	New command.
References	if_not_cond , if_pin_cond , if_not_pin_cond

Undelayed Short List Command	if_fly_x_overflow
Function	<p><i>Conditional command execution for Processing-on-the-fly applications:</i> This command immediately executes the directly subsequent list command if the condition in accordance with Mode for the X axis was fulfilled. Otherwise, the next list command is skipped over.</p>
Call	<code>if_fly_x_overflow(Mode)</code>
Parameters	<p>Mode To-be-evaluated condition as a signed 32-bit value.</p> <p>= 0: Some kind of boundary exceedance occurred (error bit#4 = 1 or error bit#5 = 1).</p> <p>> 0: An overflow occurred (error bit#5 = 1).</p> <p>< 0: An underflow occurred (error bit#4 = 1).</p>
Comments	<ul style="list-style-type: none"> • For command usage, see section "Customer-Defined Monitoring Area and Conditional Command Execution (as of Version DLL 525, OUT 527)", page 214. • The error bits queried in accordance with Mode (error bit#4 and/or error bit#5) are reset.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 525, OUT 527.
References	get_marking_info , set_fly_limits , clear_fly_overflow , if_not_fly_x_overflow



Undelayed Short List Command	if_fly_y_overflow
Function	<i>Conditional command execution for Processing-on-the-fly applications:</i> This command immediately executes the directly subsequent list command if the condition in accordance with Mode for the Y axis was fulfilled. Otherwise, the next list command is skipped over.
Call	if_fly_y_overflow(Mode)
Parameters	Mode To-be-evaluated condition as a signed 32-bit value: = 0: Some kind of boundary exceedance occurred (error bit#6 = 1 or error bit#7 = 1). > 0: An overflow occurred (error bit#7 = 1). < 0: An underflow occurred (error bit#6 = 1).
Comments	<ul style="list-style-type: none"> For command usage, see section "Customer-Defined Monitoring Area and Conditional Command Execution (as of Version DLL 525, OUT 527)", page 214. The error bits queried in accordance with Mode (error bit#6 and/or error bit#7) are reset.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 525, OUT 527.
References	get_marking_info , set_fly_limits , clear_fly_overflow , if_not_fly_y_overflow

Undelayed Short List Command	if_fly_z_overflow
Function	<i>Conditional command execution for Processing-on-the-fly applications:</i> This command immediately executes the directly subsequent list command if the condition in accordance with Mode for the Z axis was fulfilled. Otherwise, the next list command is skipped over.
Call	if_fly_z_overflow(Mode)
Parameters	Mode To-be-evaluated condition as a signed 32-bit value: = 0: Some kind of boundary exceedance occurred (error bit#24 = 1 or error bit#25 = 1). > 0: An overflow occurred (error bit#25 = 1). < 0: An underflow occurred (error bit#24 = 1).
Comments	<ul style="list-style-type: none"> For command usage, see also chapter 8.7.9 "Monitoring Processing-on-the-fly Corrections", page 213. The error bits queried in accordance with Mode (error bit#24 and/or error bit#25) are reset.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 531, OUT 532.
References	get_marking_info , set_fly_limits_z , clear_fly_overflow , if_not_fly_z_overflow

Undelayed Short List Command	if_not_activated
Function	Conditional command execution due to an error bit from activate_fly_xy or activate_fly_2d : If the error bit is set, then the next list command executes immediately, otherwise it is skipped.
Call	<code>if_not_activated()</code>
Comments	<ul style="list-style-type: none"> See also comments at activate_fly_2d und activate_fly_xy. It is useful to insert a list jump or subroutine call in the list directly after if_not_activated that jumps to an error-handling sequence. Or you could simply insert a set_end_of_list. if_not_activated resets the error bit from activate_fly_xy or activate_fly_2d. If you still need it for subsequent querying by get_marking_info (bit #9 = 1), then you can include a renewed call of activate_fly_2d or activate_fly_xy in your error-handling sequence to set the error bit again (provided that the error situation still exists). Successful activation by activate_fly_2d or activate_fly_xy does not reset any already-set error bit. It remains set for get_marking_info until get_marking_info is called.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 536, OUT 536.
References	activate_fly_2d , activate_fly_xy , get_marking_info

Undelayed Short List Command	if_not_cond
Function	<p><i>Conditional command execution:</i> This command immediately executes the directly following list command, if the current I0value at the EXTENSION 1 socket connector's 16-bit digital input port does not meet the following condition:</p> $((\text{I0value AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not I0value}) \text{ AND Mask0}) = \text{Mask0})$ <p>(i.e. if the bits specified in Mask1 are not 1 or the bits specified in Mask0 are not 0). Otherwise, the next list command is skipped over.</p>
Call	<code>if_not_cond(Mask1, Mask0)</code>
Parameters	<p>Mask1, 16-bit masks as unsigned 32-bit values. Mask0 Only the least significant 16 bits are evaluated.</p>
Comments	<ul style="list-style-type: none"> See also chapter 9.3.2 "Conditional Command Execution", page 244.
RTC4→ RTC5	New command.
References	if_cond , if_pin_cond , if_not_pin_cond



Undelayed Short List Command	if_not_fly_x_overflow
Function	<i>Conditional command execution for Processing-on-the-fly applications:</i> This command immediately executes the directly subsequent list command if the condition in accordance with Mode for the X axis was not fulfilled. Otherwise, the next list command is skipped over.
Call	if_not_fly_x_overflow(Mode)
Parameters	Mode To-be-evaluated condition as a signed 32-bit value. = 0: Some kind of boundary exceedance occurred (error bit#4 = 1 or error bit#5 = 1). > 0: An overflow occurred (error bit#5 = 1). < 0: An underflow occurred (error bit#4 = 1).
Comments	<ul style="list-style-type: none"> For command usage, see section "Customer-Defined Monitoring Area and Conditional Command Execution (as of Version DLL 525, OUT 527)", page 214. The error bits queried in accordance with Mode (error bit#4 and/or error bit#5) are reset.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 525, OUT 527.
References	get_marking_info , set_fly_limits , clear_fly_overflow , if_fly_x_overflow

Undelayed Short List Command	if_not_fly_y_overflow
Function	<i>Conditional command execution for Processing-on-the-fly applications:</i> This command immediately executes the directly subsequent list command if the condition in accordance with Mode for the Y axis was not fulfilled. Otherwise, the next list command is skipped over.
Call	if_not_fly_y_overflow(Mode)
Parameters	Mode To-be-evaluated condition as a signed 32-bit value. = 0: Some kind of boundary exceedance occurred (error bit#6 = 1 or error bit#7 = 1). > 0: An overflow occurred (error bit#7 = 1). < 0: An underflow occurred (error bit#6 = 1).
Comments	<ul style="list-style-type: none"> For command usage, see section "Customer-Defined Monitoring Area and Conditional Command Execution (as of Version DLL 525, OUT 527)", page 214. The error bits queried in accordance with Mode (error bit#6 and/or error bit#7) are reset.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 525, OUT 527.
References	get_marking_info , set_fly_limits , clear_fly_overflow , if_fly_y_overflow



Undelayed Short List Command	if_not_fly_z_overflow
Function	<i>Conditional command execution for Processing-on-the-fly applications:</i> This command immediately executes the directly subsequent list command if the condition in accordance with <code>Mode</code> for the Z axis was not fulfilled. Otherwise, the next list command is skipped over.
Call	<code>if_not_fly_z_overflow(Mode)</code>
Parameters	<code>Mode</code> To-be-evaluated condition as a signed 32-bit value. = 0: Some kind of boundary exceedance occurred (error bit#24 = 1 or error bit#25 = 1). > 0: An overflow occurred (error bit#25 = 1). < 0: An underflow occurred (error bit#24 = 1).
Comments	<ul style="list-style-type: none"> For command usage, see also "Monitoring Processing-on-the-fly Corrections", page 213. The error bits queried in accordance with <code>Mode</code> (error bit#24 and/or error bit#25) are reset.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 531, OUT 532.
References	get_marking_info , set_fly_limits_z , clear_fly_overflow , if_fly_z_overflow

Undelayed Short List Command	if_not_pin_cond
Function	<i>Conditional command execution:</i> This command immediately executes the directly following list command, if the current <code>I0value</code> at the LASER connector's 2-bit digital input port does not meet the following condition: $((I0value \text{ AND } Mask1) = Mask1) \text{ AND } (((\text{not } I0value) \text{ AND } Mask0) = Mask0)$ (i.e. if the bits specified in <code>Mask1</code> are not 1 or the bits specified in <code>Mask0</code> are not 0). Otherwise, the next list command is skipped over.
Call	<code>if_not_pin_cond(Mask1, Mask0)</code>
Parameters	<code>Mask1</code> , 2-bit masks as unsigned 32-bit values. <code>Mask0</code> Only the least significant 2 bits are evaluated.
Comments	<ul style="list-style-type: none"> See also "Conditional Command Execution", page 244.
RTC4→ RTC5	New command.
References	if_pin_cond , if_cond , if_not_cond



Undelayed Short List Command	if_pin_cond
Function	<p><i>Conditional command execution:</i> This command immediately executes the directly following list command, if the current IOvalue at the LASER connector's 2-bit digital input port meets the following condition:</p> $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND } \text{Mask0}) = \text{Mask0})$ <p>(i.e. if the bits specified in Mask1 are 1 and the bits specified in Mask0 are 0). Otherwise, the next list command is skipped over.</p>
Call	<code>if_pin_cond(Mask1, Mask0)</code>
Parameters	<p>Mask1, 2-bit masks as unsigned 32-bit values. Mask0 Only the least significant 2 bits are evaluated.</p>
Comments	<ul style="list-style-type: none"> • See also "Conditional Command Execution", page 244.
RTC4→ RTC5	New command.
References	if_not_pin_cond , if_cond , if_not_cond

Ctrl Command	init_fly_2d
Function	Initializes the encoder reference values for 2D encoder compensation of a subsequent set_fly_2d Processing-on-the-fly application and resets both encoder values.
Call	<code>init_fly_2d(OffsetX, OffsetY)</code>
Parameters	<p>OffsetX, Reference values of the X and Y axis encoders (signed 32-bit values) OffsetY Allowed range: depends on the compensation table loaded with load_fly_2d_table. Default values (after load_program_file): = 0.</p>
Comments	<ul style="list-style-type: none"> • The final encoder value for the 2D correction (i.e. current encoder value + reference value) must not exceed the allowed range. Otherwise clipping occurs (see also load_fly_2d_table).
RTC4→ RTC5	New command.
Version info	Available as of version DLL 536, OUT 536.
References	set_fly_2d , load_fly_2d_table , get_fly_2d_offset

Ctrl Command	init_rtc5_dll
Function	Initializes control of the installed RTC5 Boards for a user program.
Call	<code>InitErrorNo = init_rtc5_dll()</code>
Result	Error code as an unsigned 32-bit value. If multiple errors occurred simultaneously, then multiple bits are set. The meanings of bit numbers, error types and error constants is identical to those for the command get_error .
Comments	<ul style="list-style-type: none"> The command <code>init_rtc5_dll</code> must be called before starting each user program. Otherwise no RTC5s are accessible by the user program. <code>init_rtc5_dll</code> detects all available RTC5 Boards and establishes corresponding board management. If no RTC5 Boards are found, the error code <code>RTC5_NO_CARD</code> is returned. If an error occurs when reading the PCI configuration register (<code>get_last_error</code> return code <code>RTC5_CONFIG_ERROR</code>), then an <code>RTC5_ACCESS_DENIED</code> error is also generated. The board then remains inaccessible until the error gets cleared by a PC restart. The <code>init_rtc5_dll</code> command automatically assigns the user program access rights to the found boards (as by an <code>acquire_rtc</code> command) if access rights are not already assigned to another user program (any number of boards and applications may be used, but each particular board cannot be used simultaneously by multiple applications). The first initialization acquires all found boards for itself. Subsequent initializations started by other applications result in return of an <code>RTC5_ACCESS_DENIED</code> error code by the <code>init_rtc5_dll</code> command. The boards are then only accessible by these applications through DLL-internal functions that require no access rights – e.g. <code>get_error</code>, <code>get_last_error</code> or <code>select_rtc</code> (most multi-board commands, too, are only callable for boards with existing access rights). If a user program has access rights for a board, then the user program must first explicitly release its access rights with <code>release_rtc</code> or <code>free_rtc5_dll</code> before the board can be used by another user program by <code>init_rtc5_dll</code> or <code>acquire_rtc</code>. An <code>RTC5_ACCESS_DENIED</code> error code is returned if access was denied by at least one of the found boards. Which board(s) denied access can be determined by <code>get_error(CardNo)</code> (CardNo from 1 to the number of found boards) or directly after <code>init_rtc5_dll</code> (before the next command) by <code>get_last_error(CardNo)</code>. If a board is acquired by <code>init_rtc5_dll</code> (as by <code>acquire_rtc</code>, then a version compatibility check is performed. If a version error is detected, then access to the board is denied (return code <code>RTC5_ACCESS_DENIED RTC5_VERSION_MISMATCH</code>). Only one user program can perform initialization at any one time. Subsequent initializations started by other applications wait until the current initialization is complete. If a user program calls the <code>init_rtc5_dll</code> command multiple times, then the board management created by the prior call is deleted for this user program and the originally-assigned access rights canceled. Board management is then newly created and access rights newly assigned.



Ctrl Command	init_RTC5_DLL
Comments (cont'd)	<ul style="list-style-type: none"> Each initialization of a user program by <code>init_RTC5_DLL</code> causes the DLL-internal numbers for all found RTC5 Boards to be newly reassigned. The relationship between the DLL-internal numbers and the installed boards can be determined by <code>get_serial_number</code>. When the accessible board with the smallest DLL-internal number is initialized, it then simultaneously becomes the active board and the target of non-multi-board commands. <code>select_RTC</code> can be called at anytime to change the active board. If no access rights exist for any board, then the board with the highest DLL-internal number (see <code>rtc5_count_cards</code>) is the active board, in which case only DLL-internal commands that require no access rights can be used. Also observe chapter 6.7.1 "Board Acquisition by a User Program", page 96. The <code>init_RTC5_DLL</code> command is also available without explicit access rights to any particular RTC5 Board. <code>init_RTC5_DLL</code> is not available as a multi-board command. After initialization of the DLL with <code>init_RTC5_DLL</code>, you can additionally select between two DLL operational modes (see <code>set_RTC4_Mode</code> and <code>set_RTC5_Mode</code>). The default is the RTC5 mode. The <code>init_RTC5_DLL</code> command does not cause a reset of the RTC5 Board. Board resets can only be initiated by the command <code>load_program_file</code>.
RTC4 → RTC5	New command.



Normal List Command	jump_abs
Function	Moves the output point (of the laser focus) along a 2D vector at jump speed from the current position to the specified position (absolute coordinate values) within a two-dimensional image field.
Call	<code>jump_abs(X, Y)</code>
Parameters	X, Y Absolute coordinates of the jump vector end point in <i>bits</i> as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.
Comments	<ul style="list-style-type: none"> If the jump speed has not been previously explicitly set by set_jump_speed or set_jump_speed_ctrl, then the jump is executed at a predefined jump speed of 10000 <i>bits per ms</i>. The “laser active” laser control signals are off during the jump (if necessary, they are switched off before the jump). After a jump command, a (variable) jump delay is inserted. Exception: a zero-length jump vector’s subsequent (variable) jump delay is not executed. However, the command itself still requires a 10 µs clock cycle for execution. During the conversion of specified coordinate values into scan system output values, any previously defined coordinate transformations, assigned correction tables etc. are taken into account after successful microvectorization (see page 141). As of version DLL 520, OUT 519, coordinate transformation settings that were only collected until then possibly take effect by jump_abs (see page 185, <code>at_once = 2</code>).
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).
Version info	Last change with version DLL 520, OUT 519: behavior for coordinate transformations with <code>at_once = 2</code> (see page 694).
References	set_jump_speed , set_scanner_delays , jump_rel , timed_jump_abs , goto_xy



Normal List Command	jump_abs_3d
Function	Moves the output point (of the laser focus) along a 3D vector at jump speed from the current position to the specified position (absolute coordinate values) within a three-dimensional process volume.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as jump_abs . However, microvectorization is calculated like a 3D command and hence influences the effective jump speed in the XY plane.
Call	<code>jump_abs_3d(X, Y, Z)</code>
Parameters	X, Y, Z Absolute coordinates of the jump vector end point in <i>bits</i> as signed 32-bit values. Allowed range: <ul style="list-style-type: none"> For X and Y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped.
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, this command functions similarly to the jump_abs command (see the comments there). The X and Y axes can be controlled with 20-bit resolution, the Z axis with 16-bit resolution. The RTC5 (in RTC5 mode as well as in RTC4 compatibility mode) automatically upscales Z coordinate values internally to 20-bit values, so that the three dimensions of space can be handled equivalently in terms of the supplied jump speed value. The DirectMove3D parameter of the set_delay_mode command determines the type of Z-axis motion (linear or with stepwise correction). During calculation of the Z output value to the scan system, any previously defined offset to the Z coordinate and focal length is taken into account (see chapter 7.3.5 "Output Values to the Scan System", page 141).
RTC4→RTC5	Unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 and RTC4.
References	jump_abs , jump_rel_3d , timed_jump_abs_3d



Normal List Command	jump_rel
Function	Moves the output point (of the laser focus) along a 2D vector at jump speed from the current position to the specified position (relative coordinate values) within a two-dimensional image field.
Call	<code>jump_rel(dx, dy)</code>
Parameters	<p><code>dx, dy</code> <i>Relative coordinates of the jump vector end point in bits as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.</i></p> <p>The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</p>
Comments	<ul style="list-style-type: none"> The coordinates for the jump vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to jump_abs (see the comments there).
RTC4→ RTC5	<p>Unchanged functionality (except for the extended range of values).</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).</p>
Version info	Last change with version DLL 520, OUT 519: behavior for coordinate transformations with <code>at_once = 2</code> (see page 694).
References	jump_abs, jump_rel_3d, timed_jump_rel



Normal List Command	jump_rel_3d
Function	Moves the output point (of the laser focus) along a 3D vector at jump speed from the current position to the specified position (relative coordinate values) within a three-dimensional process volume.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as jump_rel . However, microvectorization is calculated like a 3D command and hence influences the effective jump speed in the XY plane.
Call	<code>jump_rel_3d(dx, dy, dz)</code>
Parameters	<p><code>dx, dy, dz</code> <i>Relative coordinates of the jump vector end point in bits as signed 32-bit values. Allowed range:</i></p> <ul style="list-style-type: none"> • For X and Y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. • For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped.
Comments	<ul style="list-style-type: none"> • The coordinates for the jump vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to jump_abs_3d (see the comments there).
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 and RTC4.
References	jump_abs_3d , jump_abs , jump_rel , timed_jump_rel_3d



Variable List Command	laser_on_list
Function	Turns on the “laser active” laser control signals for a specified time interval.
Call	<code>laser_on_list(Period)</code>
Parameter	<p>Period time interval in <i>bits</i> as an unsigned 32-bit value. <i>1 bit equals 10 µs.</i> <i>Allowed range: 0 ≤ Period ≤ (2³¹–1).</i></p>
Comments	<ul style="list-style-type: none"> As of Version OUT 531, bit #31 of Period is ignored (see also comments and version info at laser_on_pulses_list). The laser control signals for “laser active” operation must first be selected with set_laser_mode, defined by further commands, and enabled by set_laser_control or enable_laser before they can be switched on with laser_on_list (see chapter 7.4 “Laser Control”, page 144). While the laser control signals are turned on, the set position of the scanners is not changed. The next list command is executed when the programmed time interval has passed. The currently set LaserOn delay is applied at the beginning of the programmed time interval: The laser control signals turn on after a LaserOn delay. <ul style="list-style-type: none"> As with other mark commands (e.g. mark_abs), the laser control signals do <i>not</i> explicitly turn off at the end of the time interval, but instead only turn off with a subsequent list command (e.g. jump_abs or list_nop). The currently set LaserOff delay is applied. The command laser_on_list is useful for marking separate points (see chapter 7.1.3 “Marking Points”, page 108). The Wobbel Mode (see chapter 8.4 “Wobbel Mode”, page 189) is retained but ignored. For Period = 0 the laser control signals do not turn on; then the laser_on_list command is a short list command. The command laser_on_list does not trigger a scanner delay. To wait until the laser (after a LaserOff delay) is actually off before a jump, then explicitly a long_delay should be inserted.
RTC4→RTC5	Unchanged functionality (except for the extended range of values).
Version info	Last change with version DLL 530, OUT 531, RBF 522: changed functionality for $2^{31} \leq \text{Period} \leq (2^{32}-1)$ (see comments for laser_on_pulses_list).
References	laser_on_pulses_list , long_delay , para_laser_on_pulses_list



Variable List Command	laser_on_pulses_list
Function	Turns on the LASERON laser control signal for the specified number of external signal pulses, but for no longer than the specified time interval. For Pulses > 65535 the function of the command is identical to laser_on_list .
Call	<code>laser_on_pulses_list(Period, Pulses)</code>
Parameters	<p>Period Time interval in <i>bits</i> as an unsigned 32-bit value. <i>1 bit equals 10 µs.</i> Allowed range: $0 \leq \text{Period} \leq (2^{32}-1)$.</p> <p>Pulses Number of external signal pulses as an unsigned 32-bit value. Allowed range: $0 \leq \text{Pulses} \leq 65535$ or larger (see comments below).</p>
Comments	<ul style="list-style-type: none"> The command is useful for marking separate points in laser mode 6 (see chapter 7.4.7 "Laser Mode 6", page 155), but also effective in the other laser modes. The external pulses must be supplied as TTL pulses at the LASER connector's DIGITAL IN1 digital input (see section "Digital Input and Output", page 48). By set_laser_control(bit #5), you can specify whether signal pulses should be counted at rising or falling edges. If <code>Period = 0</code>, then the command has no effect and <code>laser_on_pulses_list</code> becomes a short list command. If $0 < \text{Period} \leq (2^{31}-1)$, then the command's duration is always <code>Period</code> clocks (i.e. <code>Period</code> \times 10 µs), even if the specified number of external signal pulses expire in a shorter time interval. If $2^{31} \leq \text{Period} \leq (2^{32}-1)$, the command's maximum duration is $(\text{Period} - 2^{31})$ clocks (i.e. $(\text{Period} - 2^{31}) \times 10 \mu s$). Here, however, the command terminates as soon as the specified number of external signal pulses has been detected. If <code>Pulses > 65535</code>, then <code>laser_on_pulses_list</code>'s function is identical to laser_on_list (external signal pulses are not taken into account, see comments there). Otherwise (for $0 \leq \text{Pulses} \leq 65535$) <code>laser_on_pulses_list</code> (in contrast to laser_on_list) <i>do not</i> toggle the laser control signals between "laser active" and "laser standby" operation, but instead only switches the LASERON signal (hence not the LASER1 and LASER2 signals), whereby laser delays are not taken into account. Likewise during this command, unexpired laser delays activated by prior commands have no effect (though their effect resume when the LASERON signal switches off again after the final pulse or after <code>Period</code>). If $1 \leq \text{Pulses} \leq 65535$, then the LASERON signal does switch on upon the edge (according the polarity set by set_laser_control(bit#5)) of the first external pulse (unless it is already on due to an unexpired LaserOff delay) and remains on for the specified number of pulses, but no longer than until the end of the number of 10 µs periods specified by <code>Period</code>. Users should ensure to define <code>Period</code> large enough for processing <code>Pulses</code> number of signal pulses in this time interval. If the DIGITAL IN1 input does not receive any pulses, then <code>laser_on_pulses_list</code> does not alter the LASERON signal. If more signal pulses than specified by <code>Pulses</code> are received during the time interval defined by <code>Period</code>, then the surplus pulses are ignored.



Variable List Command	laser_on_pulses_list
Comments (cont'd)	<ul style="list-style-type: none"> If Pulses = 0, then laser_on_pulses_list has no effect (the LASERON signal is not switched on). Then laser_on_pulses_list becomes a short list command. The LASERON signal must first be defined and enabled by set_laser_control or enable_laser before it can be switched on with laser_on_pulses_list (see chapter 7.4 "Laser Control", page 144). While the command is executed, the set position of the scanners is not changed. The next list command is executed when the programmed time interval Period has passed. The Wobbel mode (see chapter 8.4 "Wobbel Mode", page 189) is retained but ignored. Any softstarts that were defined are not executed.
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> Available as of version DLL 515, OUT 514, RBF 512. Changed with version DLL 530, OUT 531, RBF 522: changed functionality for $2^{31} \leq \text{Period} \leq (2^{32}-1)$.
References	laser_on_list, para_laser_on_pulses_list

Ctrl Command	laser_signal_off
Function	Turns off the "laser active" laser control signals immediately.
Call	<code>laser_signal_off()</code>
Comments	<ul style="list-style-type: none"> This command is intended for direct laser control in combination with the command laser_signal_on. The command is ignored (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set, i.e. if a list is executing at the moment or if a list has been halted by pause_list. In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set) or when the board's INTERNAL-BUSY status is currently set.
RTC4→ RTC5	Unchanged.
References	laser_signal_on

Normal List Command	laser_signal_off_list
Function	Same as laser_signal_off , but a list command.
Call	<code>laser_signal_off_list()</code>
RTC4→ RTC5	Unchanged.



Ctrl Command	laser_signal_on
Function	Turns on the “laser active” laser control signals immediately.
Call	<code>laser_signal_on()</code>
Comments	<ul style="list-style-type: none"> The laser control signals for “laser active” operation must first be selected with <code>set_laser_mode</code>, defined by further commands, and enabled by <code>set_laser_control</code> or <code>enable_laser</code> before they can be switched on with <code>laser_on_list</code> (see chapter 7.4 “Laser Control”, page 144). The command is intended for turning on the laser directly, e.g. for alignment purposes. The “laser active” laser control signals must be turned off with the command <code>laser_signal_off</code>. The command is ignored (<code>get_last_error</code> return code: <code>RTC5_BUSY</code>) if the board’s BUSY status is currently set (list is being processed or has been halted by <code>pause_list</code>) or the board’s INTERNAL-BUSY status is currently set. In contrast, the command is executed when a list has been paused by <code>set_wait</code> (PAUSED status set). <i>Caution! Check the beam path before turning on the laser!</i>
RTC4→ RTC5	Unchanged.
References	laser_signal_off

Normal List Command	laser_signal_on_list
Function	Same as <code>laser_signal_on</code> , but a list command and never ignored.
Call	<code>laser_signal_on_list()</code>
RTC4→ RTC5	Unchanged.



Undelayed Short List Command	list_call
Function	Causes an unconditional jump to a subroutine that starts at the specified absolute list buffer address (in any desired location within list memory).
Call	<code>list_call(Pos)</code>
Parameter	<code>Pos</code> Absolute jump address [0 ... (2 ²⁰ -1)] as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> • The first command of a subroutine called by <code>list_call</code> is executed (possibly after a <code>list_nop</code> command or, as of version DLL 525 after a <code>list_continue</code> command) immediately and without delay. Nested or recursive calls are also possible, up to a depth of 63 (see also chapter 6.5.1 "Subroutines", page 81). • Each subroutine must be terminated with the command <code>list_return</code> so that after the subroutine (including the terminating <code>list_return</code> command) has been processed, execution continues with the command that follows the subroutine-call command. This, too, executes (after a possible <code>list_nop</code> or <code>list_continue</code> command) immediately and without delay. <p>If <code>set_end_of_list</code> is encountered instead of the expected <code>list_return</code> command, then list execution terminates or – if previously activated – an automatic list change takes place (for the latter, the current list status is a decisive factor as described below). Under no circumstances does program flow then return again to the calling location, even in the case of nested subroutine calls. Any not-yet-completed <code>mark_text</code> command does not execute to completion.</p> <p>If the end of a list area ("List 1" or "List 2") is reached without having encountered a <code>list_return</code> or <code>set_end_of_list</code>, then execution continues at the start of the current list. If such a situation occurs in the protected memory area, then a compulsory <code>list_return</code> command is inserted and executed.</p> <ul style="list-style-type: none"> • The <code>list_call</code> command is replaced by a <code>list_nop</code> if <code>Pos > (2²⁰-1)</code> or if <code>Pos</code> is also the current address (<code>get_last_error</code> return code: <code>RTC5_PARAM_ERROR</code>). • If a called subroutine executes a further <code>list_call</code> command to the address of the calling <code>list_call</code> command (recursive call), then the resulting endless loop is terminated as soon as the 63-nested-call upper limit is reached. Further <code>list_call</code> commands are then ignored and the next command is instead executed. • If the subroutine starts directly at the address which follows <code>list_call</code>, then the subroutine is executed once again after <code>list_return</code> (see also comments on a missing corresponding function call in the <code>list_return</code> command description). <p>As of version OUT 540 the next processed command is the one which follows after <code>list_return</code> (see also <code>list_repeat...list_until</code>). This bypasses the possibly unwanted list processing.</p> <ul style="list-style-type: none"> • The BUSY list status readable by <code>read_status</code> is, if necessary, altered by <code>list_call</code> if the called address is in the list area ("List 1" or "List 2"). If this address is instead in protected memory ("List 3"), then the calling location's list status is retained because the protected area does not have its own list status. During execution of a subroutine in protected memory, <code>get_status</code> too returns the calling location's list execution status, and <code>get_out_pointer</code> returns the position of the calling <code>list_call</code> command as the output pointer's position.



Undelayed Short List Command	list_call
Comments (cont'd)	<ul style="list-style-type: none"> Absolute vector and arc commands execute absolutely after list_call is called. If the subroutine needs to execute at various locations within the image field, then either the subroutine can only contain relative mark, arc or jump commands or list_call_abs must be used instead. If list_call is at the last possible memory position in a list ("List 1" or "List 2"), then – even if automatic list changing was previously enabled – execution continues at the start of the same list after processing of the called subroutine. list_call(Pos) is a synonym for list_call_repeat(Pos, 1).
RTC4→ RTC5	Essentially unchanged functionality.
References	list_continue, list_repeat, list_return, list_until, list_call_abs, list_call_cond, list_call_repeat, sub_call

Undelayed Short List Command	list_call_abs
Function	Causes an unconditional jump to a subroutine that starts at the specified absolute list buffer address (in any desired area of list memory). In the called subroutine, any absolute vector and arc commands receive an offset (based on the current coordinates at the time of the call).
Call	list_call_abs(Pos)
Parameter	Pos Absolute jump address [0 ... (2 ²⁰ -1)] as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The list_call_abs command is basically identical to list_call (see the comments there). However, list_call_abs results in a different execution of absolute vector and arc commands within the called subroutine. When list_call_abs executes, an offset is established for the called subroutine and set according to the current position. Thereby, the current position is automatically considered when absolute vector and arc commands of the called subroutine are subsequently executed. Nested calls are taken into account when the offset is determined (with list_return, the previous offset values are re-established). Subroutines can thus contain absolute vector and arc commands even if they are intended to be repeated in different parts of the image field. If the called subroutine contains no absolute commands, then there is no difference between list_call_abs and list_call. list_call_abs(Pos) is a synonym for list_call_abs_repeat(Pos, 1).
RTC4→ RTC5	New command.
References	list_call, list_call_abs_cond, list_call_abs_repeat



Undelayed Short List Command	list_call_abs_cond						
Function	<p><i>Conditional subroutine call (AbsCall):</i> This command executes the command list_call_abs(Pos), if the current IOvalue at the EXTENSION 1 socket connector's 16-bit digital input port meets the following condition:</p> $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0})$ <p>(i.e. if the bits specified in Mask1 are 1 and the bits specified in Mask0 are 0). Otherwise, the directly following list command is immediately executed.</p>						
Call	<code>list_call_abs_cond(Mask1, Mask0, Pos)</code>						
Parameters	<table> <tr> <td>Mask1</td><td>16-bit masks as unsigned 32-bit values.</td></tr> <tr> <td>Mask0</td><td>Only the least significant 16 bits are evaluated.</td></tr> <tr> <td>Pos</td><td>Absolute jump address [0 ... ($2^{20}-1$)] as an unsigned 32-bit value.</td></tr> </table>	Mask1	16-bit masks as unsigned 32-bit values.	Mask0	Only the least significant 16 bits are evaluated.	Pos	Absolute jump address [0 ... ($2^{20}-1$)] as an unsigned 32-bit value.
Mask1	16-bit masks as unsigned 32-bit values.						
Mask0	Only the least significant 16 bits are evaluated.						
Pos	Absolute jump address [0 ... ($2^{20}-1$)] as an unsigned 32-bit value.						
Comments	<ul style="list-style-type: none"> • See list_call_abs. • See also chapter 9.3.2 "Conditional Command Execution", page 244. 						
RTC4→ RTC5	New command.						
References	list_call_abs						

Undelayed Short List Command	list_call_abs_repeat				
Function	Causes an unconditional jump to a subroutine that starts at the specified absolute list buffer address (in any desired area of list memory) and executes its body several times.				
Call	<code>list_call_abs_repeat(Pos, Number)</code>				
Parameter	<table> <tr> <td>Pos</td><td>Absolute jump address [0 ... ($2^{20}-1$)] as an unsigned 32-bit value (as with list_call_abs).</td></tr> <tr> <td>Number</td><td>Number of repetitions as unsigned 32-bit value. Number = 0 is treated as Number = 1.</td></tr> </table>	Pos	Absolute jump address [0 ... ($2^{20}-1$)] as an unsigned 32-bit value (as with list_call_abs).	Number	Number of repetitions as unsigned 32-bit value. Number = 0 is treated as Number = 1.
Pos	Absolute jump address [0 ... ($2^{20}-1$)] as an unsigned 32-bit value (as with list_call_abs).				
Number	Number of repetitions as unsigned 32-bit value. Number = 0 is treated as Number = 1.				
Comments	<ul style="list-style-type: none"> • <code>list_call_abs(Pos)</code> is a synonym for <code>list_call_abs_repeat(Pos, 1)</code>. • The command <code>list_call_abs_repeat</code> avoids an empty cycle at the repetition, which otherwise inevitably occurs with <code>list_call_abs...list_call_abs</code> or <code>list_repeat...list_call_abs...list_until</code> constructions. • See list_call_repeat and list_call_abs. 				
RTC4→ RTC5	New command.				
Version info	Available as of version DLL 540, OUT 540.				
References	list_call , list_call_abs , list_call_abs_cond , list_call_repeat , list_repeat , list_until				



Undelayed Short List Command	list_call_cond						
Function	<p><i>Conditional subroutine call:</i> This command executes the command list_call(Pos), if the current IOvalue at the EXTENSION 1 socket connector's 16-bit digital input port meets the following condition:</p> $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND } \text{Mask0}) = \text{Mask0})$ <p>(i.e. if the bits specified in Mask1 are 1 and the bits specified in Mask0 are 0). Otherwise, the directly following list command is immediately executed.</p>						
Call	<code>list_call_cond(Mask1, Mask0, Pos)</code>						
Parameters	<table> <tr> <td>Mask1,</td> <td>16-bit masks as unsigned 32-bit values.</td> </tr> <tr> <td>Mask0</td> <td>Only the least significant 16 bits are evaluated.</td> </tr> <tr> <td>Pos</td> <td>Absolute jump address [0 ... (2²⁰-1)] as an unsigned 32-bit value.</td> </tr> </table>	Mask1,	16-bit masks as unsigned 32-bit values.	Mask0	Only the least significant 16 bits are evaluated.	Pos	Absolute jump address [0 ... (2 ²⁰ -1)] as an unsigned 32-bit value.
Mask1,	16-bit masks as unsigned 32-bit values.						
Mask0	Only the least significant 16 bits are evaluated.						
Pos	Absolute jump address [0 ... (2 ²⁰ -1)] as an unsigned 32-bit value.						
Comments	<ul style="list-style-type: none"> • See list_call. • See also chapter 9.3.2 "Conditional Command Execution", page 244. 						
RTC4→ RTC5	Essentially unchanged functionality (see list_call).						
References	list_call						



Undelayed Short List Command	list_call_repeat
Function	Causes an unconditional jump to a subroutine that starts at the specified absolute list buffer address (in any desired area of list memory) and executes its body several times.
Call	list_call_repeat(Pos, Number)
Parameter	<p>Pos Absolute jump address [0 ... (2^{20}-1)] as an unsigned 32-bit value (as with list_call).</p> <p>Number Number of repetitions as unsigned 32-bit value. Number = 0 is treated as Number = 1.</p>
Comments	<ul style="list-style-type: none"> • <code>list_call_abs(Pos)</code> is a synonym for <code>list_call_abs_repeat(Pos, 1)</code>. • The command <code>list_call_repeat</code> avoids an empty cycle at the repetition, which otherwise inevitably occurs with <code>list_call...list_call</code> or <code>list_repeat...list_call...list_until</code> constructions. • With the command <code>list_call_repeat</code>, for example, trajectories from micro vector commands for runup curves and coast down curves can be seamlessly joined together with shapes from subroutines. • An empty cycle must be inserted if at <code>list_call_repeat</code> another subroutine call follows immediately (nested calls). This can be avoided, if there is e.g. at least one microvector command between two subroutine calls. • The subroutine start <code>Pos</code> can be located in any part of the list memory area. This applies in particular directly after the command <code>list_call_repeat</code>. Thus the complete list memory can continuously be used for list commands without reserving a special area for protected subroutines. • After a <code>list_return</code> the next processed command is the one which follows directly after <code>list_call_repeat</code>. However, if the subroutine start follows directly after <code>list_call_repeat</code>, then the next processed command is the one which follows directly after <code>list_return</code>.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 540, OUT 540.
References	list_call , list_call_abs , list_call_abs_repeat , list_repeat , list_return , list_until , micro_vector_abs , micro_vector_abs_3d , micro_vector_rel , micro_vector_rel_3d



Normal List Command	list_continue
Function	Inserts a null operation (no operation) into the list buffer.
Call	list_continue()
Comments	<ul style="list-style-type: none"> Execution of the list_continue command (as does list_nop) requires 10 µs. When this command immediately follows a short list command, it ensures (as does list_nop) that the subsequent list command only executes in the next 10 µs clock period (the short list command's "effective" execution time is then 10 µs). Unlike list_nop, however, list_continue neither modifies laser signals nor pauses for scanner delays. In contrast to list_nop, therefore, list_continue allows postponing short list commands to the next 10 µs clock period without interrupting polylines by switching off the laser. With exceedance of the maximum allowed number of directly consecutive short list commands, an empty cycle is automatically inserted in accordance with the command list_continue (this applies as of version OUT 519; with older versions, list_nop is automatically inserted instead: this would interrupt a polyline). You can also use the list_continue command to separate from each other several outputs to the same output port. Without this command, for example, multiple directly consecutive write_da_x_list commands (short list commands) would occur in a single 10 µs clock period, whereby some values to the analog output port would never get outputted. The RTC5 never uses list_continue as a placeholder for other (rejected) list commands, but instead always uses list_nop.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 525, OUT 527.
References	list_nop



Undelayed Short List Command	list_jump_cond						
Function	<p><i>Conditional (absolute) list jump:</i> This command executes the command list_jump_pos(Pos), if the current IOvalue at the EXTENSION 1 socket connector's 16-bit digital input port meets the following condition:</p> $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0})$ <p>(i.e. if the bits specified in Mask1 are 1 and the bits specified in Mask0 are 0). Otherwise, the directly following list command is immediately executed.</p>						
Call	<code>list_jump_cond(Mask1, Mask0, Pos)</code>						
Parameters	<table> <tr> <td>Mask1,</td> <td>16-bit masks as unsigned 32-bit values.</td> </tr> <tr> <td>Mask0</td> <td>Only the least significant 16 bits are evaluated.</td> </tr> <tr> <td>Pos</td> <td>Absolute jump address [0 ... (2²⁰-1)] as an unsigned 32-bit value.</td> </tr> </table>	Mask1,	16-bit masks as unsigned 32-bit values.	Mask0	Only the least significant 16 bits are evaluated.	Pos	Absolute jump address [0 ... (2 ²⁰ -1)] as an unsigned 32-bit value.
Mask1,	16-bit masks as unsigned 32-bit values.						
Mask0	Only the least significant 16 bits are evaluated.						
Pos	Absolute jump address [0 ... (2 ²⁰ -1)] as an unsigned 32-bit value.						
Comments	<ul style="list-style-type: none"> The command is synonymous with the list_jump_pos_cond command (see comments there). 						
RTC4→ RTC5	<p>Essentially unchanged functionality, however:</p> <ul style="list-style-type: none"> Extended range of Pos values. Jumps into or out from the protected buffer area ("List 3") are not allowed (illegal jump commands are ignored during processing, see also list_jump_pos). 						
References	list_jump_pos, list_jump_pos_cond						



Undelayed Short List Command	list_jump_pos
Function	Execution produces an unconditional jump to the specified list-memory address. The next command there is executed immediately without delay.
Call	<code>list_jump_pos(Pos)</code>
Parameter	<code>Pos</code> Absolute jump address [0 ... (2^{20} -1)] as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> • For <code>Pos</code>, an absolute address can be specified within the configured list memory ("List 1" and "List 2"), but not within the protected "List 3" area or completely outside of the list memory area. • Jumps into or out from the protected buffer area ("List 3") are not allowed. • Illegal jump commands are transmitted unaltered to the RTC5, but are ignored during processing. Instead, the next command is executed. Hence, the user program does probably no longer perform as expected. Therefore, jump commands must be carefully programmed (see also chapter 6.5.3 "Jumps", page 89). • Jump commands initiating a jump to themselves (<code>Pos</code> = list position of the jump command) are also ignored at runtime to prevent an infinite loop that excludes further activities (and that could only be halted by stop_execution or an external list stop). • Decisive are the runtime conditions. When reconfiguring list memory or converting a subroutine, an originally legal jump address might become illegal due to new list boundaries or a relocated subroutine storage position. • After a jump to another list area, the status information might under some circumstances not be correct (see also "List Status", page 76). • The BUSY list status of the two lists is alternatingly set by list_jump_pos, the USED list status of the two lists remains unchanged (see read_status). • The list_jump_pos command is synonymous with the set_list_jump command.
RTC4→ RTC5	New command.
References	set_list_jump , list_jump_rel , list_jump_pos_cond



Undelayed Short List Command	list_jump_pos_cond						
Function	<p><i>Conditional (absolute) list jump:</i> This command executes the command list_jump_pos(Pos), if the current IOvalue at the EXTENSION 1 socket connector's 16-bit digital input port meets the following condition:</p> $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0})$ <p>(i.e. if the bits specified in Mask1 are 1 and the bits specified in Mask0 are 0). Otherwise, the directly following list command is immediately executed.</p>						
Call	<code>list_jump_pos_cond(Mask1, Mask0, Pos)</code>						
Parameters	<table> <tr> <td>Mask1</td><td>16-bit masks as unsigned 32-bit values.</td></tr> <tr> <td>Mask0</td><td>Only the least significant 16 bits are evaluated.</td></tr> <tr> <td>Pos</td><td>Absolute jump address [0 ... (2^{20}-1)] as an unsigned 32-bit value.</td></tr> </table>	Mask1	16-bit masks as unsigned 32-bit values.	Mask0	Only the least significant 16 bits are evaluated.	Pos	Absolute jump address [0 ... (2^{20} -1)] as an unsigned 32-bit value.
Mask1	16-bit masks as unsigned 32-bit values.						
Mask0	Only the least significant 16 bits are evaluated.						
Pos	Absolute jump address [0 ... (2^{20} -1)] as an unsigned 32-bit value.						
Comments	<ul style="list-style-type: none"> • See list_jump_pos. • Unlike the rules for preventing endless loops (see list_jump_pos), jumps by list_jump_pos_cond are allowed even if they are to their own address (Pos = list position of the jump command), e.g. to wait for confirmation of a signal. • The command is synonymous with the list_jump_cond command. • See also chapter 9.3.2 "Conditional Command Execution", page 244. 						
Examples (Pascal)	<ul style="list-style-type: none"> • wait until bit #3 of the input port turns HIGH (= loop while the bit is LOW): <pre>list_jump_pos_cond(0, \$0008, get_input_pointer);</pre> • skip the next two list commands if the state of the input port is xxxx xxxx xxxx 0110 : <pre>list_jump_pos_cond(6, 9, get_input_pointer + 3);</pre> • See also "Programming Examples", page 245. 						
RTC4→ RTC5	New command.						
References	list_jump_pos						



Undelayed Short List Command	list_jump_rel
Function	Execution produces an unconditional jump to the specified address within the current list. The next command there is executed immediately without delay.
Call	list_jump_rel(Pos)
Parameter	Pos Jump distance [$(-2^{20}+1)$... $(2^{20}-1)$] as a signed 32-bit value.
Comments	<ul style="list-style-type: none"> • The list_jump_rel command enables implementation of branching (e.g. "if-then-else") independently of the command's list position, in particular also coded independently of the list number because of relative addressing. • The current input list pointer can be queried by get_list_pointer. • list_jump_rel is usable in all list areas, including the protected list buffer ("List 3"). • When specifying a jump distance within "List 1" or "List 2" or for non-indexed subroutines within "List 3" be sure that the jump does not exceed the boundaries of the corresponding memory area. • If the command is used in an indexed subroutine or character set, then also be sure that the jump does not exceed the boundaries of the subroutine or character set. • Illegal jump commands are transmitted unaltered to the RTC5, but are ignored during processing. Instead, the next command is executed. Hence, the user program does probably no longer perform as expected. Therefore, jump commands must be carefully programmed (see also "Jumps", page 89). • Jump commands initiating a jump to themselves (<code>Pos = 0</code>) is also ignored at runtime to prevent an infinite loop that excludes further activities (and that could only be halted by stop_execution or an external list stop). • Decisive are the runtime conditions. When reconfiguring list memory or converting a subroutine, an originally legal jump address might become illegal due to new list boundaries or a relocated subroutine storage position.
RTC4→ RTC5	New command.
Version info	Last change with version OUT 517.
References	list_jump_pos, list_jump_rel_cond



Undelayed Short List Command	list_jump_rel_cond						
Function	<p><i>Conditional (relative) list jump:</i> This command executes the command list_jump_rel(Pos), if the current IOvalue at the EXTENSION 1 socket connector's 16-bit digital input port meets the following condition:</p> $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0})$ <p>(i.e. if the bits specified in Mask1 are 1 and the bits specified in Mask0 are 0). Otherwise, the directly following list command is immediately executed.</p>						
Call	<code>list_jump_rel_cond(Mask1, Mask0, Pos)</code>						
Parameters	<table> <tr> <td>Mask1</td><td>16-bit masks as unsigned 32-bit values.</td></tr> <tr> <td>Mask0</td><td>Only the least significant 16 bits are evaluated.</td></tr> <tr> <td>Pos</td><td>Jump distance $[(-2^{20}+1) \dots (2^{20}-1)]$ as a signed 32-bit value.</td></tr> </table>	Mask1	16-bit masks as unsigned 32-bit values.	Mask0	Only the least significant 16 bits are evaluated.	Pos	Jump distance $[(-2^{20}+1) \dots (2^{20}-1)]$ as a signed 32-bit value.
Mask1	16-bit masks as unsigned 32-bit values.						
Mask0	Only the least significant 16 bits are evaluated.						
Pos	Jump distance $[(-2^{20}+1) \dots (2^{20}-1)]$ as a signed 32-bit value.						
Comments	<ul style="list-style-type: none"> • See list_jump_rel. • Unlike the rules for preventing endless loops (see list_jump_rel), jumps by list_jump_pos_cond are allowed even if they are to their own address (<code>Pos = 0</code>), e.g. to wait for confirmation of a signal. • See also chapter 9.3.2 "Conditional Command Execution", page 244. 						
Examples (Pascal)	<ul style="list-style-type: none"> • Wait until bit #3 of the input port turns HIGH (= loop while the bit is <i>LOW</i>): <code>list_jump_rel_cond(0, \$0008, 0);</code> • Skip the next two list commands if the state of the input port is xxxx xxxx xxxx 0110 : <code>list_jump_rel_cond(6, 9, 3);</code> • See also section "Programming Examples", page 245. 						
RTC4→ RTC5	New command.						
References	list_jump_rel						



Undelayed Short List Command	list_next
Function	Executes the next list command.
Call	<code>list_next()</code>
Comments	<ul style="list-style-type: none"> The command list_next reads the next list command and executes it immediately, as long as the maximum allowed number of short list commands has not been exceeded. Otherwise, it is executed within the next 10 µs clock period. The command is a proper place holder for another command in the list, because it prevents an extra clock, which is unavoidable with other place holders such as list_nop or list_continue.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 541, OUT 541.
References	list_nop, list_continue

Normal List Command	list_nop
Function	Inserts a null operation (no operation) into the list buffer.
Call	<code>list_nop()</code>
Comments	<ul style="list-style-type: none"> The list_nop command works like long_delay(1). It switches off the “laser active” laser control signals after a LaserOff delay and waits for a possible scanner delay. Even if no delays need to be waited for, execution of the command requires 10 µs. The command serves as a placeholder for rejected list commands (get_last_error return code: RTC5_IGNORED). When this command immediately follows a short list command, it ensures that the subsequent list command only executes in the next 10 µs clock period (the short list command’s “effective” execution time is then 10 µs).
RTC4→ RTC5	Essentially unchanged functionality, but switches off the “laser active” laser control signals.
References	long_delay, list_continue



Undelayed Short List Command	list_repeat
Function	Initiates repetition of a group of list commands.
Call	<code>list_repeat()</code>
Comments	<ul style="list-style-type: none"> • See chapter 6.5.5 "Loops", page 91. • Any still-pending delayed short list command executes first. • All list commands between this command and the next list_until command is possibly repeated multiple times. • Nesting up to 8 list_repeat/list_until loops deep is allowed. Each further list_repeat command beyond that limit is ignored as long as no list_until has terminated a loop. • If a list terminates (by set_end_of_list or stop_execution or /STOP), then all not-yet-fully-processed list_repeat/list_until loops are deleted. However, this does not occur if an automatic list change (by auto_change, auto_change_pos or start_loop) is active. • Complete list_repeat/list_until loops can reside within a list or subroutine. Changing between lists and subroutines or between different subroutines is not permitted. Changing between lists is permitted as long as the list change occurs without explicit list termination (by an automatic list change or by an explicit list jump to another list with list_jump_pos). • List jumps into a list_repeat/list_until loop's body or from inside a loop to a loop-external location should be avoided. Careless use could compromise loop management integrity so severely that loops do not execute as expected. But subroutine calls from inside a loop are always reliably possible.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 536, OUT 536.
References	list_until



Undelayed Short List Command	list_return
Function	Terminates a previously called subroutine, jumps to the calling location and executes the next list command (possibly after a list_nop) immediately and without delay.
Call	<code>list_return()</code>
Comments	<ul style="list-style-type: none"> While loading an indexed subroutine, character or text string in the protected memory area, the list_return command additionally triggers entries into the corresponding internal management table of data such as the starting address. In contrast, if list_return directly follows a load_sub, load_char or load_text_table command, then the affected subroutine, character or text string are deleted from the corresponding internal management table. If, during loading into the protected memory area, indexed subroutines, characters or text strings are <i>not</i> terminated with list_return before the input pointer is explicitly set to another position, then they are not stored and are thereafter unavailable. During loading of a list_return command, a flush of the list input buffer is triggered (see page 75). If list_return follows load_sub, load_char or load_text_table, then the input pointer after list_return is invalid. That is, further commands can not be loaded without a prior request for a new input pointer positioning. If, during processing, a list_return is encountered without the prior explicit calling of a subroutine, character or text string, then processing continues at the absolute address 0 (starting address of "List 1"). With nested calls the integrity of the subroutine structure is destroyed.
RTC4→ RTC5	No changes to the previous functionality (termination of a subroutine). New: impact during loading into the protected memory area.
References	list_call , sub_call , load_char , load_text_table , load_sub



Undelayed Short List Command	list_until
Function	Terminates repetition of a group of list commands.
Call	<code>list_until(Number)</code>
Parameter	Number Number of repetitions as an unsigned 32-bit value. Number = 0 is treated like Number = 1.
Comments	<ul style="list-style-type: none"> • See chapter 6.5.5 "Loops", page 91. • Any still-pending delayed short list command executes first. • All list commands between this command and the most recent preceding list_repeat command is executed Number number of times. • Nesting up to 8 list_repeat/list_until loops deep is allowed. Each further list_until command for which there was no preceding list_repeat command is ignored. • An empty loop consisting of list_repeat directly followed by list_until terminates immediately and is not repeated. • If a list terminates (by set_end_of_list or stop_execution or /STOP), then all not-yet-fully-processed list_repeat/list_until loops are deleted. However, this does not occur if an automatic list change (by auto_change, auto_change_pos or start_loop) is active. • Complete list_repeat/list_until loops can reside within a list or subroutine. Changing between lists and subroutines or between different subroutines is not permitted. Changing between lists is permitted as long as the list change occurs without explicit list termination (by an automatic list change or by an explicit list jump to another list with list_jump_pos). • List jumps into a list_repeat/list_until loop's body or from inside a loop to a loop-external location should be avoided. Careless use could compromise loop management integrity so severely that loops do not execute as expected. But subroutine calls from inside a loop are always reliably possible.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 536, OUT 536.
References	list_repeat



Ctrl Command	load_auto_laser_control																				
Function	Loads a table with data points from an ASCII text file and determines – by linear interpolation – the non-linearity curve for position- and/or speed-dependent laser control (see page 163).																				
Call	NoOfDataPoints = load_auto_laser_control(Name, No)																				
Parameters	<table> <tr> <td>Name</td> <td>Name of the text file as a pointer to a null-terminated ANSI string. The text file may contain one or more tables.</td> </tr> <tr> <td>No</td> <td>This parameter (an unsigned 32-bit value) specifies which table in the text file shall be loaded (the parameter corresponds to the extension <No> of the instruction [AutoLaserCtrlTable<No>] at the beginning of the desired table).</td> </tr> </table>	Name	Name of the text file as a pointer to a null-terminated ANSI string. The text file may contain one or more tables.	No	This parameter (an unsigned 32-bit value) specifies which table in the text file shall be loaded (the parameter corresponds to the extension <No> of the instruction [AutoLaserCtrlTable<No>] at the beginning of the desired table).																
Name	Name of the text file as a pointer to a null-terminated ANSI string. The text file may contain one or more tables.																				
No	This parameter (an unsigned 32-bit value) specifies which table in the text file shall be loaded (the parameter corresponds to the extension <No> of the instruction [AutoLaserCtrlTable<No>] at the beginning of the desired table).																				
Result	<p>Signed 32-bit value (a positive error code in case of an error, the negative number of found data points in case of success):</p> <table> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>-1...- 50</td> <td>Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored (see also page 163).</td> </tr> <tr> <td>-1024</td> <td>For Name = 0 (see also comments).</td> </tr> <tr> <td>1</td> <td>No valid data points found (though Table No found).</td> </tr> <tr> <td>3</td> <td>File not found.</td> </tr> <tr> <td>4</td> <td>DSP memory error.</td> </tr> <tr> <td>5</td> <td>BUSY error, board was BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).</td> </tr> <tr> <td>8</td> <td>Board is locked by another user program (get_last_error return code RTC5_ACCESS_DENIED).</td> </tr> <tr> <td>11</td> <td>PCI error (get_last_error return code RTC5_SEND_ERROR), verify error (get_last_error return code RTC5_VERIFY_ERROR).</td> </tr> <tr> <td>13</td> <td>The specified table number was not found in the file.</td> </tr> </tbody> </table>	Value	Description	-1...- 50	Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored (see also page 163).	-1024	For Name = 0 (see also comments).	1	No valid data points found (though Table No found).	3	File not found.	4	DSP memory error.	5	BUSY error, board was BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).	8	Board is locked by another user program (get_last_error return code RTC5_ACCESS_DENIED).	11	PCI error (get_last_error return code RTC5_SEND_ERROR), verify error (get_last_error return code RTC5_VERIFY_ERROR).	13	The specified table number was not found in the file.
Value	Description																				
-1...- 50	Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored (see also page 163).																				
-1024	For Name = 0 (see also comments).																				
1	No valid data points found (though Table No found).																				
3	File not found.																				
4	DSP memory error.																				
5	BUSY error, board was BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).																				
8	Board is locked by another user program (get_last_error return code RTC5_ACCESS_DENIED).																				
11	PCI error (get_last_error return code RTC5_SEND_ERROR), verify error (get_last_error return code RTC5_VERIFY_ERROR).																				
13	The specified table number was not found in the file.																				
Comments	<ul style="list-style-type: none"> The format requirements for the text file's table entries with data points for the nonlinearity curve are described in section "Notes on Loading and Determining Nonlinearity Curves", page 163. When loading the table, the RTC5 determines suitable values for the entire range of percent values. The load_auto_laser_control overwrites any previously loaded nonlinearity curve. For Name = 0 (as during initialization by load_program_file), the function Scale(Percent)=1.0 is loaded for the complete percent range (no nonlinearity). The load_auto_laser_control command is <i>not</i> executed (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list) or the board's INTERNAL-BUSY status is currently set. In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set). During execution of load_auto_laser_control, external starts are suppressed. Before loading a table, load_auto_laser_control performs a DSP memory check. In case of an error, error code 4 is returned. 																				



Ctrl Command	load_auto_laser_control
RTC4→ RTC5	New command.
References	set_auto_laser_control

Ctrl Command	load_char
Function	Assigns a desired index to a character defined by subsequent list commands, and loads the character into the protected buffer area ("List 3").
Call	<code>load_char(Char)</code>
Parameter	Char Index of the indexed character as an unsigned 32-bit value. Allowed range [0 ... 1023].
Comments	<ul style="list-style-type: none"> Up to 1024 indexed characters, hence 4 character sets with 256 indexed characters per set, can be stored. For defining character sets, the following applies: Char = character set number * 256 + ASCII number of the character (character sets are numbered 0 to 3). If Char > 1023 then the load_char command is ignored (get_last_error return code RTC5_PARAM_ERROR). The addresses in the protected buffer area where the character definitions are to be stored are automatically determined and internally managed. This management is independent of that for indexed subroutines (see load_sub) and text string definitions (see load_text_table). Indexed character definitions must be terminated with a list_return command. This is a prerequisite for actual storage of the commands, entry of the start address and other information (e.g., the number of commands) into the internal management table, and initiating a flush of the list input buffer (see page 75). Otherwise (the input pointer is altered without a preceding list_return command) the character definition with this index is not available. An indexed character definition is not stored if the protected buffer area ("List 3") was not previously configured for a sufficient size beyond "List 1" and "List 2". If list_return is the next command after load_char, then the corresponding character definition is deleted from the internal management table. Observe all notes in chapter 6.5.2 "Character Sets and Text Strings", page 87.
RTC4→ RTC5	New command.
References	list_return , load_sub , load_text_table



Ctrl Command	load_correction_file																												
Function	Loads the specified image field correction file into RTC5 memory (as table #1, 2,3 or 4) and automatically calls select_cor_table with the most recently used parameter values (or the default parameter values).																												
Call	ErrorNo = load_correction_file(Name, No, Dim)																												
Parameters	<p>Name Name of the correction file as a pointer to a null-terminated ANSI string.</p> <p>No This number (unsigned 32-bit value) determines whether the file shall be stored as correction table #1, 2, 3 or 4. Allowed values: [1...4].</p> <p>Dim This parameter (an unsigned 32-bit value) determines whether the file should be stored as a 2D correction table or (if possible) as a 3D correction table (see also comments). = 2: 2D and 3D correction files are stored as a 2D correction table (downgraded when necessary). = 3: If the 3D option <i>is</i> enabled, 2D and 3D correction files are stored as a 3D correction table (upgraded if necessary). If the 3D option <i>is not</i> enabled (default), then the command executes with Dim = 2. Otherwise, the parameter has no effect. A 2D correction file is stored as a 2D correction table. A 3D correction file is stored as a 3D correction table if the 3D option <i>is</i> enabled and as a 2D correction table (downgraded) if the 3D option <i>is not</i> enabled.</p>																												
Result	<p>Error code as an unsigned 32-bit value:</p> <table> <tr> <td>Value</td> <td>Description</td> </tr> <tr> <td>0</td> <td>Success.</td> </tr> <tr> <td>1</td> <td>File error (file corrupt or incomplete).</td> </tr> <tr> <td>2</td> <td>Memory error (DLL-internal, Windows system memory).</td> </tr> <tr> <td>3</td> <td>File-open error (empty string submitted for Name parameter, file not found, etc.).</td> </tr> <tr> <td>4</td> <td>DSP memory error.</td> </tr> <tr> <td>5</td> <td>PCI download error (driver error).</td> </tr> <tr> <td>8</td> <td>System driver not found (get_last_error return code: RTC5_ACCESS_DENIED).</td> </tr> <tr> <td>10</td> <td>Parameter error (incorrect No).</td> </tr> <tr> <td>11</td> <td>Access error: board reserved for another user program (get_last_error return code: RTC5_ACCESS_DENIED) or version check has detected an error (OUT or RTC version not compatible to the current DLL version, get_last_error return code: RTC5_ACCESS_DENIED RTC5_VERSION_MISMATCH).</td> </tr> <tr> <td>12</td> <td>Warning: 3D table or Dim = 3 selected, but the 3D option is not enabled. The system subsequently operates as an ordinary 2D system (this warning is only returned, if no other error has occurred).</td> </tr> <tr> <td>13</td> <td>Busy error: no download, board is BUSY or INTERNAL-BUSY (get_last_error return code: RTC5_BUSY).</td> </tr> <tr> <td>14</td> <td>PCI upload error (driver error, only applicable for download verification)</td> </tr> <tr> <td>15</td> <td>Verify error (only applicable for download verification).</td> </tr> </table>	Value	Description	0	Success.	1	File error (file corrupt or incomplete).	2	Memory error (DLL-internal, Windows system memory).	3	File-open error (empty string submitted for Name parameter, file not found, etc.).	4	DSP memory error.	5	PCI download error (driver error).	8	System driver not found (get_last_error return code: RTC5_ACCESS_DENIED).	10	Parameter error (incorrect No).	11	Access error: board reserved for another user program (get_last_error return code: RTC5_ACCESS_DENIED) or version check has detected an error (OUT or RTC version not compatible to the current DLL version, get_last_error return code: RTC5_ACCESS_DENIED RTC5_VERSION_MISMATCH).	12	Warning: 3D table or Dim = 3 selected, but the 3D option is not enabled. The system subsequently operates as an ordinary 2D system (this warning is only returned, if no other error has occurred).	13	Busy error: no download, board is BUSY or INTERNAL-BUSY (get_last_error return code: RTC5_BUSY).	14	PCI upload error (driver error, only applicable for download verification)	15	Verify error (only applicable for download verification).
Value	Description																												
0	Success.																												
1	File error (file corrupt or incomplete).																												
2	Memory error (DLL-internal, Windows system memory).																												
3	File-open error (empty string submitted for Name parameter, file not found, etc.).																												
4	DSP memory error.																												
5	PCI download error (driver error).																												
8	System driver not found (get_last_error return code: RTC5_ACCESS_DENIED).																												
10	Parameter error (incorrect No).																												
11	Access error: board reserved for another user program (get_last_error return code: RTC5_ACCESS_DENIED) or version check has detected an error (OUT or RTC version not compatible to the current DLL version, get_last_error return code: RTC5_ACCESS_DENIED RTC5_VERSION_MISMATCH).																												
12	Warning: 3D table or Dim = 3 selected, but the 3D option is not enabled. The system subsequently operates as an ordinary 2D system (this warning is only returned, if no other error has occurred).																												
13	Busy error: no download, board is BUSY or INTERNAL-BUSY (get_last_error return code: RTC5_BUSY).																												
14	PCI upload error (driver error, only applicable for download verification)																												
15	Verify error (only applicable for download verification).																												



Ctrl Command	load_correction_file
Comments	<p>Notes on loading correction tables:</p> <ul style="list-style-type: none"> The RTC5 can store four different correction tables at the same time, e.g. for use in a multiple scan head configuration. Correction tables number 3 and 4 must always be loaded only after load_program_file. These tables occupy the upper half of the memory area reserved for data recording by set_trigger or set_trigger4 (see comments for those commands). load_program_file automatically initializes this memory area and any already loaded correction tables number 3 and 4 are lost. Before storing a correction file, load_correction_file performs a DSP memory check. In case of an error, error code 4 is returned. If the parameter No is out of range, then no correction table is loaded (return value 10). The name of the to-be-loaded correction file must be passed to load_correction_file as a pointer to a null-terminated ANSI string. If Name is passed as a null pointer (0), the corresponding table is replaced by a 1-to-1 table (for Dim = 3 and enabled 3D option with a 1-to-1 3D table, otherwise with a 1-to-1 2D table). Most of the above-mentioned parameters are then set to 0. An empty string ("") for Name result in an error return code of 3. If the 3D option is <i>not</i> enabled (default), then 2D and 3D correction files are stored as 2D correction tables – regardless of the value specified for Dim (3D correction files also include 2D correction tables). The 3D data sections of 3D correction files are then ignored. If, on the other hand, the 3D option is <i>enabled</i>, then both 2D and 3D tables can be loaded. <ul style="list-style-type: none"> For Dim = 2, a 2D table is always stored (the 3D data section of 3D correction files are ignored) and, accordingly, only 2D corrections are calculated (the Z axis thereby remains unchanged, as if no 3D option was enabled). For Dim = 3, if the 3D option is enabled then both 2D and 3D correction files are stored as 3D correction tables. 2D correction tables are thereby automatically expanded to incorporate a linear Z correction. The actually suitable Z correction can subsequently be loaded by the load_z_table command (see page 196). All other values for Dim do not change the type of the correction file.



Ctrl Command	load_correction_file
Comments (cont'd)	<p>Notes on assigning correction tables by <code>select_cor_table</code>:</p> <ul style="list-style-type: none"> • Use the <code>select_cor_table</code> or <code>select_cor_table_list</code> command to assign one (or two) correction table(s) stored on the RTC5 to the scan head (or to both scan head connectors). • As of Version DLL 521, OUT 521, the command <code>load_correction_file</code> automatically calls <code>select_cor_table</code> after loading of a correction table. However, if you call <code>load_correction_file</code> before loading the program file (by <code>load_program_file</code>), then the automatic call of <code>select_cor_table</code> has no effect. If you call <code>load_correction_file</code> after <code>load_program_file</code>, then the <code>select_cor_table</code> call uses the parameter values (HeadA = 1, HeadB = 0) or the values most recently used (after <code>load_program_file</code>) when having called <code>select_cor_table</code>. As of version DLL 527, OUT 529, the command only returns to the user program, after the <code>select_cor_table</code>-induced jump to the corrected galvanometer position has completed. • With older versions (prior to DLL 521, OUT 521), you must absolutely call <code>select_cor_table</code> or <code>select_cor_table_list</code> after <code>load_correction_file</code> or <code>load_program_file</code>. • For further information on this subject, see notes on page 138. <p>Other notes:</p> <ul style="list-style-type: none"> • RTC5 correction tables contain parameters that formerly (with an RTC4/RTC3/RTC2) had to be manually copied into a user program from a supplied ReadMe file. To directly integrate these parameters into the user program, the RTC5 can load them by <code>get_table_para</code> from the currently loaded correction table or read them by <code>get_head_para</code> from the assigned correction table. • During execution of <code>load_correction_file</code>, no other control commands can be executed; external starts are disabled. • The <code>load_correction_file</code> command is ignored (<code>get_last_error</code> return code: <code>RTC5_BUSY</code>) if the board's BUSY status is currently set (list is being processed or has been halted by <code>pause_list</code>) or the board's INTERNAL-BUSY status is currently set. In contrast, the command is executed when a list has been paused by <code>set_wait</code> (PAUSED status set). But for older versions (prior to DLL 521, OUT 521), you should absolutely call <code>select_cor_table</code> before restarting a list. This ensures a smooth transition by <code>jump_speed</code> from the old galvanometer positions to the new ones. Otherwise, hard jumps can occur (see also notes on page 138). • If an <code>RTC5_VERSION_MISMATCH</code> error occurs (return value 11), a DLL version appropriate for the program file must be chosen and the board must be made currentless (to unload the program software) or alternatively program files appropriate for the DLL version must be reloaded by the multi-board command <code>n_load_program_file</code> (after <code>RTC5_ACCESS_DENIED</code>, the single-board command <code>load_program_file</code> does not get access rights for the board). Only afterward (and after the board has been accessed by <code>acquire_RTC</code> or <code>select_RTC</code> and possibly specified as default board by <code>select_RTC</code>) <code>load_correction_file</code> can be normally executed again.



Ctrl Command	load_correction_file
RTC4→ RTC5	<p>Other than its primary purpose and its first two parameters (loading of correction files), the command's functionality has substantially changed:</p> <ul style="list-style-type: none"> The command now uses the new parameter <code>Dim</code>. This allows, for instance, storage of 3D correction files as 2D correction tables and storage of 2D correction files as 3D correction tables. Now, two 3D correction tables can be stored in RTC5 memory (in the 3D-enabled version). However, two 3D correction tables can <i>not</i> be simultaneously assigned to the scan head control ports (see select_cor_table). The parameters <code>k</code>, <code>Phi</code> and <code>Offset</code> no longer exist. Therefore, table transformations (scaling, rotation, translation) are no longer possible during loading of the correction file. Such coordinate transformations can now be specified by <code>set_scale</code>, <code>set_angle</code> and <code>set_offset</code> in addition to <code>set_matrix</code>. As of version DLL 521, OUT 521, select_cor_table is automatically called (see comments above and notes on page 138). As of version DLL 527, OUT 529, the command only returns after the correction motion has completed (see comments above). As of version DLL 533, OUT 534 up to four correction files can be loaded (see comments above)
Version info	Last change with version DLL 533, OUT 534 (see above).



Ctrl Command	load_disk
Function	Loads into the protected memory area ("List 3") the indexed characters, text strings and subroutines previously stored in a binary file by save_disk and returns the number of actually loaded list commands.
Call	NoOfLoadedCommands = load_disk(Name, Mode)
Parameter	<p>Name File name as a pointer to a null-terminated ANSI string.</p> <p>Mode This parameter (unsigned 32-bit value) specifies how the loading procedure is executed.</p> <p>=0: The internal management tables for indexed characters, text strings and subroutines are initialized (all old references are thereby lost) and the input pointer is set to the beginning of "List 3" (the resulting loading process overwrites list commands stored there).</p> <p>>0: Internal management tables are not initialized (all old references are initially retained, but then replaced or supplemented by other references during the loading process, depending on the file's content) and the input pointer is set to the position after the last stored (by load_char, load_text_table or load_sub) indexed character, text string or subroutine (the resulting loading process does <i>not</i> overwrite the list commands of old indexed characters, text strings and subroutines).</p>
Result	The number of commands loaded by load_disk , as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> For <code>Name</code> = 0, only the internal management tables are initialized as with <code>Mode</code> = 0 (no loading occurs, no "empty" binary file must be provided). Otherwise, the load_disk command can only be used for reading files that were stored with save_disk. For all indexed characters, text strings and subroutines in the specified file, load_disk executes a corresponding load_char, load_text_table or load_sub command. The corresponding list commands are thereby written to the protected memory area and appropriate entries are made in the internal management tables in accordance with the index assignment stored in the file. If there is no character, text string or subroutine for a particular index in the specified file, then no list commands are loaded for this index, and if <code>Mode</code> > 0 then any already existing entries in the internal management table remains unaltered. Thus, supplementary loading of indexed characters, text strings and subroutines from various files is possible. Together with the save_disk command, load_disk can be used, for example, to defragment the protected memory area and to subsequently protect subroutines (see page 84 and page 85). If the characters, text strings and subroutines come from various files, then defragmentation can be achieved if the first file is loaded in <code>Mode</code> = 0 and subsequent files are loaded with <code>Mode</code> > 0, provided that no indices are used simultaneously in different files. Memory gaps in the protected area caused by dereferencing of indexed characters, text strings or subroutines are thereby closed.



Ctrl Command	load_disk
Comments (cont'd)	<ul style="list-style-type: none"> If, during loading, the end of the protected memory area is reached before the end of the file (EOF), then all further list commands in the file are ignored. Likewise, incomplete characters, text strings and subroutines (as with individual load_char, load_text_table or load_sub commands) are not stored. Before each load_disk command, be sure that the memory configuration provides a sufficiently large protected area above the list areas ("List 1" and "List 2"). The save_disk command returns the number of list commands stored in the file. If a board changes ownership, it is the user's responsibility to ensure that the memory configuration data is consistent (<code>Mem1</code> and <code>Mem2</code> are queried from the DLL, not from the board – see also get_config_list and page 96). If the specified file is corrupt and cannot be read to the end, then only the characters, text strings and subroutines fully readable to that point are stored. The command is ignored (get_last_error return code: <code>RTC5_PARAM_ERROR</code>) if "List 3" has not been assigned a memory area ($\text{Mem1} + \text{Mem2} = 2^{20}$). The command is likewise ignored (get_last_error return code: <code>RTC5_BUSY</code>) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list) or the board's INTERNAL-BUSY status is currently set. In contrast, if a list has only been paused by set_wait (PAUSED status set), then load_disk can be executed. But users are responsible for ensuring that no still-needed commands are overwritten, e.g. if set_wait was called from an indexed subroutine. While the load_disk command is running, no other commands can execute. During execution of the command, external starts are suppressed.
RTC4→ RTC5	New command.
References	save_disk



Ctrl Command	load_fly_2d_table																						
Function	Loads a 2D table from an ASCII text file for a set_fly_2d -Processing-on-the-fly application with 2D encoder compensation for XY stages, see section "2D Encoder Compensation for XY Stages", page 206 .																						
Call	NoOfDataPoints = load_fly_2d_table(Name, No)																						
Parameters	<table> <tr> <td>Name</td> <td>Name of the text file as a pointer to a null-terminated ANSI string (the text file may contain one or more tables).</td> </tr> <tr> <td>No</td> <td>This parameter (an unsigned 32-bit value) specifies which table in the text file shall be loaded (the parameter corresponds to the extension <No> of the instruction [Fly2DTable<No>] at the beginning of the desired table).</td> </tr> </table>	Name	Name of the text file as a pointer to a null-terminated ANSI string (the text file may contain one or more tables).	No	This parameter (an unsigned 32-bit value) specifies which table in the text file shall be loaded (the parameter corresponds to the extension <No> of the instruction [Fly2DTable<No>] at the beginning of the desired table).																		
Name	Name of the text file as a pointer to a null-terminated ANSI string (the text file may contain one or more tables).																						
No	This parameter (an unsigned 32-bit value) specifies which table in the text file shall be loaded (the parameter corresponds to the extension <No> of the instruction [Fly2DTable<No>] at the beginning of the desired table).																						
Result	<p>Signed 32-bit value (a positive error code in case of an error, the negative number of found data points in case of success).</p> <table> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>< 0</td> <td>Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored (see page 207).</td> </tr> <tr> <td>0</td> <td>For Name = 0 (see comments).</td> </tr> <tr> <td>1</td> <td>No valid data points found (though Table No found).</td> </tr> <tr> <td>2</td> <td>Out of Memory (not enough Windows system memory).</td> </tr> <tr> <td>3</td> <td>File not found.</td> </tr> <tr> <td>4</td> <td>DSP memory error.</td> </tr> <tr> <td>5</td> <td>BUSY error, board is BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).</td> </tr> <tr> <td>8</td> <td>Board is locked by another user program (get_last_error return code RTC5_ACCESS_DENIED).</td> </tr> <tr> <td>11</td> <td>PCI error (get_last_error return code RTC5_SEND_ERROR), verify error (get_last_error return code RTC5_VERIFY_ERROR).</td> </tr> <tr> <td>13</td> <td>The specified table number was not found in the file.</td> </tr> </tbody> </table>	Value	Description	< 0	Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored (see page 207).	0	For Name = 0 (see comments).	1	No valid data points found (though Table No found).	2	Out of Memory (not enough Windows system memory).	3	File not found.	4	DSP memory error.	5	BUSY error, board is BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).	8	Board is locked by another user program (get_last_error return code RTC5_ACCESS_DENIED).	11	PCI error (get_last_error return code RTC5_SEND_ERROR), verify error (get_last_error return code RTC5_VERIFY_ERROR).	13	The specified table number was not found in the file.
Value	Description																						
< 0	Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored (see page 207).																						
0	For Name = 0 (see comments).																						
1	No valid data points found (though Table No found).																						
2	Out of Memory (not enough Windows system memory).																						
3	File not found.																						
4	DSP memory error.																						
5	BUSY error, board is BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).																						
8	Board is locked by another user program (get_last_error return code RTC5_ACCESS_DENIED).																						
11	PCI error (get_last_error return code RTC5_SEND_ERROR), verify error (get_last_error return code RTC5_VERIFY_ERROR).																						
13	The specified table number was not found in the file.																						
Comments	<ul style="list-style-type: none"> The text file's data format requirements for reference points of the 2D encoder compensation table are described on page 207. The largest of these reference points should not exceed the range -524288 to 524287 (otherwise precision may be lost). During runtime, the current encoder values (including reference values) must not exceed the largest values specified in the table. Otherwise clipping occurs. The command load_fly_2d_table overwrites any previously loaded table for 2D encoder compensation. If Name = 0, then a 0-correction table for 2D encoder compensation is loaded. The command is <i>not</i> executed (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list) or the board's INTERNAL-BUSY status is currently set. In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set). During execution of load_fly_2d_table, external starts are suppressed. Before loading a table, load_fly_2d_table performs a DSP memory check. In case of an error, error code 4 is returned. 																						



Ctrl Command	load_fly_2d_table
RTC4→ RTC5	New command.
Version info	Available as of version DLL 536, OUT 536.
References	set_fly_2d , init_fly_2d , get_fly_2d_offset

Control Command	load_jump_table												
Function	Loads a value table with jump delay data points from an ASCII text file (or alternatively performs automatic determination on the attached scan system) and uses linear interpolation to create the internal jump delay table for 2D jumps executable in jump mode.												
Call	NoOfDataPoints = load_jump_table(Name, No, PosAck, MinDelay, MaxDelay, ListPos)												
Parameters	<table> <tr> <td>Name</td> <td>See load_jump_table_offset.</td> </tr> <tr> <td>No</td> <td>See load_jump_table_offset.</td> </tr> <tr> <td>PosAck</td> <td>See load_jump_table_offset.</td> </tr> <tr> <td>MinDelay</td> <td>See load_jump_table_offset.</td> </tr> <tr> <td>MaxDelay</td> <td>See load_jump_table_offset.</td> </tr> <tr> <td>ListPos</td> <td>See load_jump_table_offset.</td> </tr> </table>	Name	See load_jump_table_offset .	No	See load_jump_table_offset .	PosAck	See load_jump_table_offset .	MinDelay	See load_jump_table_offset .	MaxDelay	See load_jump_table_offset .	ListPos	See load_jump_table_offset .
Name	See load_jump_table_offset .												
No	See load_jump_table_offset .												
PosAck	See load_jump_table_offset .												
MinDelay	See load_jump_table_offset .												
MaxDelay	See load_jump_table_offset .												
ListPos	See load_jump_table_offset .												
Result	See load_jump_table_offset .												
Notes	<ul style="list-style-type: none"> The command is identical to load_jump_table_offset with Offset = 0. 												
Version info	Available as of version DLL 525.												
RTC4→ RTC5	New command.												
Reference	load_jump_table_offset												

Control Command	load_jump_table_offset														
Function	Loads a value table with jump delay data points from an ASCII text file (or alternatively performs automatic determination on the attached scan system) and uses linear interpolation to create the internal jump delay table for 2D jumps executable in jump mode.														
Call	NoOfDataPoints = load_jump_table_offset(Name, No, PosAck, Offset, MinDelay, MaxDelay, ListPos)														
Parameters	<table> <tr> <td>Name</td> <td>Name of the text file as a pointer to a null-terminated ANSI string (the text file can contain one or several tables) or NULL for automatic determination.</td> </tr> <tr> <td>No</td> <td>This parameter (unsigned 32-bit value) specifies: <ul style="list-style-type: none"> For Name = Filename, which table of the text file should be loaded (the parameter corresponds to the suffix <No> of the statement [JumpTable<No>] at the beginning of the desired table). For Name = 0, which scan system (or which scan head connector) should be used for automatic determination. Allowed values: [1, 2]. </td> </tr> <tr> <td>PosAck</td> <td>Position tolerance value in [bits] as an unsigned 32-bit number (only relevant for automatic determination).</td> </tr> <tr> <td>Offset</td> <td>Offset in [10 µs] as a signed 32-bit number, which is added to all automatically determined delay values (only relevant for automatic determination).</td> </tr> <tr> <td>MinDelay</td> <td>Minimum jump delay in [10 µs] as an unsigned 32-bit value (only relevant for automatic determination).</td> </tr> <tr> <td>MaxDelay</td> <td>Maximum jump delay in [10 µs] as an unsigned 32-bit value (only relevant for automatic determination).</td> </tr> <tr> <td>ListPos</td> <td>List position (in the area of list 1 or 2) for six list commands that can be overwritten (only relevant for automatic determination).</td> </tr> </table>	Name	Name of the text file as a pointer to a null-terminated ANSI string (the text file can contain one or several tables) or NULL for automatic determination.	No	This parameter (unsigned 32-bit value) specifies: <ul style="list-style-type: none"> For Name = Filename, which table of the text file should be loaded (the parameter corresponds to the suffix <No> of the statement [JumpTable<No>] at the beginning of the desired table). For Name = 0, which scan system (or which scan head connector) should be used for automatic determination. Allowed values: [1, 2]. 	PosAck	Position tolerance value in [bits] as an unsigned 32-bit number (only relevant for automatic determination).	Offset	Offset in [10 µs] as a signed 32-bit number, which is added to all automatically determined delay values (only relevant for automatic determination).	MinDelay	Minimum jump delay in [10 µs] as an unsigned 32-bit value (only relevant for automatic determination).	MaxDelay	Maximum jump delay in [10 µs] as an unsigned 32-bit value (only relevant for automatic determination).	ListPos	List position (in the area of list 1 or 2) for six list commands that can be overwritten (only relevant for automatic determination).
Name	Name of the text file as a pointer to a null-terminated ANSI string (the text file can contain one or several tables) or NULL for automatic determination.														
No	This parameter (unsigned 32-bit value) specifies: <ul style="list-style-type: none"> For Name = Filename, which table of the text file should be loaded (the parameter corresponds to the suffix <No> of the statement [JumpTable<No>] at the beginning of the desired table). For Name = 0, which scan system (or which scan head connector) should be used for automatic determination. Allowed values: [1, 2]. 														
PosAck	Position tolerance value in [bits] as an unsigned 32-bit number (only relevant for automatic determination).														
Offset	Offset in [10 µs] as a signed 32-bit number, which is added to all automatically determined delay values (only relevant for automatic determination).														
MinDelay	Minimum jump delay in [10 µs] as an unsigned 32-bit value (only relevant for automatic determination).														
MaxDelay	Maximum jump delay in [10 µs] as an unsigned 32-bit value (only relevant for automatic determination).														
ListPos	List position (in the area of list 1 or 2) for six list commands that can be overwritten (only relevant for automatic determination).														
Result	<p>Error code as an unsigned 32-bit value.</p> <ul style="list-style-type: none"> -1...- 50 Success for Name = filename. The absolute value of the return value equals the number of valid data points found in the table. Invalid entry values are ignored (see also page 179). -1024 Success for Name = 0: Table was automatically determined. 1 No valid data points found (but Table No found). 3 File not found. 4 Verify error: DSP memory error. 5 Busy error, board is BUSY or INTERNAL BUSY, no download (get_last_error return code RTC5_BUSY). 8 Access error: board reserved for another user program (get_last_error return code: RTC5_ACCESS_DENIED). 10 only if Name = 0: Param error: HeadNo or ListPos invalid. 11 PCI error during download (get_last_error return code RTC5_SEND_ERROR). 13 The supplied table number could not be found in the file. 														



Control Command	load_jump_table_offset
Comments	<ul style="list-style-type: none"> For information on command usage, see "Jump-Length-Dependent Jump Delays", page 178. For jump mode information, see page 176. Format requirements for placing the table with jump delay data points into the text file are described in the section "Notes on Loading Determined Jump Delay Values" on page 179. When loading the table, the RTC5 uses linear interpolation to establish appropriate values for the complete jump length range. The command load_jump_table_offset overwrites any previously loaded jump delay tables for jump mode. If Name = filename, then the table number No is loaded. The other parameters are not used. If Name = 0, then the jump delay table for head No is automatically determined and loaded (if jump mode has been previously enabled and activated successfully by set_jump_mode (Flag = 1)). Here, the parameters PosAck, Offset, MinDelay, MaxDelay and ListPos are used (see "Automatic Determination of the Jump Delay Table", page 180). Jump mode takes effect upon the next 2D jump only if it was activated and enabled by set_jump_mode or activated by set_jump_mode_list. The command load_jump_table_offset does not execute (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or was paused by pause_list) or if the board's INTERNAL BUSY status is set. In contrast, the command executes if a list was paused by set_wait (PAUSED status set). During the command's execution, external starts are suppressed. Before loading a table, load_jump_table_offset performs a DSP memory check that produces error code 4 in case of error.
RTC4→ RTC5	New command. There is no RTC4 compatibility mode for this command.
Version info	Available as of version DLL 526.
Reference	load_jump_table , set_jump_mode , set_jump_mode_list , get_jump_table , set_jump_table



Ctrl Command	load_list
Function	Opens the list buffer for writing with list commands and sets the input pointer to the specified (relative) position within the desired list ("List 1" or "List 2"), but only if the list is not currently being processed (BUSY status not set) or has already been processed (USED status set). In case of success, the next list command is stored at this address and all further commands at subsequent addresses in the selected list.
Call	NoOfOpenedList = load_list(ListNo, Pos)
Parameters	<p>ListNo Number of the list in which the input pointer should be set, as an unsigned 32-bit value. Allowed values: [0 ... 3]. Only the two least significant bits are evaluated.</p> <ul style="list-style-type: none"> =1: "List 1" is opened if not BUSY (BUSY1 status not set). =2: "List 2" is opened if not BUSY (BUSY2 status not set). =0: The list that is currently not BUSY is opened. If both lists are not BUSY, then "List 1" is opened. =3: The list that is currently not BUSY but USED (USED status set) is opened. If both lists are not BUSY and both are USED, then that list is opened, which would be executed by a following automatic list change. <p>For Mem2 = 0 (see config_list) "List 1" is opened if not BUSY (ListNo = 0...2) or if not BUSY but USED (ListNo = 3).</p> <p>Pos Position of the input pointer (offset relative to the start of the respective list) as an unsigned 32-bit value. Allowed range: [0 ... (2^{30}-1)].</p>
Result	Number of the opened list [1 or 2] if successful, otherwise 0 (as an unsigned 32-bit value).
Comments	<ul style="list-style-type: none"> • The load_list command (ListNo = 3) is useful in scenarios such as alternating list changes, where you want to wait specifically for a list to be processed (see "Alternating List Changes", page 80) without needing to separately query the list status. Unintentional overwriting of not-yet-executed commands is thereby automatically avoided. To instead perform <i>unconditional</i> loading, you can use commands such as set_start_list_pos. • After a successful check of the list number (BUSY status not set and, if applicable, USED status set), the load_list command behaves like set_start_list_pos (see comments there). • If the load_list command was not successful (return code 0), then no list is opened and the input pointer is set to an invalid position. Then no further list commands can be input until the input pointer is correctly set (e.g. by repeating the load_list command with a positive result or by the set_start_list_pos command etc.). It is up to users to react to a return code of 0. The load_list command produces no wait loop and does not block execution of subsequent control commands.



Ctrl Command	load_list
Comments (cont'd)	<ul style="list-style-type: none"> If <code>ListNo = 0...2</code> when using the load_list command, then note that a list's BUSY status can change between opening of the list with load_list and the actual loading of list commands, e.g. if in the meantime an automatic list change has occurred that was previously defined by auto_change or start_loop. In cases such as this, where loading of a list might occur simultaneously with processing of the same list, users must always ensure that the output pointer does not overtake the input pointer. In contrast, for <code>ListNo = 3</code> load_list ensures that a list is actually opened for loading only directly after processing (and after any successful automatic list changes). Particularly, if no list is BUSY and both lists are USED (e.g. after initialization, after stop_execution or after an external list stop), the opened list number is synchronized with the following automatic list change.
RTC4→ RTC5	New command.
References	set_start_list_pos

Ctrl Command	load_position_control																				
Function	Loads a table with data points from an ASCII text file and determines – by linear interpolation – the scaling function for position-dependent laser control (radial correction, see section "Position-Dependent Laser Control", page 161).																				
Call	NoOfDataPoints = load_position_control(Name, No)																				
Parameters	<table> <tr> <td>Name</td> <td>Name of the text file as a pointer to a null-terminated ANSI string (the text file may contain one or more tables).</td> </tr> <tr> <td>No</td> <td>This parameter (an unsigned 32-bit value) specifies which table in the text file shall be loaded (the parameter corresponds to the extension <No> of the instruction [PositionCtrlTable<No>] at the beginning of the desired table).</td> </tr> </table>	Name	Name of the text file as a pointer to a null-terminated ANSI string (the text file may contain one or more tables).	No	This parameter (an unsigned 32-bit value) specifies which table in the text file shall be loaded (the parameter corresponds to the extension <No> of the instruction [PositionCtrlTable<No>] at the beginning of the desired table).																
Name	Name of the text file as a pointer to a null-terminated ANSI string (the text file may contain one or more tables).																				
No	This parameter (an unsigned 32-bit value) specifies which table in the text file shall be loaded (the parameter corresponds to the extension <No> of the instruction [PositionCtrlTable<No>] at the beginning of the desired table).																				
Result	<p>Signed 32-bit value (a positive error code in case of an error, the negative number of found data points in case of success):</p> <table> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>-1...- 50</td> <td>Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored (see also section "Notes on Loading a Scaling Function", page 162).</td> </tr> <tr> <td>-256</td> <td>For Name = 0 (see also comments).</td> </tr> <tr> <td>1</td> <td>No valid data points found (though Table No found).</td> </tr> <tr> <td>3</td> <td>File not found.</td> </tr> <tr> <td>4</td> <td>DSP memory error.</td> </tr> <tr> <td>5</td> <td>BUSY error, board was BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).</td> </tr> <tr> <td>8</td> <td>Board is locked by another user program (get_last_error return code RTC5_ACCESS_DENIED).</td> </tr> <tr> <td>11</td> <td>PCI error (get_last_error return code RTC5_SEND_ERROR), verify error (get_last_error return code RTC5_VERIFY_ERROR).</td> </tr> <tr> <td>13</td> <td>The specified table number was not found in the file.</td> </tr> </tbody> </table>	Value	Description	-1...- 50	Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored (see also section "Notes on Loading a Scaling Function", page 162).	-256	For Name = 0 (see also comments).	1	No valid data points found (though Table No found).	3	File not found.	4	DSP memory error.	5	BUSY error, board was BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).	8	Board is locked by another user program (get_last_error return code RTC5_ACCESS_DENIED).	11	PCI error (get_last_error return code RTC5_SEND_ERROR), verify error (get_last_error return code RTC5_VERIFY_ERROR).	13	The specified table number was not found in the file.
Value	Description																				
-1...- 50	Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored (see also section "Notes on Loading a Scaling Function", page 162).																				
-256	For Name = 0 (see also comments).																				
1	No valid data points found (though Table No found).																				
3	File not found.																				
4	DSP memory error.																				
5	BUSY error, board was BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).																				
8	Board is locked by another user program (get_last_error return code RTC5_ACCESS_DENIED).																				
11	PCI error (get_last_error return code RTC5_SEND_ERROR), verify error (get_last_error return code RTC5_VERIFY_ERROR).																				
13	The specified table number was not found in the file.																				
Comments	<ul style="list-style-type: none"> The format requirements for the text file's table entries with data points for position-dependent laser control are described in "Notes on Loading a Scaling Function", Seite 162. When loading the table, the RTC5 determines suitable values for the entire range of control values. The command load_position_control overwrites any previously loaded scaling function for position-dependent laser control. For Name = 0 (as during initialization by load_program_file), the scaling function $Scale(Position)=1.0$ is loaded for the complete position range so that no position-dependent correction takes place. Position-dependent laser control only takes effect during subsequent mark commands or arc commands if it was initialized by set_auto_laser_control. Position-dependent laser control is deactivated by set_auto_laser_control (Ctrl = 0) or by loading $Scale(Position)=1.0$. See also "Position-Dependent Laser Control", page 161. 																				



Ctrl Command	load_position_control
Comments (cont'd)	<ul style="list-style-type: none"> The load_position_control command is <i>not</i> executed (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list) or the board's INTERNAL-BUSY status is currently set. In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set). During execution of load_position_control, external starts are suppressed. Before loading a table, load_position_control performs a DSP memory check. In case of an error, error code 4 is returned.
RTC4→ RTC5	New command.
References	set_auto_laser_control



Ctrl Command	load_program_file																																					
Function	Executes a board reset, performs a DSP memory check, loads into RTC5 memory the program file RTC5OUT.out along with the files RTC5RBF.rbf and RTC5DAT.dat from the specified directory, performs a version check and starts the signal processor (DSP).																																					
Call	ErrorNo = load_program_file(pPath)																																					
Parameter	pPath Path name of the directory, where the files RTC5OUT.out, RTC5RBF.rbf and RTC5DAT.dat are stored (as a pointer to a null-terminated string).																																					
Result	<p>Error code as an unsigned 32-bit value.</p> <table> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>Success.</td> </tr> <tr> <td>1</td> <td>Reset error: the board could not be reset.</td> </tr> <tr> <td>2</td> <td>Unreset error: the board does not restart.</td> </tr> <tr> <td>3</td> <td>File error: file not found or cannot be opened.</td> </tr> <tr> <td>4</td> <td>Format error: RTC5OUT.out has incorrect format. PCI error: driver reported an error during PCI transfer because driver is not compatible with DLL. This occurs particularly when DLL 533 or earlier is used with driver 6.1.7600.16385, or if PCI communication is in any way impaired. Up to DLL 536, error code 4 indicates a PCI error, too. As of DLL 537, PCI errors are indicated by error code 16, see the description there.</td> </tr> <tr> <td>5</td> <td>System error: not enough Windows memory.</td> </tr> <tr> <td>6</td> <td>Access error: board reserved for another user program.</td> </tr> <tr> <td>7</td> <td>Version error: DLL version (RTC5 DLL), RTC version (firmware file) and HEX version (DSP program file) not compatible.</td> </tr> <tr> <td>8</td> <td>System driver not found (get_last_error return code: RTC5_ACCESS_DENIED).</td> </tr> <tr> <td>9</td> <td>DriverCall error: loading of RTC5OUT.out file failed.</td> </tr> <tr> <td>10</td> <td>Configuration error: DSP register initialization failed.</td> </tr> <tr> <td>11</td> <td>Firmware error: loading of RTC5RBF.rbf file failed.</td> </tr> <tr> <td>12</td> <td>PCI download error: loading of RTC5DAT.dat file failed.</td> </tr> <tr> <td>13</td> <td>Busy error: Download locked, board is BUSY or INTERNAL-BUSY (get_last_error return code: RTC5_BUSY).</td> </tr> <tr> <td>14</td> <td>DSP memory error.</td> </tr> <tr> <td>15</td> <td>Verify error (only applicable for download verification).</td> </tr> <tr> <td>16</td> <td>PCI error: as of DLL 537. Or a possible driver/DLL conflict, see note on page 26: "Older RTC5 DLL files (version DLL 533 and lower) are not compatible with the new WDF drivers." .</td> </tr> </tbody> </table>		Value	Description	0	Success.	1	Reset error: the board could not be reset.	2	Unreset error: the board does not restart.	3	File error: file not found or cannot be opened.	4	Format error: RTC5OUT.out has incorrect format. PCI error: driver reported an error during PCI transfer because driver is not compatible with DLL. This occurs particularly when DLL 533 or earlier is used with driver 6.1.7600.16385, or if PCI communication is in any way impaired. Up to DLL 536, error code 4 indicates a PCI error, too. As of DLL 537, PCI errors are indicated by error code 16, see the description there.	5	System error: not enough Windows memory.	6	Access error: board reserved for another user program.	7	Version error: DLL version (RTC5 DLL), RTC version (firmware file) and HEX version (DSP program file) not compatible.	8	System driver not found (get_last_error return code: RTC5_ACCESS_DENIED).	9	DriverCall error: loading of RTC5OUT.out file failed.	10	Configuration error: DSP register initialization failed.	11	Firmware error: loading of RTC5RBF.rbf file failed.	12	PCI download error: loading of RTC5DAT.dat file failed.	13	Busy error: Download locked, board is BUSY or INTERNAL-BUSY (get_last_error return code: RTC5_BUSY).	14	DSP memory error.	15	Verify error (only applicable for download verification).	16	PCI error: as of DLL 537. Or a possible driver/DLL conflict, see note on page 26 : "Older RTC5 DLL files (version DLL 533 and lower) are not compatible with the new WDF drivers." .
Value	Description																																					
0	Success.																																					
1	Reset error: the board could not be reset.																																					
2	Unreset error: the board does not restart.																																					
3	File error: file not found or cannot be opened.																																					
4	Format error: RTC5OUT.out has incorrect format. PCI error: driver reported an error during PCI transfer because driver is not compatible with DLL. This occurs particularly when DLL 533 or earlier is used with driver 6.1.7600.16385, or if PCI communication is in any way impaired. Up to DLL 536, error code 4 indicates a PCI error, too. As of DLL 537, PCI errors are indicated by error code 16, see the description there.																																					
5	System error: not enough Windows memory.																																					
6	Access error: board reserved for another user program.																																					
7	Version error: DLL version (RTC5 DLL), RTC version (firmware file) and HEX version (DSP program file) not compatible.																																					
8	System driver not found (get_last_error return code: RTC5_ACCESS_DENIED).																																					
9	DriverCall error: loading of RTC5OUT.out file failed.																																					
10	Configuration error: DSP register initialization failed.																																					
11	Firmware error: loading of RTC5RBF.rbf file failed.																																					
12	PCI download error: loading of RTC5DAT.dat file failed.																																					
13	Busy error: Download locked, board is BUSY or INTERNAL-BUSY (get_last_error return code: RTC5_BUSY).																																					
14	DSP memory error.																																					
15	Verify error (only applicable for download verification).																																					
16	PCI error: as of DLL 537. Or a possible driver/DLL conflict, see note on page 26 : "Older RTC5 DLL files (version DLL 533 and lower) are not compatible with the new WDF drivers." .																																					



Ctrl Command	load_program_file
Comments	<p>If pPath = 0, then the path of the user program's current working directory is used.</p> <p>Caution: The user program's current working directory is not always the directory from which the user program was launched. The current working directory can change, for example, when a file from another directory is selected by the Windows Explorer (unless the "NoChangeDir" flag was set when incorporating the Explorer window into the user program).</p> <ul style="list-style-type: none"> After each hardware reset (powerup), the first user program must begin by issuing a load_program_file command during initialization of the RTC5 Board (see "Initialization of the Board", page 69). The command should also be executed (e.g. if another user program acquires the board – see acquire_RTC) when the board needs to be returned to the default state. If multiple RTC5 Boards are connected as master and slave, then load_program_file must be executed on all boards prior to initializing and operating the individual boards with further commands (see chapter 6.6.3 "Master/Slave Operation", page 93). The command load_program_file resets the RTC5, initializes the memory configuration (in the default configuration), performs a DSP memory check, loads the firmware (RTC5RBF.rbf), the program file RTC5OUT.out and a binary support file (RTC5DAT.dat) and starts the signal processor (DSP). After execution of the load_program_file command, the laser focus is positioned in the center of the image field at the point (0 0) and the laser control is deactivated. The load_program_file command does <i>not</i> load correction tables. Even 1-to-1 tables therefore need to be explicitly requested – see load_correction_file. Already-loaded correction tables remain loaded after load_program_file. During a RTC5 reset, list memory contents are erased and all parameters (e.g. memory configuration, internal variables, matrices, offsets and table assignments) previously set with config_list or select_cor_table are deleted or reset to their default values. During a reset, a correction table is assigned as by select_cor_table(1,0) but no scanner motion to the corrected output position is executed. The function only returns to the calling program, when DSP initialization has been completed. The load_program_file command is <i>not</i> executed (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list) or the board's INTERNAL-BUSY status is currently set. In contrast, the command <i>is</i> executed if a list has been previously terminated in an orderly manner or paused by set_wait (PAUSED status set), stop_execution or an external Stop.



Ctrl Command	load_program_file
comments (cont'd)	<ul style="list-style-type: none"> The files RTC5OUT.out, RTC5RBF.rbf and RTC5DAT.dat are included in the RTC5 software package. For easy identifying and archiving of different software versions, the files are also delivered zipped (the zip file names RTC5<...>_<Version>.zip include the version numbers). Copy or unzip the three files (of desired version) to the harddisk of your PC. Assorted versions of the RTC5 DLL and the files RTC5OUT.out, RTC5RBF.rbf and RTC5DAT.dat cannot be arbitrarily combined with another (each zip file in the RTC5 software includes a text file with corresponding version information). The command load_program_file performs a version compatibility check. If there is a version error, then the loaded programs remain in RTC5 memory, but the board is released by release_RTC directly after the version check and therefore is not available for further commands other than those not requiring access rights (get_last_error return code: RTC5_ACCESS_DENIED RTC5_VERSION_MISMATCH). To then load a correct program version, only the multi-board command n_load_program_file can be called. Hereby, temporary access rights are requested and released after the download (if the board has not been reserved by another user program; n_load_program_file does not perform an acquire_RTC command). In this case, the single-board command load_program_file does not get access rights for the board. Alternatively, the board can be made currentless to unload the program software (afterward also load_program_file can be used again). The filename extension for RTC5 program files is no longer *.hex (as with the RTC4, RTC3 and RTC2), but instead *.out. As before, the version number can be queried by the command get_hex_version. The dsp_start command does not exist for the RTC5 (see page 642). The DSP is automatically started by load_program_file.
RTC4→RTC5	<ul style="list-style-type: none"> This command parameters specifies a directory name with RTC5 (in contrast a file name with the RTC4). The command loads three files, with fixed formats and names (RTC5OUT.out, RTC5RBF.rbf, RTC5DAT.dat) (see above). After execution of the command, the laser control is deactivated.
Version info	Last change with version DLL 537.



Ctrl Command	load_stretch_table																						
Function	Loads a table with data pairs from an ASCII text file for enhanced 3D correction, see chapter 8.6.5 "Enhanced 3D Correction", page 198 .																						
Call	NoOfDataPairs = load_stretch_table(Name, No)																						
Parameters	<p>Name Name of the text file as a pointer to a null-terminated ANSI string (the text file may contain one or more tables) or NULL.</p> <p>No Signed 32-bit value.</p> <ul style="list-style-type: none"> • For $No \geq 0$, this parameter specifies which table in the text file shall be loaded (the parameter corresponds to the extension $<No>$ of the instruction [StretchTable<No>] at the beginning of the desired table). • $No < 0$: Reserved. 																						
Result	<p>Signed 32-bit value (a positive error code in case of an error, the negative number of found data pairs in case of success).</p> <table> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>< 0</td> <td>Success. The absolute value of the return value is equal to the number of valid data pairs found in the table.</td> </tr> <tr> <td>0</td> <td>Reserved.</td> </tr> <tr> <td>2</td> <td>Out of Memory (not enough Windows memory).</td> </tr> <tr> <td>3</td> <td>File not found.</td> </tr> <tr> <td>4</td> <td>DSP memory error.</td> </tr> <tr> <td>5</td> <td>BUSY error, board is BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).</td> </tr> <tr> <td>6</td> <td>Data error: data pairs missing.</td> </tr> <tr> <td>11</td> <td>PCI download error (get_last_error return code RTC5_SEND_ERROR).</td> </tr> <tr> <td>13</td> <td>The specified table number was not found in the file.</td> </tr> <tr> <td>15</td> <td>Verify error (get_last_error return code RTC5_VERIFY_ERROR, only possible with set_verify option set).</td> </tr> </tbody> </table>	Value	Description	< 0	Success. The absolute value of the return value is equal to the number of valid data pairs found in the table.	0	Reserved.	2	Out of Memory (not enough Windows memory).	3	File not found.	4	DSP memory error.	5	BUSY error, board is BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).	6	Data error: data pairs missing.	11	PCI download error (get_last_error return code RTC5_SEND_ERROR).	13	The specified table number was not found in the file.	15	Verify error (get_last_error return code RTC5_VERIFY_ERROR, only possible with set_verify option set).
Value	Description																						
< 0	Success. The absolute value of the return value is equal to the number of valid data pairs found in the table.																						
0	Reserved.																						
2	Out of Memory (not enough Windows memory).																						
3	File not found.																						
4	DSP memory error.																						
5	BUSY error, board is BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).																						
6	Data error: data pairs missing.																						
11	PCI download error (get_last_error return code RTC5_SEND_ERROR).																						
13	The specified table number was not found in the file.																						
15	Verify error (get_last_error return code RTC5_VERIFY_ERROR, only possible with set_verify option set).																						
Comments	<ul style="list-style-type: none"> • Details about enhanced 3D correction (e.g. format requirements for the text file's table entries) are described on page 198. • A successfully loaded table activates the new enhanced 3D correction. Here, load_stretch_table overwrites any previously loaded table. • If Name is not NULL, but no table was successfully read, then load_stretch_table returns an error code (e.g. code 13 if a No value is specified for which no [StretchTable<No>] entry is included in the text file), but otherwise has no effect (i.e. any previous successfully downloaded table stays enabled). 																						



Ctrl Command	load_stretch_table
Comments (cont'd)	<ul style="list-style-type: none"> If Name is NULL, then load_stretch_table disables any enhanced 3D correction enabled by a previous load_stretch_table command. The command is <i>not</i> executed (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list) or the board's INTERNAL-BUSY status is currently set. In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set). During execution of load_stretch_table, external starts are suppressed. Before loading a table, load_stretch_table performs a DSP memory check. In case of an error, error code 4 is returned.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 536, OUT 536.

Ctrl Command	load_sub
Function	Assigns a desired index to a subroutine defined by subsequent list commands and loads the subroutine into the protected buffer area ("List 3").
Call	<code>load_sub(Index)</code>
Parameter	Index Index of the indexed subroutine as an unsigned 32-bit value. Allowed range [0 ... 1023].
Comments	<ul style="list-style-type: none"> Up to 1024 indexed subroutines can be stored. If <code>Index > 1023</code> then the load_sub command is ignored (get_last_error return code RTC5_PARAM_ERROR). The address in the protected buffer area where the subroutine should be stored is automatically determined and internally managed. Indexed subroutines must be terminated with a list_return command. This is a prerequisite for actual storage of the commands, entry of the start address and other information (e.g., the number of commands) into the internal management table, and initiating a flush of the list input buffer (see page 75). Otherwise (the input pointer is altered without a preceding list_return command) the subroutine with this index is not available. An indexed subroutine is not stored if the protected buffer area ("List 3") was not previously configured for a sufficient size beyond "List 1" and "List 2". If list_return is the next command after load_sub, then the corresponding subroutine is deleted from the internal management table. Indexed subroutines can be called by the sub_call command along with the corresponding index (see section "Calling Subroutines", page 83). Observe all notes in section "Indexed Subroutines", page 82.
RTC4→ RTC5	New command.
References	list_return , sub_call , load_char , load_text_table



Ctrl Command	load_text_table
Function	Assigns a desired index to a text string defined by subsequent list commands and loads the text string into the protected buffer area ("List 3").
Call	<code>load_text_table(Index)</code>
Parameter	Index Index of the indexed text string as an unsigned 32-bit value. Allowed range [0 ... 41].
Comments	<ul style="list-style-type: none"> • Up to 42 indexed text strings can be stored (for marking times, dates and serial numbers by other commands). The following ordering applies: <ul style="list-style-type: none"> – Index = 0 ... 9: digits for marking the time and date [0 ... 9] – Index = 10...21: months [January ... December] – Index = 22...28: days-of-the-week [Sunday ... Saturday] – Index = 29: blank character for marking serial numbers – Index = 30...39: digits for marking serial numbers [0 ... 9] – Index = 40: text for "a.m." – Index = 41: text for "p.m." If <code>Index > 41</code> then the <code>load_text_table</code> command is ignored (<code>get_last_error</code> return code <code>RTC5_PARAM_ERROR</code>). • Even if digits are defined by <code>mark_text</code> instead of as individual characters with <code>mark_char</code>, the character set can still be subsequently switched for this purpose (see <code>select_char_set</code>). • The addresses in the protected buffer area where the text string definitions are stored are automatically determined and internally managed. Management is independent of that for indexed subroutines (see <code>load_sub</code>) and character definitions (see <code>load_char</code>). • Indexed text string definitions must be terminated with a <code>list_return</code> command. This is a prerequisite for actual storage of the commands, entry of the start address and other information (e.g., the number of commands) into the internal management table, and initiating a flush of the list input buffer (see page 75). Otherwise (the input pointer is altered without a preceding <code>list_return</code> command) the text string with this index is not available. • An indexed text string definition is not stored if the protected buffer area ("List 3") was not previously configured for a sufficient size beyond "List 1" and "List 2". • If <code>list_return</code> is the next command after <code>load_text_table</code>, then the corresponding text string definition is deleted from the internal management table. • Also observe all notes in the chapter 6.5.2 "Character Sets and Text Strings", page 87.
RTC4→ RTC5	New command.
References	list_return , load_sub , load_char



Ctrl Command	load_varpolydelay																				
Function	Loads a table with data points from an ASCII text file for the scaling function of the variable polygon delay (see page 120).																				
Call	NoOfDataPoints = load_varpolydelay(Name, No)																				
Parameters	<p>Name Name of the text file as a pointer to a null-terminated ANSI string (the text file may contain one or more tables).</p> <p>No This parameter (an unsigned 32-bit value) specifies which table in the text file shall be loaded (the parameter corresponds to the extension <No> of the instruction [VarPolyTable<No>] at the beginning of the desired table).</p>																				
Result	<p>Signed 32-bit value (a positive error code in case of an error, the negative number of found data points in case of success).</p> <table> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>-1...- 50</td> <td>Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored (see page 120).</td> </tr> <tr> <td>-1024</td> <td>For Name = 0: the table initialized according to $1-\cos(\phi)$ was internally loaded (as with program start; see figure 36).</td> </tr> <tr> <td>1</td> <td>No valid data points found (though Table No found).</td> </tr> <tr> <td>3</td> <td>File not found.</td> </tr> <tr> <td>4</td> <td>DSP memory error.</td> </tr> <tr> <td>5</td> <td>BUSY error, board is BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).</td> </tr> <tr> <td>8</td> <td>Board is locked by another user program (get_last_error return code RTC5_ACCESS_DENIED).</td> </tr> <tr> <td>11</td> <td>PCI error (get_last_error return code RTC5_SEND_ERROR), verify error (get_last_error return code RTC5_VERIFY_ERROR).</td> </tr> <tr> <td>13</td> <td>The specified table number was not found in the file.</td> </tr> </tbody> </table>	Value	Description	-1...- 50	Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored (see page 120).	-1024	For Name = 0: the table initialized according to $1-\cos(\phi)$ was internally loaded (as with program start; see figure 36).	1	No valid data points found (though Table No found).	3	File not found.	4	DSP memory error.	5	BUSY error, board is BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).	8	Board is locked by another user program (get_last_error return code RTC5_ACCESS_DENIED).	11	PCI error (get_last_error return code RTC5_SEND_ERROR), verify error (get_last_error return code RTC5_VERIFY_ERROR).	13	The specified table number was not found in the file.
Value	Description																				
-1...- 50	Success. The absolute value of the return value is equal to the number of valid data points found in the table. Invalid entries are ignored (see page 120).																				
-1024	For Name = 0: the table initialized according to $1-\cos(\phi)$ was internally loaded (as with program start; see figure 36).																				
1	No valid data points found (though Table No found).																				
3	File not found.																				
4	DSP memory error.																				
5	BUSY error, board is BUSY or INTERNAL-BUSY, no download (get_last_error return code RTC5_BUSY).																				
8	Board is locked by another user program (get_last_error return code RTC5_ACCESS_DENIED).																				
11	PCI error (get_last_error return code RTC5_SEND_ERROR), verify error (get_last_error return code RTC5_VERIFY_ERROR).																				
13	The specified table number was not found in the file.																				
Comments	<ul style="list-style-type: none"> The format requirements for text file's table entries with data points for the customized variable polygon delay are described in "Customizing the Variable Polygon Delay" on page 120. When loading the table, the RTC5 determines suitable values for the entire range of angles by linear interpolation. The command load_varpolydelay overwrites any previously loaded table for the variable polygon delay. For Name = 0 (as during initialization by load_program_file), the internal (default) table for the variable polygon delay ($1-\cos(\phi)$, see figure 36) is loaded. The command is <i>not</i> executed (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list) or the board's INTERNAL-BUSY status is currently set. In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set). During execution of load_varpolydelay, external starts are suppressed. Before loading a table, load_varpolydelay performs a DSP memory check. In case of an error, error code 4 is returned. 																				



Ctrl Command	load_varpolydelay
RTC4→ RTC5	<p>Essentially unchanged functionality, however:</p> <ul style="list-style-type: none"> • To return to the internal standard polygon delay table, a reset or renewed program loading by load_program_file is no longer necessary (unlike with the RTC4; see Name = 0 above). • The ASCII text file can have any filename extension (i.e. not only *.STB as with the RTC4).
References	load_program_file , set_delay_mode

Ctrl Command	load_z_table																										
Function	Loads coefficients A, B and C into the currently assigned 3D correction table.																										
Restriction	If the 3D option has not been enabled or a 3D correction table has not been assigned (see select_cor_table), then the command returns the error code 12 or 13 and otherwise has no effect.																										
Call	ErrorNo = load_z_table(A, B, C)																										
Parameters	<p>A, B, C coefficients (as 64-bit IEEE floating point values) of the parabolic function $z_{out} = A + BI + CI^2$ which is used for calculating the Z output values.</p> <p>Allowed range:</p> <ul style="list-style-type: none"> • For A: [-67108864.0 ... 67108864.0] • For B: [-2048.0 ... 2048.0] • For C: [-16.0 ... 16.0] <p>Out-of-range values are edge-clipped.</p>																										
Result	<p>Error code as an unsigned 32-bit value:</p> <p>Error bits with values 1 to 64 can also occur combined, but not in conjunction with error codes 11 to 14, which only occur separately. Warnings 12 and 13 are only returned if no other errors exist.</p> <table> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>0</td> <td>No error.</td> </tr> <tr> <td>1</td> <td>A exceeded the maximum allowed value.</td> </tr> <tr> <td>2</td> <td>A undercut the minimum allowed value.</td> </tr> <tr> <td>4</td> <td>B exceeded the maximum allowed value.</td> </tr> <tr> <td>8</td> <td>B undercut the minimum allowed value.</td> </tr> <tr> <td>16</td> <td>C exceeded the maximum allowed value.</td> </tr> <tr> <td>32</td> <td>C undercut the minimum allowed value.</td> </tr> <tr> <td>64</td> <td>Execution denied (possibly a BUSY or INTERNAL BUSY error; for exact reason: see get_last_error).</td> </tr> <tr> <td>11</td> <td>Access denied.</td> </tr> <tr> <td>12</td> <td>3D Option not enabled.</td> </tr> <tr> <td>13</td> <td>No 3D table is currently assigned.</td> </tr> <tr> <td>14</td> <td>System driver not found.</td> </tr> </tbody> </table>	Value	Description	0	No error.	1	A exceeded the maximum allowed value.	2	A undercut the minimum allowed value.	4	B exceeded the maximum allowed value.	8	B undercut the minimum allowed value.	16	C exceeded the maximum allowed value.	32	C undercut the minimum allowed value.	64	Execution denied (possibly a BUSY or INTERNAL BUSY error; for exact reason: see get_last_error).	11	Access denied.	12	3D Option not enabled.	13	No 3D table is currently assigned.	14	System driver not found.
Value	Description																										
0	No error.																										
1	A exceeded the maximum allowed value.																										
2	A undercut the minimum allowed value.																										
4	B exceeded the maximum allowed value.																										
8	B undercut the minimum allowed value.																										
16	C exceeded the maximum allowed value.																										
32	C undercut the minimum allowed value.																										
64	Execution denied (possibly a BUSY or INTERNAL BUSY error; for exact reason: see get_last_error).																										
11	Access denied.																										
12	3D Option not enabled.																										
13	No 3D table is currently assigned.																										
14	System driver not found.																										



Ctrl Command	load_z_table
Comments	<ul style="list-style-type: none"> This command is only needed for re-calibrating the Z-axis in a 3-axis scan system (for adjusting corresponding coefficients see page 196). Both positive and negative coefficients can be specified. The coefficients should preferably be chosen so that all Z output values lie within the range [-32768 ... +32767]. The command is ignored (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list) or the board's INTERNAL-BUSY status is currently set. In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set). This command should always be used <i>after</i> the command load_correction_file, since load_correction_file sets the three coefficients to the default values of the loaded correction table. Prior to the next list command that directly follows load_z_table, a smooth transition from the last Z position to the changed position is performed at jump_speed. You can also immediately force this by select_cor_table. This way, time delays can be avoided during an external start. Coefficients A, B and C can be queried from the loaded 3D correction table by get_table_para (and from the currently assigned 3D correction table by get_head_para).
RTC4→ RTC5	Unchanged functionality (except for changed value ranges and error codes). If the correct calibration factors are used (see page 194), then the same ABC coefficients can be used on the same 3-axis scan system with the RTC4 and RTC5 boards.
Version info	Last change with version DLL 516.
References	get_z_distance , read_abc_from_file , write_abc_to_file

Normal List Command	long_delay
Function	Pauses further processing of the list for the specified time.
Call	<code>long_delay(Delay)</code>
Parameter	<p>Delay Delay time in <i>bits</i> as an unsigned 32-bit value. <i>1 bit equals 10 µs.</i> Allowed range: $0 \leq \text{delay} \leq (2^{32}-1)$.</p>
Comments	<ul style="list-style-type: none"> This command (if applicable) switches off the "laser active" laser control signals after a LaserOff delay, waits for a possible scanner delay and pauses further processing of the list for the specified time. This command should always be called after changing the lamp current of a YAG laser to obtain a constant laser power. The command list_nop corresponds to <code>long_delay(1)</code>.
RTC4→ RTC5	Unchanged functionality (except for the increased range of values).



Normal List Command	mark_abs
Function	Moves the laser focus at marking speed along a 2D vector from the current position to the specified position (absolute coordinate values) within a two-dimensional image field.
Call	<code>mark_abs(X, Y)</code>
Parameters	X, Y Absolute coordinates of the mark vector end point in <i>bits</i> as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.
Comments	<ul style="list-style-type: none"> If the marking speed has not been previously explicitly set by set_mark_speed or set_mark_speed_ctrl, then the marking is executed at a predefined marking speed of 1000 <i>bits per ms</i>. The “laser active” laser control signals are automatically turned on at the beginning of the command (or remain on after a directly preceding mark or arc command). The defined scanner and laser delays are thereby taken into account (see chapter 7.2 “Delay Settings for Synchronizing Scan Head and Laser Control”, page 111). Note that other delays are executed in Sky writing mode (see page 127). Exception: zero-length mark commands (see notes on page 115). During the conversion of specified coordinate values into scan system output values, any previously defined coordinate transformations, assigned correction tables etc. are taken into account after successful microvectorization (see page 141).
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).
References	set_mark_speed , set_scanner_delays , mark_rel , arc_abs , timed_mark_abs



Normal List Command	mark_abs_3d
Function	Moves the laser focus at marking speed along a 3D vector from the current position to the specified position (absolute coordinate values) within a three-dimensional process volume.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as mark_abs . However, microvectorization is calculated like a 3D command and hence influences the effective marking speed in the XY plane.
Call	<code>mark_abs_3d(X, Y, Z)</code>
Parameters	X, Y, Z Absolute coordinates of the mark vector end point in <i>bits</i> as signed 32-bit values. Allowed range: <ul style="list-style-type: none">• For X and Y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.• For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped.
Comments	<ul style="list-style-type: none">• Except for the additional motion in the third dimension, this command functions similarly to the mark_abs command (see the comments there).• The X and Y axes can be controlled with 20-bit resolution, the Z axis with 16-bit resolution. The RTC5 (in RTC5 mode as well as in RTC4 compatibility mode) automatically upscales Z coordinate values internally to 20-bit values, so that the three dimensions of space can be handled equivalently in terms of the supplied jump speed value.• During calculation of the Z output value to the scan system, any previously defined offset to the Z coordinate and focal length is taken into account (see page 141).
RTC4→RTC5	Unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 and RTC4.
References	mark_abs , mark_rel_3d , timed_mark_abs_3d



Undelayed Short List Command	mark_char
Function	Marks an indexed character.
Call	<code>mark_char(Char)</code>
Parameter	Char Index of the indexed character to be marked (as an unsigned 32-bit value, allowed range: [0 ... 1023]). The following applies: $\text{Char} = \text{character set number} * 256 + \text{ASCII number of the character}$ (character sets are numbered 0 to 3).
Comments	<ul style="list-style-type: none"> The mark_char command reads the indexed character's starting address from the internal management table based on the supplied index and then calls the command list_call (see also the comments there), which then starts the corresponding command list. The mark_char command starts indexed characters in protected memory (that were loaded and/or referenced by load_char, load_disk or copy_dst_src) as well as indexed subroutines in the unprotected list area (that were referenced as characters by set_char_pointer or copy_dst_src). If no character is referenced for the supplied index, then the jump is suppressed and execution continues at the command located after the calling position. In some circumstances, a list_nop might be executed (siehe page 250). get_char_pointer(Char) can be used to determine whether a character has been referenced for a particular index. If no character has been referenced, this command returns the value “-1” (d.h. $2^{32}-1$). If $\text{Char} > 1023$, then mark_char is, already during loading, replaced by a list_nop (get_last_error return code RTC5_PARAM_ERROR). Absolute vector and arc commands execute absolutely after being called with mark_char. If the character needs to execute at various locations within the image field, then either the command list can only contain relative mark, arc or jump commands or mark_char_abs must be used instead. The called character should not contain mark_text commands that also contain this character. Such text is <i>not</i> marked. The called character itself then might not be complete. See also chapter 6.5.2 “Character Sets and Text Strings”, page 87.
RTC4→ RTC5	New command.
References	mark_char_abs , mark_text , set_char_pointer , get_char_pointer



Undelayed Short List Command	mark_char_abs
Function	Marks an indexed character. In the called command list (see below), any absolute vector and arc commands receive an offset (based on the current coordinates at the time of the call).
Call	<code>mark_char_abs(Char)</code>
Parameter	Char Index of the indexed character to be marked (as an unsigned 32-bit value, allowed range: [0 ... 1023]). The following applies: Char = character set number * 256 + ASCII number of the character (character sets are numbered 0 to 3).
Comments	<ul style="list-style-type: none"> The mark_char_abs command reads the indexed character's starting address from the internal management table based on the supplied index and then calls the command list_call_abs (see also the comments there), which then starts the corresponding command list. If the command list of the called character contains no absolute commands, then there is no difference between mark_char_abs and mark_char.
RTC4→ RTC5	New command.
References	mark_char



Normal List Command	mark_date
Function	Marks a part of the date previously stored by time_fix , time_fix_f or time_fix_f_off in the selected format at the current position.
Call	mark_date(Part, Mode)
Parameters	<p>Part This parameter (unsigned 32-bit value, allowed range: [0 ... 7]) specifies which part of the date should be marked.</p> <ul style="list-style-type: none"> = 0: Year (only the last two digits). = 1: Month (in customer specific notation). = 2: Day (corresponding to the normal Gregorian date). = 3: Day-of-the-week (in customer specific notation). = 4: Julian day (see also time_fix_f_off). = 5: Year (four digits). = 6: Month as numerals (January = 1, ...). = 7: Day-of-the-week as numerals (Sunday = 1, ...). <p>Mode Selection of the format as an unsigned 32-bit value (allowed range: [0 ... 3]).</p> <p>a) The number of leading zeros in the day number (only affects Part = 2, 4, 6 or 7).</p> <p>Bit#0 = 0: leading zeros are suppressed.</p> <p>Bit#0 = 1: day of the month (Part = 2), always two digits. day of the year (Part = 4), always three digits. month (Part = 6), always two digits. day-of-the-week (Part = 7), always two digits.</p> <p>b) Desired character set for marking digits (only affects Part = 0, 2 or 4 ... 7). Bit#1 = 0: The indexed text string for digits [0...9], as defined by load_text_table or set_text_table_pointer, is marked.</p> <p>Bit#1 = 1: The indexed characters for digits [0...9], as defined by load_char or set_char_pointer, are marked. The desired character set can be specified with select_char_set.</p> <p>For Part = 1 or 3, days of the month or days of the week are marked by indexed text strings (Index = 10...28) defined with load_text_table or set_text_table_pointer (here, the parameter Mode is ignored). For marking as numerals, also Part = 6 or 7 can be used (then Mode is considered).</p>



Normal List Command	mark_date
Comments	<ul style="list-style-type: none"> Before marking dates (after every boot-up), the RTC5 and PC times should be synchronized (see time_update) and the current (to be marked) date value should be stored with time_fix, time_fix_f or time_fix_f_off (see "Marking Dates, Times and Serial Numbers", page 170). The complete date can be marked by multiple calls of the mark_date command. The mark_date command reads (according to the stored date and according to the selected date part) the starting address of the corresponding indexed text string or character from the internal management table and then calls the command list_call (see also the comments there) an appropriate number of times, which then starts the corresponding command list. The command lists must contain marking instructions for digits 0...9 and for the month and day-of-the-week designations (see page 88). Non-defined text strings or characters are ignored (i.e. not marked). The called indexed text strings can also contain calls to indexed characters (mark_char or mark_char_abs) and complete texts (mark_text or mark_text_abs). In the latter case, the character set can be switched when needed (before marking by mark_date) with select_char_set (see also "Character Sets and Text Strings", page 87). If <code>Part > 7</code> and/or <code>Mode > 3</code>, then mark_date is, already during loading, replaced by a list_nop (get_last_error return code <code>RTC5_PARAM_ERROR</code>). Absolute vector and arc commands execute absolutely after being called with mark_date. If date markings need to execute at various locations within the image field, then the corresponding indexed text strings (or characters) can only contain relative mark, arc or jump commands or mark_date_abs must be used instead. When marking Gregorian dates or Julian days, the transition to the next day occurs at 24:00 o'clock. Leap years are represented in both date styles.
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). The command was previously only available for the RTC SCANalone Board, i.e. the standalone version of the RTC4 Board.
References	time_fix , time_fix_f , time_fix_f_off , load_text_table , set_text_table_pointer , mark_date_abs



Normal List Command	mark_date_abs
Function	Marks a part of the date previously stored by time_fix , time_fix_f or time_fix_f_off in the selected format at the current position. In the called indexed text strings or characters (see below), any absolute vector and arc commands receive an offset (based on the current coordinates at the time of the call).
Call	<code>mark_date_abs(Part, Mode)</code>
Parameters	Part See mark_date .
	Mode See mark_date .
Comments	<ul style="list-style-type: none"> The mark_date_abs command works like mark_date; however, internal calling of the indexed text strings (or characters) is by list_call_abs instead of list_call. If the command lists of the called indexed text strings (or characters) contain no absolute commands, then there is no difference between mark_date_abs and mark_date.
RTC4→ RTC5	New command.
References	mark_date



Normal List Command	mark_ellipse_abs
Function	Moves the laser focus at marking speed along an elliptical arc around the specified midpoint (absolute coordinate values) within a two-dimensional image field.
Call	<code>mark_ellipse_abs(X, Y, Alpha)</code>
Parameters	<p>X, Y Absolute coordinates of the ellipse midpoint in <i>bits</i> as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.</p> <p>Alpha Angle (in ° as a 64-bit IEEE floating point value) between elliptical half-axis <i>a</i> (defined by set_ellipse) and the X axis (the angle is referenced to the positive X direction, positive angle values correspond to counterclockwise angles). Alpha gets normalized to the value range [0...<360°].</p>
Comments	<ul style="list-style-type: none"> The parameters for mark_ellipse_abs only determine the position and orientation of the to-be-executed arc. Its shape must be specified by set_ellipse before marking by mark_ellipse_abs. For descriptions of the individual parameters, see also page 105. If the arc starting point defined by mark_ellipse_abs and set_ellipse does not equal the current position, then a hard jump to the starting point is executed prior to marking (see also notes on page 105). See also all comments for arc_abs.
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified ellipse midpoint coordinate values by 16 (the allowed range of values is correspondingly reduced to [-524288 ... 524287]).
Version info	<ul style="list-style-type: none"> Available as of version DLL 518, OUT 517, RBF 515. For older RTC5 Boards with DSP version numbers < 2 (get_RTC_version bits #16-23), the command is neither executed nor replaced by list_nop (get_last_error return code: RTC5_TYPE_REJECTED).
References	set_ellipse , mark_ellipse_rel , set_mark_speed , set_scanner_delays , arc_abs



Normal List Command	mark_ellipse_rel
Function	Moves the laser focus at marking speed along an elliptical arc around the specified midpoint (relative coordinate values) within a two-dimensional image field.
Call	<code>mark_ellipse_rel(dx, dy, Alpha)</code>
Parameters	<p><code>dx, dy</code> Relative coordinates of the ellipse midpoint in <i>bits</i> as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.</p> <p><code>Alpha</code> Angle (in ° as a 64-bit IEEE floating point value) between elliptical half-axis <i>a</i> (defined by set_ellipse) and the X axis (see mark_ellipse_abs).</p>
Comments	<ul style="list-style-type: none"> The coordinates for the ellipse midpoint are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to mark_ellipse_abs (see the comments there). The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.
RTC4→ RTC5	<p>New command.</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified ellipse midpoint coordinate values by 16 (the allowed range of values is correspondingly reduced to [-524288 ... 524287]).</p>
Version info	<ul style="list-style-type: none"> Available as of version DLL 518, OUT 517, RBF 515. For older RTC5 Boards with DSP version numbers < 2 (get_RTC_version bits #16-23), the command is neither executed nor replaced by list_nop (get_last_error return code: RTC5_TYPE_REJECTED).
References	mark_ellipse_abs , set_ellipse



Normal List Command	mark_rel
Function	Moves the laser focus at marking speed along a 2D vector from the current position to the specified position (relative coordinate values) within a two-dimensional image field.
Call	<code>mark_rel(dx, dy)</code>
Parameters	<p><code>dx, dy</code> <i>Relative coordinates of the mark vector end point in bits as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.</i></p> <p>The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</p>
Comments	<ul style="list-style-type: none"> The coordinates for the mark vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to mark_abs (see the comments there).
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).
References	mark_abs , mark_rel_3d , timed_mark_rel



Normal List Command	mark_rel_3d
Function	Moves the laser focus at marking speed along a 3D vector from the current position to the specified position (relative coordinate values) within a three-dimensional process volume.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as mark_rel . However, microvectorization is calculated like a 3D command and hence influences the effective marking speed in the XY plane.
Call	<code>mark_rel_3d(dx, dy, dz)</code>
Parameters	<p><code>dx, dy, dz</code> <i>Relative coordinates of the mark vector end point in bits as signed 32-bit values. Allowed range:</i></p> <ul style="list-style-type: none"> • For X and Y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. • For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped.
Comments	<ul style="list-style-type: none"> • The coordinates for the mark vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to mark_abs_3d (see the comments there).
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 and RTC4.
References	mark_abs_3d , mark_abs , mark_rel , timed_mark_rel_3d



Normal List Command	mark_serial
Function	Marks the current serial number of the serial-number-set most recently selected by select_serial_set_list (or of serial-number-set 0 after load_program_file) in the selected format at the current position. Afterward the serial number is (optionally) automatically incremented.
Call	<code>mark_serial(Mode, Digits)</code>
Parameters	<p>Mode Selection of the serial number format and (de)activation of automatic serial number incrementing as an unsigned 32-bit value. $Mode = 20 \times M_1 + 10 \times M_2 + M_3$ Allowed range for M_1: [0, 1], for M_2: [0, 1], for M_3: [0 … 2].</p> <p>a) Selection of the character set for digit marking.</p> <ul style="list-style-type: none"> $M_1 = 0$: The indexed text string for digits [30…39], as defined by load_text_table or set_text_table_pointer, is marked. $M_1 = 1$: The indexed characters for digits [0…9], as defined by load_char or set_char_pointer, are marked. The desired character set can be selected (previously) with select_char_set. <p>b) Incrementing of the serial number after marking.</p> <ul style="list-style-type: none"> $M_2 = 0$: The serial number is incremented after marking. $M_2 = 1$: The serial number is <i>not</i> incremented after marking. <p>c) Marking of leading zeros.</p> <ul style="list-style-type: none"> $M_3 = 0$: Leading zeros are marked as zeros. Dependent on M_1 the corresponding indexed text string definition (Index = 30) or the indexed character definition '0' is used. $M_3 = 1$: Leading zeros are suppressed (left-justified marking). $M_3 = 2$: Leading zeros are marked as blank characters (right-justified marking). Dependent on M_1 the corresponding indexed text string definition (Index = 29) or the indexed character definition ' ' is used. <p>Digits Number [0-12] of to-be-marked digits as an unsigned 32-bit value. Allowed range: [0-12]. Larger values are clipped.</p>
Comments	<ul style="list-style-type: none"> The first serial number to be marked must have been previously specified by set_serial, set_serial_step or set_serial_step_list; otherwise, the starting serial number is 0. The starting serial number can have a maximum length of 10 digits. With every call of mark_serial, the serial number is formatted in accordance with M_3 and when $M_2 = 0$ it is automatically incremented before the actual marking. Here, the increment size is 1 unless otherwise specified by set_serial_step or set_serial_step_list. The current serial number can be queried with get_list_serial, e.g. after an aborted list to determine if the current number was incremented or not. If the incremented serial number exceeds 10^{digits}, then marking begins again at 0. The control command set_max_counts allows specification of the maximum number of external list starts and thus the maximum number of markings. Here, all markings of all serial-number-sets contribute jointly to the count.



Normal List Command	mark_serial
Comments (cont'd)	<ul style="list-style-type: none"> If digits = 0, then a "markless" marking is executed. If M₂ = 0, then the serial number is incremented by 1 (any increment size defined by set_serial_step or set_serial_step_list are not used in this case!). This is useful if a single serial number should be omitted and can also be used (as with the RTC4) for indirectly defining the increment size (or with the RTC5 an <i>additional</i> increment size) for incrementing serial numbers. For each to-be-marked serial number digit, the mark_serial command reads the starting address of the corresponding indexed text string (or – for M₁ = 1 – of the corresponding indexed character) from the internal management table and then calls the command list_call (see also the comments there) an appropriate number of times, which then starts the corresponding command list. The command lists must contain marking instructions for digits 0...9 (see page 88). Non-defined text strings or characters are ignored (i.e. not marked). The called indexed text strings can also contain calls to indexed characters (mark_char or mark_char_abs) and complete texts (mark_text or mark_text_abs). In the latter case, the character set can be switched if needed (before marking by mark_serial) with select_char_set (see also "Character Sets and Text Strings", page 87). For invalid Mode values, the mark_serial is, already during loading, replaced by a list_nop (get_last_error return code RTC5_PARAM_ERROR). Absolute vector and arc commands execute absolutely after being called with mark_serial. If serial number markings need to execute at various locations within the image field, then the corresponding indexed text strings (or characters) can only contain relative mark, arc or jump commands or mark_serial_abs must be used instead.
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). The command was previously only available for the RTC SCANalone Board, i.e. the standalone version of the RTC4 Board.
References	set_serial , set_serial_step , set_serial_step_list , get_list_serial , set_max_counts , get_counts , load_text_table , set_text_table_pointer , mark_serial_abs



Normal List Command	mark_serial_abs
Function	Marks the current serial number of the serial-number-set most recently selected by select_serial_set_list (or of serial-number-set 0 after load_program_file) in the selected format at the current position. In the called indexed text strings or characters (see below), any absolute vector and arc commands receive an offset (based on the current coordinates at the time of the call). Afterward the serial number is (optionally) automatically incremented.
Call	<code>mark_serial_abs(Mode, Digits)</code>
Parameters	Mode See mark_serial .
	Digits See mark_serial .
Comments	<ul style="list-style-type: none"> The mark_serial_abs command works like mark_serial; however, internal calling of the indexed text strings (or characters) is by list_call_abs instead of list_call. If the command lists of the called indexed text strings (or characters) contain no absolute commands, then there is no difference between mark_serial_abs and mark_serial.
RTC4→ RTC5	New command.
References	mark_serial



Variable List Command	mark_text
Function	Marks a null-terminated string.
Call	<code>mark_text(Text)</code>
Parameter	Text PC memory address of the first character (byte) of the to-be-marked text string as a pointer to a null-terminated ANSI string
Comments	<ul style="list-style-type: none"> The to-be-marked text (character sequence, byte array, null-terminated string) must be terminated with a \0 character (0 byte, NULL). The 0\ character itself is not marked. When a mark_text command is loaded, the to-be-marked text (if more than 12 characters in length, \0 not included) is split into blocks of 12 characters, with each block receiving its own mark_text command in the list memory (keep this in mind to prevent unintended overflow of the corresponding buffer area). During processing of the individual mark_text commands, the corresponding mark_char commands (indexed characters) are executed in accordance with the selected character set. The desired character set can be selected prior to the mark_text command by the select_char_set command. For the default setting, character set 0 is used. If the command select_char_set is used within a called indexed character, then all subsequently called indexed characters are marked using this character set. If the end of a list ("List 1" or "List 2") is reached during loading of a mark_text command, then loading continues at the start of the corresponding list. In contrast, loading in the protected area (as part of an indexed subroutine) is aborted (get_last_error return code RTC5_REJECTED) and the indexed subroutine is not stored. Absolute vector and arc commands execute absolutely after being called with mark_text. If the text string needs to execute at various locations within the image field, then either the indexed character definitions can only contain relative mark, arc or jump commands or mark_text_abs must be used instead. The mark_text command should not be used within an indexed character definition. The corresponding text is <i>not</i> marked and the indexed character is therefore not fully processed.
RTC4→ RTC5	New command.
References	mark_text_abs



Variable List Command	mark_text_abs
Function	Marks a null-terminated string. In the called indexed characters (see below), any absolute vector and arc commands receive an offset (based on the current coordinates at the time of the call).
Call	mark_text_abs(Text)
Parameter	Text PC memory address of the first character (byte) of the to-be-marked text string as a pointer to a null-terminated ANSI string.
Comments	<ul style="list-style-type: none"> During processing of the individual mark_text_abs commands, the corresponding mark_char_abs commands (indexed characters) are executed in accordance with the selected character set. If the command list of the called indexed character contains no absolute commands, then there is no difference between mark_text_abs and mark_text.
RTC4 → RTC5	New command.
References	mark_text

Normal List Command	mark_time
Function	Marks a part of the time previously stored by time_fix , time_fix_f or time_fix_f_off in the specified format at the current position.
Call	mark_time(Part, Mode)
Parameters	<p>Part This parameter (unsigned 32-bit value, allowed range: [0 ... 4]) specifies which part of the time to mark.</p> <p>= 0: Hours (24-h time, no a.m./p.m.) = 1: Minutes = 2: Seconds = 3: Hours (12-h time, no a.m./p.m.) = 4: a.m./p.m. (automatically switched in accordance with 24-h time)</p> <p>Mode Format choice as an unsigned 32-bit value (allowed range: [0...3], only affects Part = 0...3).</p> <p>a) Number of leading zeros: Bit#0 = 0: Leading zeros are suppressed. Bit#0 = 1: always two digits.</p> <p>b) Character set choice for marking of digits: Bit#1 = 0: The indexed text strings for digits [0...9], as defined by load_text_table or set_text_table_pointer, are marked.</p> <p>Bit#1 = 1: The indexed characters for digits [0...9], as defined by load_char or set_char_pointer, are marked. The desired character set can be specified with select_char_set.</p> <p>a.m./p.m. (Part = 4) can only be marked in accordance with the indexed text strings (Index = 40, 41) defined by load_text_table or set_text_table_pointer. Here, the Parameter Mode is ignored.</p>



Normal List Command	mark_time
Comments	<ul style="list-style-type: none"> Before marking times (after every boot-up), the RTC5 and PC times should be synchronized (see time_update) and the current (to be marked) time should be stored with time_fix, time_fix_f or time_fix_f_off (see "Marking Dates, Times and Serial Numbers", page 170). The complete time can be marked by multiple calls of the mark_time command. The mark_time command reads (according to the stored time and according to the selected time part) the starting address of the corresponding indexed text string (or – for Part = 0...3 and Mode = 2 or 3 – of the corresponding indexed character) from the internal management table and then calls the command list_call (see also the comments there) an appropriate number of times, which starts the corresponding command list. The command lists must contain marking instructions for digits 0...9 and for a.m./p.m. (see page 88). Non-defined text strings or characters are ignored (i.e. not marked). The called indexed text strings can also contain calls to indexed characters (mark_char or mark_char_abs) and complete texts (mark_text or mark_text_abs). In the latter case, the character set can be switched, if needed (before marking with mark_time), by select_char_set (see also "Character Sets and Text Strings", page 87). If Part > 4 and/or Mode > 3, then mark_time is, already during loading, replaced by a list_nop (get_last_error return code <code>RTC5_PARAM_ERROR</code>). Absolute vector and arc commands execute absolutely after being called with mark_time. If time markings need to execute at various locations within the image field, then the corresponding indexed text strings (or characters) can only contain relative mark, arc or jump commands or mark_time_abs must be used instead.
RTC4→RTC5	Unchanged functionality (except for the extended range of values). The command was previously only available for the RTC SCANalone Board, i.e. the standalone version of the RTC4 Board.
References	time_fix , time_fix_f , time_fix_f_off , load_text_table , set_text_table_pointer , mark_time_abs



Normal List Command	mark_time_abs				
Function	Marks a part of the time previously stored by time_fix , time_fix_f or time_fix_f_off in the specified format at the current position. In the called indexed text strings or characters (see below), any absolute vector and arc commands receive an offset (based on the current coordinates at the time of the call).				
Call	<code>mark_time_abs(Part, Mode)</code>				
Parameters	<table> <tr> <td>Part</td> <td>See mark_time.</td> </tr> <tr> <td>Mode</td> <td>See mark_time.</td> </tr> </table>	Part	See mark_time .	Mode	See mark_time .
Part	See mark_time .				
Mode	See mark_time .				
Comments	<ul style="list-style-type: none"> The mark_time_abs command works like mark_time; however, internal calling of the indexed text strings (or characters) is by list_call_abs instead of list_call. If the command lists of the called indexed text strings (or characters) contain no absolute commands, then there is no difference between mark_time_abs and mark_time. 				
RTC4→ RTC5	New command.				
References	mark_time				

Ctrl Command	mcbsp_init				
Function	Defines the data delays for transmitting and receiving data by the McBSP/SPI interface (Multi channel Buffered Serial Port, see also page 54).				
Call	<code>mcbsp_init(XDelay, RDelay)</code>				
Parameters	<table> <tr> <td>XDelay</td> <td>Transmission delay as an unsigned 32-bit value. Allowed range: [0 ... 2].</td> </tr> <tr> <td>RDelay</td> <td>Receiving delay as an unsigned 32-bit value. Allowed range: [0 ... 2]</td> </tr> </table>	XDelay	Transmission delay as an unsigned 32-bit value. Allowed range: [0 ... 2].	RDelay	Receiving delay as an unsigned 32-bit value. Allowed range: [0 ... 2]
XDelay	Transmission delay as an unsigned 32-bit value. Allowed range: [0 ... 2].				
RDelay	Receiving delay as an unsigned 32-bit value. Allowed range: [0 ... 2]				
Comments	<ul style="list-style-type: none"> For invalid parameter values the command is <i>not</i> executed (get_last_error return code RTC5_PARAM_ERROR). The initialized values (after program start) are XDelay = RDelay = 1. The signals and operating conditions of the McBSP/SPI interface are presented in chapter 4.4.6 "SPI / I2C Socket Connector", page 54. 				
RTC4→ RTC5	New command.				
References	read_mcbsp , set_mcbsp_out , set_mcbsp_out_ptr , set_mcbsp_freq , mcbsp_init_spi				



Ctrl Command	mcbsp_init_spi
Function	Initializes the SPI (Serial Peripheral Interface) functionality at the "SPI / I ² C" socket connector (instead of McBSP functionality).
Call	<code>mcbsp_init_spi (ClockLevel, ClockDelay)</code>
Parameters	<p><code>ClockLevel</code> =0: inactive low. >0: inactive high. Unsigned 32-bit value.</p> <p><code>ClockDelay</code> =0: Clock signal and data bits at the same time. >0: Clock signal is delayed a half period. Unsigned 32-bit value.</p>
Comments	<ul style="list-style-type: none"> See chapter 4.4.6 "SPI / I²C Socket Connector", page 54. With the SPI protocol a common CLOCK signal, data input and data output occur simultaneously. The RTC5 is the SPI master. mcbsp_init_spi can be called at any time. Data transmissions in progress (started but not yet finished) are cancelled. The clock frequency can be set in advance by set_mcbsp_freq. mcbsp_init can be used to reestablish the McBSP functionality at the "SPI / I²C" socket connector at any time. Irrespective of the clock frequency the SPI functionality only permits a single transmission every 10 µs (for example, see set_mcbsp_in).
RTC4→RTC5	New command.
Version info	Available as of version DLL 538, OUT 538.
References	mcbsp_init , set_mcbsp_freq



Ctrl Command	measurement_status	
Function	Returns the status of a measurement session started by set_trigger or set_trigger4 and the current position of the measurement pointer.	
Call	<code>measurement_status(&Busy, &Pos)</code>	
Returned parameter values	Busy	Measurement status as a pointer to an unsigned 32-bit value. > 0: A measurement session is currently in progress. = 0: No measurement session is currently in progress.
	Pos	current position of the measurement pointer (within the RTC5's measurement storage area) as a pointer to an unsigned 32-bit value ($0 \leq \text{Pos} \leq 2^{20}-1$ or $0 \leq \text{Pos} \leq (2^{19}-1)$ or $\text{Pos} = 2^{32}-1$, see below).
Comments	<ul style="list-style-type: none"> If a measurement session started with set_trigger or set_trigger4 is no longer active, then <code>Pos+1</code> indicates the number of recorded data pairs up to termination (maximum 2^{20} for set_trigger, maximum 2^{19} for set_trigger4). $\text{Pos} = 2^{32}-1$ indicates that data recording still has not occurred after load_program_file. Stored data can be queried with get_waveform. The status of a measurement session is reset by set_trigger(<code>Period = 0</code>), set_trigger4(<code>Period = 0</code>), stop_trigger or stop_execution (see set_trigger comments). 	
RTC4→RTC5	Unchanged functionality.	
Version info	Last change with version DLL 516, OUT 515.	
References	set_trigger , set_trigger4	



Normal List Command	micro_vector_abs
Function	Moves the output point (of the laser focus) by a hard jump (without microvectorization) directly from the current position to the specified position (absolute coordinate values) within a two-dimensional image field.
Call	<code>micro_vector_abs(X, Y, LasOn, LasOff)</code>
Parameters	<p>X, Y Absolute coordinates of the micro vector end point in <i>bits</i> as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</p> <p>LasOn, LasOff LaserOn delay and LaserOff delay as signed 32-bit values. <i>1 bit equals 0.5 µs.</i> Allowed range (as with set_laser_delays): [-2³¹ ... +(2¹⁵-1)]. ≥ 0: Delay is newly set (values over (2¹⁵-1) are clipped). < 0: The previously defined delay continues unaffected.</p>
Comments	<ul style="list-style-type: none"> For information on microvector command usage, see also page 221. During the conversion of specified coordinate values into scan system output values, any previously defined coordinate transformations, assigned correction tables etc. are taken into account after successful microvectorization (see page 221). The microvector always executes as a hard jump. By <code>LasOn ≥ 0</code> and <code>LasOff ≥ 0</code>, you can set a new LaserOn or LaserOff delay for each individual microvector. Each delay thereby gets set at the end of the clock period in which the new position actually gets outputted (this output clock cycle might be delayed by a preceding scanner delay). Negative values (<code>LasOn < 0</code> and <code>LasOff < 0</code>) do not affect laser delays. Hereby, the laser can remain on or off across multiple clock periods (mark and jump simulation). Delays set with <code>LasOn</code> and <code>LasOff</code> only apply to the execution of microvectors. For execution of normal mark and arc commands (such as mark_abs), only the laser delays defined by set_laser_delays apply. <code>LasOn</code> and <code>LasOff</code> do not overwrite the laser delay parameter from set_laser_delays. The optional XY2-100 converter (see page 43) introduces a 10 µs runtime latency to scan-system control. This runtime latency can be compensated by increasing the LaserOn delay and LaserOff delay by 10 µs each.
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 and the specified delay values by 2. The allowed ranges of values are correspondingly reduced.
Version info	Last change with version DLL 536, OUT 536: <code>LasOn, LasOff < 2¹⁵</code> .
References	micro_vector_rel , micro_vector_abs_3d



Normal List Command	micro_vector_abs_3d
Function	Moves the output point (of the laser focus) by a hard jump (without microvectorization) directly from the current position to the specified position (absolute coordinate values) within a three-dimensional process volume.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as micro_vector_abs .
Call	<code>micro_vector_abs_3d(X, Y, Z, LasOn, LasOff)</code>
Parameters	<p>X, Y, Z Absolute coordinates of the micro vector end point in <i>bits</i> as signed 32-bit values. Allowed range:</p> <ul style="list-style-type: none"> For X and Y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped. <p>LasOn, LasOff LaserOn delay and LaserOff delay as signed 32-bit values. <i>1 bit equals 0.5 µs.</i> Allowed range (as with set_laser_delays): [-2³¹ ... +(2¹⁵-1)]. ≥ 0: Delay is newly set (values over (2¹⁵-1) are clipped). < 0: The previously defined delay continues unaffected.</p>
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, this command functions similarly to the micro_vector_abs command (see the comments there). The X and Y axes can be controlled with 20-bit resolution, the Z axis with 16-bit resolution. The RTC5 (in RTC5 mode as well as in RTC4 compatibility mode) automatically upscales Z coordinate values internally to 20-bit values, so that the three dimensions of space can be handled equivalently in terms of the supplied jump speed value. During calculation of the Z output value to the scan system, any previously defined offset to the Z coordinate and focal length is taken into account (see page 141).
RTC4→RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 and the specified delay values by 2 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 and RTC4.
Version info	Available with version DLL 536, OUT 536
References	micro_vector_abs , micro_vector_rel_3d



Normal List Command	micro_vector_rel
Function	Moves the output point (of the laser focus) by a hard jump (without microvectorization) directly from the current position to the specified position (relative coordinate values) within a two-dimensional image field.
Call	<code>micro_vector_rel(dx, dy, LasOn, LasOff)</code>
Parameters	<p><code>dx, dy</code> <i>Relative coordinates of the micro vector end point in bits as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.</i> The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</p> <p><code>LasOn, LasOff</code> LaserOn delay and LaserOff delay (see micro_vector_abs).</p>
Comments	<ul style="list-style-type: none"> The coordinates for the vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to micro_vector_abs (see the comments there). The microvector always executes as a hard jump.
RTC4→ RTC5	New command. RTC4 compatibility mode: see micro_vector_abs .
Version info	Last change with version DLL 536, OUT 536: <code>LasOn, LasOff < 2¹⁵</code> .
References	micro_vector_abs , micro_vector_rel_3d



Normal List Command	micro_vector_rel_3d
Function	Moves the output point (of the laser focus) by a hard jump (without microvectorization) directly from the current position to the specified position (relative coordinate values) within a three-dimensional process volume.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as micro_vector_rel .
Call	<code>micro_vector_rel_3d(dx, dy, dz, LasOn, LasOff)</code>
Parameters	<p><code>dx,dy,dz</code> Relative coordinates of the micro vector end point in <i>bits</i> as signed 32-bit values. Allowed range:</p> <ul style="list-style-type: none"> For x and y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. For z: [-32768 ... 32767]. Out-of-range values are edge-clipped. <p><code>LasOn, LasOff</code> LaserOn delay and LaserOff delay (see micro_vector_abs_3d).</p>
Comments	<ul style="list-style-type: none"> The coordinates for the vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to micro_vector_abs_3d (see the comments there). The microvector always executes as a hard jump.
RTC4→ RTC5	New command. RTC4 compatibility mode: see micro_vector_abs_3d .
Version info	Available with version DLL 536, OUT 536
References	micro_vector_abs_3d , micro_vector_rel

Ctrl Command	move_to
Comments	<ul style="list-style-type: none"> The command move_to is described in the manual "Installation and Operation RTC Step Motor Extension for the RTC4 and RTC5 PC interface boards" (available in English only). The command move_to has been introduced for the extension board "RTC4 STEP MOTOR EXTENSION" (#112097). It is <i>not</i> supported by the extension board RTC5 varioSCAN 40 FLEX Extension (#128683).

Normal List Command	para_jump_abs
Function	Moves the output point (of the laser focus) along a 2D vector at jump speed from the current position to the specified position (absolute coordinate values) within a two-dimensional image field and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Call	<code>para_jump_abs(X, Y, P)</code>
Parameters	<p>X, Y Absolute coordinates of the jump vector end point in <i>bits</i> as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</p> <p>P End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (<i>Ctrl</i> parameter), identical with set_vector_control (<i>Value</i> parameter).</p>
Comments	<ul style="list-style-type: none"> If vector-defined laser control has not been previously activated by set_vector_control, then para_jump_abs behaves like jump_abs (see comments there). The parameter P is then ignored. If vector-defined laser control is activated, then simultaneously with the motion of the output point of the laser focus the signal parameter selected by set_vector_control is linearly varied from the last valid value to P (see "Vector-Defined Laser Control", page 166). There is no abs mechanism for P. P is, if necessary, clipped to the maximum allowed value. If para_jump_abs is used along with position- and/or speed-dependent laser control for the same control parameter, then the current value of P is used as the basis of the 100% value for laser control (see "Vector-Defined Laser Control", page 166).
RTC4 → RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced). The parameter P does not have a RTC4 compatibility mode. The original RTC5 units must be used.
References	jump_abs , set_vector_control , para_jump_abs_3d , para_jump_rel



Normal List Command	para_jump_abs_3d
Function	Moves the output point (of the laser focus) along a 3D vector at jump speed from the current position to the specified position (absolute coordinate values) within a three-dimensional process volume and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as para_jump_abs . However, microvectorization is calculated like a 3D command and hence influences the effective jump speed in the XY plane.
Call	para_jump_abs_3d(X, Y, Z, P)
Parameters	<p>X, Y, Z Absolute coordinates of the jump vector end point in <i>bits</i> as signed 32-bit values. Allowed range:</p> <ul style="list-style-type: none"> • For X and Y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. • For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped. <p>P End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (<i>Ctrl</i> parameter), identical with set_vector_control (<i>Value</i> parameter).</p>
Comments	<ul style="list-style-type: none"> • Except for the additional motion in the third dimension, this command functions similarly to the para_jump_abs command (see the comments there). • Further comments see jump_abs_3d.
RTC4→ RTC5	<p>New command.</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 and RTC4.</p> <p>The parameter P does not have a RTC4 compatibility mode. The original RTC5 units must be used.</p>
References	jump_abs_3d, set_vector_control, para_jump_abs, para_jump_rel_3d



Normal List Command	para_jump_rel
Function	Moves the output point (of the laser focus) along a 2D vector at jump speed from the current position to the specified position (relative coordinate values) within a two-dimensional image field and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Call	para_jump_rel(dX, dY, P)
Parameters	<p>dX, dY <i>Relative coordinates of the jump vector end point in bits as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.</i> <i>The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</i></p> <p>P <i>End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (Ctrl parameter), identical with set_vector_control (Value parameter).</i></p>
Comments	<ul style="list-style-type: none"> • The coordinates for the jump vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to para_jump_abs (see the comments there). • P is not treated on a relative basis.
RTC4→ RTC5	<p>New command.</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).</p> <p>The parameter P does not have a RTC4 compatibility mode. The original RTC5 units must be used.</p>
References	jump_rel, set_vector_control, para_jump_abs, para_jump_rel_3d



Normal List Command	para_jump_rel_3d
Function	Moves the output point (of the laser focus) along a 3D vector at jump speed from the current position to the specified position (relative coordinate values) within a three-dimensional process volume and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as para_jump_rel . However, microvectorization is calculated like a 3D command and hence influences the effective jump speed in the XY plane.
Call	para_jump_rel_3d(dx, dy, dz, P)
Parameters	<p>dx, dy, dz Relative coordinates of the jump vector end point in <i>bits</i> as signed 32-bit values. Allowed range:</p> <ul style="list-style-type: none"> • For X and Y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. • For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped. <p>P End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (<i>ctrl</i> parameter), identical with set_vector_control (<i>value</i> parameter).</p>
Comments	<ul style="list-style-type: none"> • The coordinates for the jump vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to para_jump_abs_3d (see the comments there). • P is not treated on a relative basis.
RTC4→ RTC5	<p>New command.</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 and RTC4.</p> <p>The parameter P does not have a RTC4 compatibility mode. The original RTC5 units must be used.</p>
References	jump_rel_3d, set_vector_control, para_jump_abs_3d, para_jump_rel



Variable List Command	para_laser_on_pulses_list						
Function	Turns on the LASERON laser control signal for the specified number of external signal pulses (but for no longer than the specified time interval) and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.						
Call	<code>para_laser_on_pulses_list(Period, Pulses, P)</code>						
Parameters	<table> <tr> <td>Period</td> <td>time interval in <i>bits</i> as an unsigned 32-bit value. <i>1 bit equals 10 µs.</i> Allowed range: $0 \leq \text{Period} \leq (2^{32}-1)$.</td> </tr> <tr> <td>Pulses</td> <td>Number of external signal pulses as an unsigned 32-bit value. Allowed range: $0 \leq \text{Pulses} \leq 65535$ or larger (see comments below).</td> </tr> <tr> <td>P</td> <td>End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (<i>Ctrl</i> parameter), identical with set_vector_control (<i>Value</i> parameter).</td> </tr> </table>	Period	time interval in <i>bits</i> as an unsigned 32-bit value. <i>1 bit equals 10 µs.</i> Allowed range: $0 \leq \text{Period} \leq (2^{32}-1)$.	Pulses	Number of external signal pulses as an unsigned 32-bit value. Allowed range: $0 \leq \text{Pulses} \leq 65535$ or larger (see comments below).	P	End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (<i>Ctrl</i> parameter), identical with set_vector_control (<i>Value</i> parameter).
Period	time interval in <i>bits</i> as an unsigned 32-bit value. <i>1 bit equals 10 µs.</i> Allowed range: $0 \leq \text{Period} \leq (2^{32}-1)$.						
Pulses	Number of external signal pulses as an unsigned 32-bit value. Allowed range: $0 \leq \text{Pulses} \leq 65535$ or larger (see comments below).						
P	End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (<i>Ctrl</i> parameter), identical with set_vector_control (<i>Value</i> parameter).						
Comments	<ul style="list-style-type: none"> The command is useful for marking separate points (see page 108). If vector-defined laser control has not been previously activated by set_vector_control, then para_laser_on_pulses_list behaves like laser_on_pulses_list and – if $\text{Pulses} > 65535$ – like laser_on_list (see comments there). The parameter <i>P</i> is then ignored. If vector-defined laser control is activated, then the signal parameter selected by set_vector_control is linearly varied from the last valid value to <i>P</i> within the command's duration ($\text{Period} \times 10 \mu\text{s}$) (see "Vector-Defined Laser Control", page 166). There is no abs mechanism for <i>P</i>. <i>P</i> is, if necessary, clipped to the maximum allowed value. This maximum value is $(2^{31}-1)$ or a lower value depending on what was selected with set_vector_control (<i>Ctrl</i> parameter). If para_jump_abs is used along with position- and/or speed-dependent laser control for the same control parameter, then the current value of <i>P</i> is used as the basis of the 100% value for laser control (see "Vector-Defined Laser Control", page 166). If $\text{Period} = 0$, then the command has no effect and para_laser_on_pulses_list becomes a short list command. If $0 < \text{Period} \leq (2^{31}-1)$, then the command's duration is always <i>Period</i> clocks (i.e. $\text{Period} \times 10 \mu\text{s}$), even if the specified number of external signal pulses expires in a shorter time interval. If $2^{31} \leq \text{Period} \leq (2^{32}-1)$, the command's maximum duration is $(\text{Period} - 2^{31})$ clocks (i.e. $(\text{Period} - 2^{31}) \times 10 \mu\text{s}$). Here, however, the command terminates as soon as the specified number of external signal pulses has been detected. 						
RTC4→ RTC5	New command.						
Version info	Available as of version OUT 526, DLL 524.						
References	laser_on_pulses_list , laser_on_list						



Normal List Command	para_mark_abs
Function	Moves the laser focus at marking speed along a 2D vector from the current position to the specified position (absolute coordinate values) within a two-dimensional image field and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Call	<code>para_mark_abs(X, Y, P)</code>
Parameters	<p>X, Y Absolute coordinates of the mark vector end point in <i>bits</i> as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</p> <p>P End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (<i>Ctrl</i> parameter), identical with set_vector_control (<i>Value</i> parameter).</p>
Comments	<ul style="list-style-type: none"> If vector-defined laser control has not been previously activated by set_vector_control, then para_mark_abs behaves like mark_abs (see comments there). The parameter <i>P</i> is then ignored. If vector-defined laser control is activated, then simultaneously with the laser focus' motion (Sky writing is not taken into account here) the signal parameter selected by set_vector_control is linearly varied from the last valid value to <i>P</i> (see "Vector-Defined Laser Control", page 166). There is no abs mechanism for <i>P</i>. <i>P</i> is, if necessary, clipped to the maximum allowed value. If para_mark_abs is used along with position- and/or speed-dependent laser control for the same control parameter, then the current value of <i>P</i> is used as the basis of the 100% value for laser control (see "Vector-Defined Laser Control", page 166).
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced). The parameter <i>P</i> does not have a RTC4 compatibility mode. The original RTC5 units must be used.
References	mark_abs, set_vector_control, para_mark_abs_3d, para_mark_rel



Normal List Command	para_mark_abs_3d
Function	Moves the laser focus at marking speed along a 3D vector from the current position to the specified position (absolute coordinate values) within a three-dimensional process volume and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as para_mark_abs . However, microvectorization is calculated like a 3D command and hence influences the effective marking speed in the XY plane.
Call	<code>para_mark_abs_3d(X, Y, Z, P)</code>
Parameters	<p>X, Y, Z Absolute coordinates of the mark vector end point in <i>bits</i> as signed 32-bit values. Allowed range:</p> <ul style="list-style-type: none"> • For X and Y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. • For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped. <p>P End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (<i>Ctrl</i> parameter), identical with set_vector_control (<i>Value</i> parameter).</p>
Comments	<ul style="list-style-type: none"> • Except for the additional motion in the third dimension, this command functions similarly to the para_mark_abs command (see the comments there). • Further comments see mark_abs_3d.
RTC4→ RTC5	<p>New command.</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 and RTC4.</p> <p>The parameter P does not have a RTC4 compatibility mode. The original RTC5 units must be used.</p>
References	mark_abs_3d, set_vector_control, para_mark_abs, para_mark_rel_3d



Normal List Command	para_mark_rel
Function	Moves the laser focus at marking speed along a 2D vector from the current position to the specified position (relative coordinate values) within a two-dimensional image field and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Call	<code>para_mark_rel(dx, dy, P)</code>
Parameters	<p>dx, dy <i>Relative coordinates of the mark vector end point in bits as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.</i> <i>The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</i></p> <p>P <i>End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (Ctrl parameter), identical with set_vector_control (Value parameter).</i></p>
Comments	<ul style="list-style-type: none"> • The coordinates for the mark vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to para_mark_abs (see the comments there). • P is not treated on a relative basis.
RTC4→ RTC5	<p>New command. In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced). The parameter P does not have a RTC4 compatibility mode. The original RTC5 units must be used.</p>
References	mark_rel, set_vector_control, para_mark_abs, para_mark_rel_3d



Normal List Command	para_mark_rel_3d
Function	Moves the laser focus at marking speed along a 3D vector from the current position to the specified position (relative coordinate values) within a three-dimensional process volume and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as para_mark_rel . However, microvectorization is calculated like a 3D command and hence influences the effective marking speed in the XY plane.
Call	para_mark_rel_3d(dx, dy, dz, P)
Parameters	<p>dx, dy, dz <i>Relative coordinates of the mark vector end point in bits as signed 32-bit values. Allowed range:</i></p> <ul style="list-style-type: none"> • For x and y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. • For z: [-32768 ... 32767]. Out-of-range values are edge-clipped. <p>P <i>End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (Ctrl parameter), identical with set_vector_control (Value parameter).</i></p>
Comments	<ul style="list-style-type: none"> • The coordinates for the mark vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to para_mark_abs_3d (see the comments there). • P is not treated on a relative basis.
RTC4→ RTC5	<p>New command.</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 and RTC4.</p> <p>The parameter P does not have a RTC4 compatibility mode. The original RTC5 units must be used.</p>
References	mark_rel_3d, set_vector_control, para_mark_abs_3d, para_mark_rel

Variable List Command	park_position
Function	For temporary parking, this command moves the output point (of the laser focus) along a 2D vector at jump speed from the current position to the specified position (absolute coordinate values) within the two-dimensional image field.
Restriction	If the Processing-on-the-fly option is not enabled, then the command functions like a normal jump_abs command.
Call	<code>park_position(Mode, X, Y)</code>
Parameter	<p>Mode Unsigned 32-bit value.</p> <p>= 0: X and Y are interpreted as park position coordinates in the real image field. Allowed range [-524288 ... 524287]. The current Processing-on-the-fly mode is switched off and the laser focus moved at the currently set jump speed to the specified park position in the real image field. During a subsequent list interruption (by wait_for_encoder etc.), the galvanometer scanners remains stationary (even for a Processing-on-the-fly application initiated by set_fly_2d).</p> <p>> 0: X and Y are interpreted as park position coordinates in the virtual image field. Allowed range [-8388608 ... 8388607]. The current Processing-on-the-fly mode remains switched on and the laser focus moved at the currently set jump speed to the specified park position in the virtual image field (subject to Processing-on-the-fly correction and possibly clipped to the real image field's boundaries). During a subsequent list interruption (by wait_for_encoder etc.), the galvanometer scanners' positions are continuously Processing-on-the-fly-corrected in accordance with the current encoder values (even for a Processing-on-the-fly application initiated by set_fly_x/set_fly_y) and possibly clipped to the real image field's boundaries.</p> <p>X, Y Absolute coordinates of the jump vector end point in bits as signed 32-bit values. Allowed range: see above. Out-of-range values are edge-clipped.</p>
Comments	<ul style="list-style-type: none"> The command is intended for encoder-based set_fly_2d or set_fly_x/set_fly_y Processing-on-the-fly applications where the laser focus needs to be moved to a safe parking area during an intermediate motion (prior to list interruption by wait_for_encoder etc.) (see chapter 8.7.7 "Synchronization of Processing-on-the-fly Applications", page 211). For the return jump (away from the park position), use the park_return command (refer to all comments there). If another (or no) Processing-on-the-fly application is active, then park_position functions like a normal jump command, as does the return jump by park_return (but Processing-on-the-fly remains switched off). If the laser focus was already previously moved to a safe park position, then the command is a short list command with no effect. park_position switches off the "laser active" laser control signals after a LaserOff delay, but does not activate a scanner delay afterward.



Variable List Command	park_position
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).
Version info	Available as of version DLL 536, OUT 536.
References	park_return

Variable List Command	park_return								
Function	Moves the output point (of the laser focus) away from a park position along a 2D vector at jump speed to the specified position (absolute coordinate values) within the two-dimensional image field.								
Restriction	If the Processing-on-the-fly option is not enabled, then the command functions as a normal jump command. If the laser focus is not currently in a park position, then the command is a short list command with no effect (see below).								
Call	<code>park_return(Mode, X, Y)</code>								
Parameter	<table> <tr> <td>Mode</td> <td>Unsigned 32-bit value.</td> </tr> <tr> <td>= 0:</td> <td>X and Y are ignored (see comments).</td> </tr> <tr> <td>> 0:</td> <td>X and Y are interpreted as return jump coordinates (see comments).</td> </tr> <tr> <td>X, Y</td> <td>Absolute coordinates of the jump vector end point in the virtual image field in bits as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.</td> </tr> </table>	Mode	Unsigned 32-bit value.	= 0:	X and Y are ignored (see comments).	> 0:	X and Y are interpreted as return jump coordinates (see comments).	X, Y	Absolute coordinates of the jump vector end point in the virtual image field in bits as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.
Mode	Unsigned 32-bit value.								
= 0:	X and Y are ignored (see comments).								
> 0:	X and Y are interpreted as return jump coordinates (see comments).								
X, Y	Absolute coordinates of the jump vector end point in the virtual image field in bits as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.								
Comments	<ul style="list-style-type: none"> The command is intended for encoder-based set_fly_2d or set_fly_x/set_fly_y Processing-on-the-fly applications where the laser focus should leave a safe parking area previously reached by park_position after an intermediate motion (following list interruption by wait_for_encoder etc.) (see chapter 8.7.7 "Synchronization of Processing-on-the-fly Applications", page 211, see also comments at park_position). If a set_fly_2d or set_fly_x/set_fly_y Processing-on-the-fly mode was switched off by a prior park_position(Mode = 0) call, then park_return switches the same Processing-on-the-fly mode back on. Other Processing-on-the-fly modes are not switched back on. If the park position was attained by park_position, then park_return(Mode = 0) returns the galvanometer scanners to the most recent valid position before park_position was called, whereas park_return(Mode = 1) moves them to the position specified with the command. When calculating the new position, the RTC5 takes the current Processing-on-the-fly correction into account. But if clipping to the boundaries of the virtual image field occurs (e.g. due to a too long intermediate motion during a list interruption by wait_for_encoder), then the Processing-on-the-fly mode is not reactivated (see also activate_fly_2d and activate_fly_xy) and the jump executes without Processing-on-the-fly correction (and possibly clipped to the real image field's boundaries). 								



Variable List Command	park_return
Comments (cont'd)	<ul style="list-style-type: none"> If the laser focus was <i>not</i> previously moved to a safe area by park_position, then park_return is a short list command with no further effect. If no Processing-on-the-fly mode was previously active or if any Processing-on-the-fly mode is currently active, then park_return performs a normal jump to the specified location. If a coordinate transformation in the virtual image field is active, then the specified or most recent valid position is subsequently appropriately transformed (see page 208). A scanner jump delay is activated after the park_return command. park_return switches off the "laser active" laser control signals after a LaserOff delay.
RTC4→ RTC5	<p>New command. In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).</p>
Version info	Available as of version DLL 536, OUT 536.
References	park_position



Ctrl Command	pause_list
Function	Pauses execution of the list and disables the "laser active" laser control signals.
Call	<code>pause_list()</code>
Comments	<ul style="list-style-type: none"> • pause_list is synonymous with the previous command stop_list, which is often confused with stop_execution. Therefore, it is preferable to use pause_list. • If pause_list is called during execution of a list, the command causes the "laser active" laser control signals to be disabled (then the laser output ports are in the high impedance tristate mode) and keeps the scan system in the most recently defined state – even if in the middle of microvectoring. The PAUSED status (queryable by get_status) is set, but the BUSY status is left unchanged. Continuation by execute_list_pos or release_wait or by an external start is suppressed. However, stop_execution or an external list stop is possible. restart_list, stop_execution or an external list stop ends suppression of the list start. • If processing of a list should be continued, then the command restart_list must be used. After the successful restart_list command, the scan system resumes the planned movements (of the current command) and the "laser active" laser control signals are reenabled (the laser is started with the standard laser parameters, i.e. without any soft-start or similar parameters that might have been defined prior to pause_list). The PAUSED status is then reset (here too, BUSY remains unchanged). • If, during calling of the pause_list command, no list is currently executing (BUSY status not set) or a list has already been halted by pause_list or set_wait (PAUSED status set), then the pause_list command is ignored (get_last_error return code RTC5_BUSY; the laser is then already off).
RTC4→RTC5	New command.
References	restart_list , set_wait

Ctrl Command	periodic_toggle
Function	Same as periodic_toggle_list , but a control command.
Call	<code>periodic_toggle (Port, Mask, P1, P2, Count, Start)</code>
Parameters	See periodic_toggle_list .
Comments	<ul style="list-style-type: none"> • See periodic_toggle_list. • If an unallowed parameter value is supplied for <code>Port</code> or 0 for <code>P1</code> or <code>P2</code>, then the command is not sent and a get_last_error return code RTC5_PARAM_ERROR is generated. • See also chapter 9.4 "Periodical I/O Signals", page 249.
RTC4→RTC5	New command.
Version info	Last change with version DLL 543, OUT 543: endless toggling with <code>Count = 2³²-1</code> .
References	periodic_toggle_list



Undelayed Short List Command	periodic_toggle_list
Function	Generates and periodically toggles a signal at an adjustable output port.
Call	<code>periodic_toggle_list (Port, Mask, P1, P2, Count, Start)</code>
Parameters	<p>Port Output port (0–4, as with set_port_default).</p> <p>Mask Defines the bits to be toggled, (the maximum value depends on the port). 1 = bit is toggled. 0 = bit remains unchanged.</p> <p>P1, P2 Duration of the toggled signal in [10 µs] (>0, 16 bit max.).</p> <p>Count Number of P1–P2 period repetitions.</p> <p>Start Duration until the first toggling starts in [10 µs].</p> <p>All parameters are unsigned 32-bit values.</p>
Comments	<ul style="list-style-type: none"> If an unallowed parameter value is supplied for <code>Port</code> or 0 for <code>P1</code> or <code>P2</code>, then the command is replaced by a list_nop and a get_last_error return code <code>RTC5_PARAM_ERROR</code> is generated. Mask defines the bits to be toggled. Bits which are set to 1 in Mask are toggled and bits which are set to 0 remain unchanged. Mask is limited to the maximum allowed value of the selected port (see also set_port_default). Extra bits are ignored. The selected bits are toggled. This state is maintained for $P1 \times 10 \mu s$ clock periods. Then they are toggled once again and kept stable for $P2 \times 10 \mu s$ clock periods. Users define the toggle bit start values ("off" state; "on" signal is active-high or active-low) by other standard commands (write_da_x, write_8bit_port, write_io_port, etc.). These – and (should the situation arise) other queued delayed short list commands – are executed before the command periodic_toggle_list. The first toggling occurs after <code>Start</code> $\times 10 \mu s$ clock periods. Toggling is repeated <code>Count</code>-times. <code>Count</code> = 0 immediately stops an output in progress. The remaining parameters are then not relevant. If the stop happens in the <code>P1</code> period ("On"), a toggle occurs to restore the initial state ("Off"). The parameters <code>P1</code> and <code>P2</code> are clipped to the value range [0...65535]. <code>P1</code> and <code>P2</code> must not be 0 at the same time. The selected port should not concurrently be used for other outputs such as automatic laser control. At a set_end_of_list, stop_execution or external /STOP the periodical signals continue. As of version DLL 543, OUT 543, periodic_toggle_list toggles endless with <code>Count</code> = $2^{32}-1$. See also chapter 9.4 "Periodical I/O Signals", page 249.
RTC4→ RTC5	New command.
Version info	Last change with version DLL 543, OUT 543: endless toggling with <code>Count</code> = $2^{32}-1$.
References	periodic_toggle , set_port_default



Ctrl Command	quit_loop
Function	Stops the repeating automatic list change started with start_loop .
Call	<code>quit_loop()</code>
Comments	<ul style="list-style-type: none"> Before list execution is stopped, the current list fully executes until the next set_end_of_list command is encountered. If no start_loop command was previously issued, then quit_loop has no effect.
RTC4→ RTC5	Unchanged.
References	start_loop

Undelayed Short List Command	range_checking	
Function	Defines an emergency action if a galvanometer exceeds its position limit.	
Call	<code>range_checking (HeadNo, Mode, Data)</code>	
Parameter	HeadNo	Scan head to be monitored as an unsigned 32-bit value. 0: No monitoring (default after load_program_file). 1: Primary scan head is monitored. 2: Secondary scan head is monitored. 3: Both scan heads are monitored. Only the 2 least significant bits are evaluated.
	Mode	Action to take as an unsigned 32-bit value. 0: Laser is switched-off immediately and list execution continues. 1: Laser is switched-off immediately and list execution is stopped immediately (as with stop_execution or /STOP).
	Data	Data type chosen to be monitored as an unsigned 32-bit value. 0: Sample values (Processing-on-the-fly and Wobbel corrected, as with set_trigger signal types 7–9). 1: Transformed control values (with scan head-specific transformations, as with set_trigger signal types 25–30). 2: Corrected control values (with correction file, as with set_trigger signal types 10–15). 3: Actual output values (with "Offset" and "Gain", as with set_trigger signal types 20–23). 4: Real position of galvanometer (only with iDRIVE systems, varioSCAN is internally connected to an X axis, as with set_trigger signal types 1, 2 and 3). 5: Real position of galvanometer (only with iDRIVE systems, varioSCAN is internally connected to an Y axis, as with set_trigger signal types 1, 2 and 4).



Undelayed Short List Command	range_checking
Comments	<ul style="list-style-type: none"> No monitoring takes place if the specified scan head does not have a correction table assigned (see select_cor_table). The used position limits (that is, if exceeded the emergency action is executed) are the parameters of a customer-specific Processing-on-the-fly application monitoring (see set_fly_limits, set_fly_limits_z). Exceeding these limits only cause error messages in case of Processing-on-the-fly applications. The error messages can be queried with get_marking_info or the respective if_fly_<axis>_overflow/if_not_fly_<axis>_overflow commands. They do not trigger any activities. Processing-on-the-fly applications use the data type Data = 0 (sample values) for customer-specific range limits. Inconsistent error messages and switch-offs may occur if used simultaneously with range_checking data types Data > 0. The range_checking monitoring is active without Processing-on-the-fly application. Data > 5 causes a get_last_error RTC5_PARAM_ERROR error code. In this case the command is sent to the RTC5 Board as list_nop. Data=2 and Data=3 have the identical effect for the Z axis. For Data = 4 and Data = 5 users must define the set position as the to be returned data type. A simultaneous use of a speed-dependent laser control with Mode = 2 (see set_auto_laser_control) is not possible. Data = 4 and Data = 5 only differ in regards to the axis onto an optional varioSCAN is connected. For 2D systems without varioSCAN both selections have the identical effect. Data = 4 und Data = 5 may produce nonsensical switch-offs if an intelliSCAN is connected but is either not switched-on or a real position has not been set as feedback. If the laser has been switched-off with Mode = 0 and the fault condition is gone then the laser is <i>not</i> switched on until the next marking command. The laser is <i>not</i> switched-on right in the middle of a currently executing marking command.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 537, OUT 537.
References	set_fly_limits , set_fly_limits_z



Ctrl Command	read_abc_from_file	
Function	Reads the ABC values directly from a specified correction file on the PC.	
Call	ErrorNo = read_abc_from_file(Name, &A, &B, &C)	
Result	ErrorNo Error code as an unsigned 32-bit value: 0 No error. 1 File error (corrupt or incomplete). 2 Memory error (DLL-internal, Windows working memory). 3 File-open error (empty string, file not found etc.).	
Parameter	Name	Correction file name as a pointer to a null-terminated ANSI string.
Returned parameter values	A, B, C	Coefficients (as 64-bit IEEE floating point values) of the parabolic function $z_{out} = A + Bz + Cz^2$ which is used for calculating the Z output values.
Comments	<ul style="list-style-type: none"> • The command read_abc_from_file is also available without explicit access rights to a specific RTC5 Board. • The command read_abc_from_file is not available as a multi-board command. • The board-specific error variables <code>LastError</code> and <code>AccError</code> (see chapter 6.8 "Error Handling", page 98) are neither generated nor altered by read_abc_from_file. 	
RTC4→RTC5	New command.	
Version info	Available as of version DLL 538, OUT 538.	
References	wwrite_abc_to_file	



Ctrl Command	read_analog_in
Function	Reads the analog input values (ANALOG IN0 and ANALOG IN1) from the optional ADC add-on board of the RTC5 PCI Board or, from the "SPI / I2C" socket connector of the RTC5 PCI Express board and returns them as 12-bit digital values.
Call	AnalogValue = read_analog_in()
Result	<p>An unsigned 32-bit value:</p> <p>Bits#0...11 ANALOG IN0 input value as 12-bit digital value. Bit#12 = 0 (channel number). Bits#13...14 = 0. Bit#15 Old bit for ANALOG IN0. Bits#16...27 ANALOG IN1 input value as 12-bit digital value. Bit#28 = 1 (channel number). Bits#29...30 = 0. Bit#31 Old bit for ANALOG IN1.</p>
Comments	<ul style="list-style-type: none"> • read_analog_in can only be used with RTC5 PCI Express boards and RTC5 PCI Boards having an ADC add-on board installed (see chapter 4.4.8 "Analog Inputs (Optional Accessory)", page 59 and chapter 16.2.3 "McBSP/SPI Interface and Analog Inputs", page 676). • read_analog_in cannot be used with RTC5 PC/104-Plus Boards and RTC5 PCIe/104 Boards. • The old bits (bits #15 and #31) are set to 1 after reading by read_analog_in. They are automatically reset, when new data are available at the corresponding analog input. • As of version DLL 543, OUT 543, RBF 524 the analog input values (ANALOG IN0 and ANALOG IN1) can be recorded with trigger signal 54 (see set_trigger or set_trigger4). However, old bits are not recorded. The format of the data is ((ANALOG IN1 << 16) + ANALOG IN0).
RTC4→RTC5	New command.
Version info	Available as of version DLL 527, OUT 529, RBF 519.
References	set_trigger , set_trigger4



Ctrl Command	read_encoder
Function	Returns the counts of the two RTC5 encoder counters that were stored by store_encoder .
Call	<code>read_encoder(&Encoder0_0, &Encoder1_0, &Encoder0_1, &Encoder1_1)</code>
Returned parameter values	<p><code>Encoder0_0, Encoder1_0, Encoder0_1, Encoder1_1</code> Counts as pointers to signed 32-bit values.</p> <p>For parameter “<code>Encoder_n_m</code>”:</p> <ul style="list-style-type: none"> • n is the number of the encoder counter (Encoder0, Encoder1). • m is the number of the buffer position (parameter <code>Pos</code> of store_encoder).
Comments	<ul style="list-style-type: none"> • See also “Processing-on-the-fly (Optional)”, page 199 and “Synchronization by Encoder Signals”, page 246. • If incremental encoders are used to detect the motion of the parts to be processed, encoder counter Encoder0 is triggered by the signals at encoder input ENCODER X and encoder counter Encoder1 by the signals at encoder input ENCODER Y. In contrast, if an encoder simulation has been started by simulate_encoder, both encoder counters are triggered by an internal periodic 1 MHz clock signal. • For buffer positions in which counts were not previously stored by store_encoder, the value 0 is returned (initialized value).
RTC4→ RTC5	New command.
Version info	Available as of version DLL 520, OUT 519.
References	store_encoder , get_encoder , set_fly_x , set_fly_y , set_fly_rot , wait_for_encoder

Ctrl Command	read_io_port
Function	Returns the current state of the 16-bit digital input port on the EXTENSION 1 socket connector.
Call	<code>IOPort = read_io_port()</code>
Result	16-bit value (DIGITAL IN0 ... DIGITAL IN15) as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> • See also chapter 9.2.1 “16-Bit Digital Input”, page 238.
RTC4→ RTC5	Unchanged functionality.
References	write_io_port , write_io_port_mask , set_io_cond_list , clear_io_cond_list , read_io_port_buffer , read_io_port_list



Ctrl Command	read_io_port_buffer				
Function	Returns the data previously stored in the IOPort buffer by read_io_port_list (input value read at the EXTENSION 1 socket connector's 16-bit digital input; the galvanometer set position and time value that was current at the time of reading).				
Call	CurrentIndex = read_io_port_buffer(Index, &Value, &XPos, &YPos, &Time)				
Parameter	Index	Number of the to-be-returned entry in the IOPort buffer as an unsigned 32-bit value. Only the 4 least significant bits are evaluated (the IOPort buffer only has sixteen entries). Therefore, the user program can use a continuous counter for the index.			
Returned parameter values	Value	The 16-bit value stored in the IOPort buffer under <code>Index</code> as a pointer to an unsigned 32-bit value.			
	XPos, YPos	The galvanometer scanner set positions (X and Y values) stored in the IOPort buffer under <code>Index</code> as pointers to signed 32-bit values.			
	Time	The time in seconds stored in the IOPort buffer under <code>Index</code> (as a pointer to an unsigned 32-bit value).			
Result	The index to which data is written upon the next read_io_port_list , as an unsigned 32-bit value.				
Comments	<ul style="list-style-type: none"> See also comments for read_io_port_list. If no read_io_port_list command has been called before read_io_port_buffer, then the initialization values (default: 0) are returned. 				
Example (C/C++)	<p>Wait until the data under <code>Index</code> gets newly written:</p> <pre>while (Index == read_io_port_buffer(Index, &Value, &XPos, &YPos, &Time));</pre> <p>Read all data from <code>Index</code> to the current (not yet newly written) read position:</p> <pre>while (Index != read_io_port_buffer(Index, &Value, &XPos, &YPos, &Time)) { Index = (Index+1) & 0xf; ...}</pre>				
RTC4→ RTC5	New command.				
References	read_io_port_list				



Undelayed Short List Command	read_io_port_list
Function	Writes into the internal IOPort buffer the current state (DIGITAL IN0 ... DIGITAL IN15) of the EXTENSION 1 socket connector's 16-bit digital input. At the same time, the current XY set positions and (relative) time in seconds are stored.
Call	<code>read_io_port_list()</code>
Comments	<ul style="list-style-type: none"> The read_io_port_list command does not return values. However, the values can be subsequently queried from the IOPort buffer by the read_io_port_buffer command. The IOPort buffer holds 16 entries and is circularly organized. It always stores the 16 most recent entries and overwrites older entries without warning. The incremental Index starts after a reset (program start) at Index 0, and after passing 15 does rollover to 0. The stored time is the current value of an internal seconds counter that is set to 0 at program start (load_program_file) or by time_update. See also chapter 9.2.1 "16-Bit Digital Input", page 238.
RTC4→ RTC5	New command.
References	read_io_port_buffer , read_io_port



Ctrl Command	read_mcbsp
Function	Returns the most recent input value that was fully copied to an internal memory location by the McBSP/SPI interface.
Call	<code>mcbsp_value = read_mcbsp(No)</code>
Parameter	<p>No Number (unsigned 32-bit value) of the internal memory location whose input value should be queried.</p> <ul style="list-style-type: none"> = 0: Memory location for Processing-on-the-fly applications and for set_mcbsp_in input with coding bit#31 = 0. = 1, 2: Memory locations for online positioning and for set_mcbsp_in input. = 3: Memory location for set_mcbsp_in input with coding bit#31 = 1. For set_multi_mcbsp_in, the following applies: memory locations 0 through 3 are continuously written to as a ring buffer. Only the two least significant bits are evaluated.
Result	Input value as a signed 32-bit value.
Comments	<ul style="list-style-type: none"> • For information on the McBSP/SPI interface (Multi channel Buffered Serial Port) and the related memory locations, see also page 54. • It is up to users whether to interpret the returned value as one signed or unsigned 32-bit data word, as two signed 16-bit data words or as two signed 15-bit data words: <ul style="list-style-type: none"> – For Processing-on-the-fly applications (<code>No = 0</code>) see page 199 and page 248. – For online positioning (<code>No = 1...2</code>) see page 187. – For set_mcbsp_in input (<code>No = 0...3</code>) see page 203 and page 205. – For set_multi_mcbsp_in, see page 526. • After a reset by load_program_file, the input values at the McBSP/SPI interface are transferred to location 0 (default) even if no Processing-on-the-fly correction is activated. • The McBSP/SPI interface always ignores the first FrameSync signal that follows a load_program_file or mcbsp_init, so any available data is not transferred (see page 56).
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> • Available as of version OUT 526, DLL 524. • Last change with version DLL 531, OUT 532: <code>No = 3</code> and set_mcbsp_in input.
References	get_mcbsp , get_mcbsp_list , set_mcbsp_out , mcbsp_init , set_fly_x_pos , set_fly_y_pos , set_fly_rot_pos , set_mcbsp_x , set_mcbsp_y , set_mcbsp_rot , set_mcbsp_matrix , apply_mcbsp



Ctrl Command	read_multi_mcbsp
Function	Queries the McBSP/SPI multi transmission input value that was most recently type-sorted and stored.
Call	<code>mcbsp_value = read_multi_mcbsp(No)</code>
Parameter	No Number of the type-sorted internal memory location as an unsigned 32-bit value (No corresponds to coding bits #0 - 2 of the McBSP/SPI multi input). 0 = memory location for the Processing-on-the-fly application's X coordinate. 1 = memory location for the Processing-on-the-fly application's Y coordinate. 2 = memory location for the Processing-on-the-fly application's Z coordinate. 3 = memory location for the laser power P. 4 = memory location for extra parameter E1. 5 = memory location for extra parameter E2. 6 = memory location for extra parameter E3. 7 = memory location for extra parameter E4. Only the three least significant bits are evaluated.
Result	Memory content as a signed 32-bit value.
Comments	<ul style="list-style-type: none"> You must have previously activated and used McBSP/SPI multi transmission by set_multi_mcbsp_in or set_multi_mcbsp_in_list. Otherwise, default or old values are returned.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 537, OUT 537.
References	set_multi_mcbsp_in , set_multi_mcbsp_in_list



Ctrl Command	read_status																				
Function	Returns the RTC5 list status, see chapter 6.4.2 "List Status", page 76.																				
Call	Status = read_status()																				
Result	<p>List status as an unsigned 32-bit value:</p> <table> <thead> <tr> <th>Bit #</th> <th>Name</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Bit #0 (LSB)</td> <td>LOAD1</td> <td>= 1: indicates that the input pointer is currently in "List 1", i.e. that all following list commands are stored in "List 1". LOAD1 is set if the input pointer is set into "List 1" (e.g. by set_start_list_pos). LOAD1 is reset if a set_end_of_list command is written to "List 1" (READY1 set) or if LOAD2 is set (e.g. by set_start_list_pos).</td> </tr> <tr> <td>Bit #1</td> <td>LOAD2</td> <td>= 1: indicates that the input pointer is currently in "List 2", i.e. that all following list commands are stored in "List 2". LOAD2 is set if the input pointer is set into "List 2" (e.g. by set_start_list_pos). LOAD2 is reset if a set_end_of_list command is written to "List 2" (READY2 set) or if LOAD1 is set (e.g. by set_start_list_pos).</td> </tr> <tr> <td>Bit #2</td> <td>READY1</td> <td>= 1: indicates that during the loading procedure a set_end_of_list command was written to "List 1". READY1 is reset if LOAD1 is newly set (e.g. by set_start_list_pos).</td> </tr> <tr> <td>Bit #3</td> <td>READY2</td> <td>= 1: indicates that during the loading procedure a set_end_of_list command was written to "List 2". READY2 is reset if LOAD2 is newly set (e.g. by set_start_list_pos).</td> </tr> <tr> <td>Bit #4</td> <td>BUSY1</td> <td>= 1: indicates that "List 1" is executing at the moment (or more precisely: that the output pointer currently resides in "List 1" after execution of "List 1" or "List 2" was started). BUSY1 is set by starting execution of "List 1" (e.g. by execute_list_pos) or by a list change to "List 1" (automatic list change or jump). BUSY1 is reset if set_end_of_list is executed in "List 1", if a jump into "List 2" is executed (e.g. list_jump_pos), if "List 2" is started (e.g. by execute_list_pos) or if stop_execution is executed.</td> </tr> </tbody> </table>			Bit #	Name	Description	Bit #0 (LSB)	LOAD1	= 1: indicates that the input pointer is currently in "List 1", i.e. that all following list commands are stored in "List 1". LOAD1 is set if the input pointer is set into "List 1" (e.g. by set_start_list_pos). LOAD1 is reset if a set_end_of_list command is written to "List 1" (READY1 set) or if LOAD2 is set (e.g. by set_start_list_pos).	Bit #1	LOAD2	= 1: indicates that the input pointer is currently in "List 2", i.e. that all following list commands are stored in "List 2". LOAD2 is set if the input pointer is set into "List 2" (e.g. by set_start_list_pos). LOAD2 is reset if a set_end_of_list command is written to "List 2" (READY2 set) or if LOAD1 is set (e.g. by set_start_list_pos).	Bit #2	READY1	= 1: indicates that during the loading procedure a set_end_of_list command was written to "List 1". READY1 is reset if LOAD1 is newly set (e.g. by set_start_list_pos).	Bit #3	READY2	= 1: indicates that during the loading procedure a set_end_of_list command was written to "List 2". READY2 is reset if LOAD2 is newly set (e.g. by set_start_list_pos).	Bit #4	BUSY1	= 1: indicates that "List 1" is executing at the moment (or more precisely: that the output pointer currently resides in "List 1" after execution of "List 1" or "List 2" was started). BUSY1 is set by starting execution of "List 1" (e.g. by execute_list_pos) or by a list change to "List 1" (automatic list change or jump). BUSY1 is reset if set_end_of_list is executed in "List 1", if a jump into "List 2" is executed (e.g. list_jump_pos), if "List 2" is started (e.g. by execute_list_pos) or if stop_execution is executed.
Bit #	Name	Description																			
Bit #0 (LSB)	LOAD1	= 1: indicates that the input pointer is currently in "List 1", i.e. that all following list commands are stored in "List 1". LOAD1 is set if the input pointer is set into "List 1" (e.g. by set_start_list_pos). LOAD1 is reset if a set_end_of_list command is written to "List 1" (READY1 set) or if LOAD2 is set (e.g. by set_start_list_pos).																			
Bit #1	LOAD2	= 1: indicates that the input pointer is currently in "List 2", i.e. that all following list commands are stored in "List 2". LOAD2 is set if the input pointer is set into "List 2" (e.g. by set_start_list_pos). LOAD2 is reset if a set_end_of_list command is written to "List 2" (READY2 set) or if LOAD1 is set (e.g. by set_start_list_pos).																			
Bit #2	READY1	= 1: indicates that during the loading procedure a set_end_of_list command was written to "List 1". READY1 is reset if LOAD1 is newly set (e.g. by set_start_list_pos).																			
Bit #3	READY2	= 1: indicates that during the loading procedure a set_end_of_list command was written to "List 2". READY2 is reset if LOAD2 is newly set (e.g. by set_start_list_pos).																			
Bit #4	BUSY1	= 1: indicates that "List 1" is executing at the moment (or more precisely: that the output pointer currently resides in "List 1" after execution of "List 1" or "List 2" was started). BUSY1 is set by starting execution of "List 1" (e.g. by execute_list_pos) or by a list change to "List 1" (automatic list change or jump). BUSY1 is reset if set_end_of_list is executed in "List 1", if a jump into "List 2" is executed (e.g. list_jump_pos), if "List 2" is started (e.g. by execute_list_pos) or if stop_execution is executed.																			



Ctrl Command	read_status			
Result (cont'd)	Bit #5	BUSY2	= 1:	indicates that "List 2" is executing at the moment (or more precisely: that the output pointer currently resides in "List 2" after execution of "List 1" or "List 2" was started). BUSY2 is set by starting execution of "List 2" (e.g. by <code>execute_list_pos</code>) or by a list change to "List 2" (automatic list change or jump). BUSY2 is reset if <code>set_end_of_list</code> is executed in "List 2", if a jump into "List 1" is executed (e.g. <code>list_jump_pos</code>), if "List 1" is started (e.g. by <code>execute_list_pos</code>) or if <code>stop_execution</code> is executed.
	Bit #6	USED1	= 1:	indicates that a <code>set_end_of_list</code> command was reached during processing of "List 1". USED1 is reset when LOAD1 is set (e.g. by <code>set_start_list_pos</code>).
	Bit #7	USED2	= 1:	indicates that a <code>set_end_of_list</code> command was reached during processing of "List 2". USED2 is reset when LOAD2 is set (e.g. by <code>set_start_list_pos</code>).
	Bits #8...31		1	
Comments	<ul style="list-style-type: none"> When interpreting the status values returned by <code>read_status</code>, always take into account the programmed loading or execution processes of the lists. Under some circumstances, the status values can be misleading, as illustrated by the following examples: <ul style="list-style-type: none"> Even during a loading process, a list's LOAD status can already have been reset and its READY status set if – after loading of a <code>set_end_of_list</code> command into a list – further list commands are loaded into the same list. The status values remain unchanged if a <code>set_end_of_list</code> command is overwritten with another command (READY is not reset). If – during a loading process – a <code>set_end_of_list</code> command is processed at the same time in the same list, then the list is regarded as already processed (USED status set), even though it is still newly loaded (USED was then initially reset). Even if a completely loaded command list (incl. <code>set_end_of_list</code>) has been stored, a list's READY status can be reset if the input pointer was newly set (e.g. by <code>set_input_pointer</code>) into the list. Then a list's USED status can be reset too, even though a completely loaded and already processed command list is stored. For jumps from one list area to another ("List 1" <-> "List 2", e.g. by <code>list_jump_pos</code>) during execution, the USED status values of both lists (unlike their BUSY status values) remain unchanged and are therefore not meaningful. 			



Ctrl Command	read_status
Comments (cont'd)	<ul style="list-style-type: none"> If the list status is queried during processing of a subroutine in the protected buffer area ("List 3"), then the status is returned of the list ("List 1" or "List 2") in which the output pointer most recently resided (typically from where the subroutine was originally called). If list execution is interrupted (by pause_list, stop_list or set_wait), then the above-mentioned status values remains unchanged. The list execution statuses BUSY and PAUSED (see chapter 6.4.3 "List Execution Status", page 77) can be queried by get_status. To read the status signals from the <i>scan heads</i>, use the command get_head_status.
RTC4→ RTC5	Essentially unchanged functionality, however: The RTC5 also provides a USED status.
References	get_status , get_head_status

Ctrl Command	read_user_data
Function	Queries the status information currently returned from the scan system by the user data bit.
Comments	<ul style="list-style-type: none"> Up to now, this command has no effect, but it is reserved for future use according to chapter 7.3.6 "Status Monitoring and Diagnostics", page 143 and chapter 8.1.2 "Configuring Status Return Behavior", page 173.
References	send_user_data



Ctrl Command	release_rtc
Function	Releases the specified RTC5 for use by other user programs.
Call	NoOfReleasedCard = release_rtc(CardNo)
Parameter	CardNo DLL-internal number of the RTC5 Board as an unsigned 32-bit value.
Result	The returned (unsigned 32-bit) value is CardNo if the user program was still in possession of access rights for this board. Otherwise, 0 is returned.
Comments	<ul style="list-style-type: none"> After the user program releases a board with release_rtc, it no longer has access rights for this board. The board can then be subsequently acquired by the same or another user program by acquire_rtc or init_rtc5_dll. If a board released by release_rtc was the active board, then (other than multi-board commands) only those non-multi-board commands not requiring explicit access rights are subsequently available. Another board is not automatically selected as the active board. Activation of another board (for which access rights are assigned to the user program) can be achieved by the select_rtc command. The release_rtc command does not cause a reset of the RTC5 Board. Board resets can only be initiated by the command load_program_file. The release_rtc command is also available without explicit access rights for a particular RTC5 Board, but the command then has no effect (return value 0). The release_rtc command also has no effect (return value 0) if CardNo exceeds the number of RTC5 Boards found during initialization (see rtc5_count_cards) or if CardNo = 0 (real boards begin at 1). release_rtc is not available as a multi-board command.
RTC4→RTC5	New command.
References	acquire_rtc, init_rtc5_dll



Ctrl Command	release_wait
Function	Resumes execution of a list that was interrupted by a set_wait command.
Call	<code>release_wait()</code>
Comments	<ul style="list-style-type: none"> The command release_wait is only executed if the RTC5 is actually in a wait state (hence if a wait marker was previously reached and execution was halted; the PAUSED status is then set, the BUSY status is <i>not</i> set). Otherwise, the command is ignored (get_last_error return code RTC5_BUSY). release_wait resets the PAUSED status and newly sets the BUSY status (both queriable with get_status, see also "List Execution Status", page 77). The command release_wait resets to zero the wait_word that receives the wait marker number during an interrupt. If a home jump (defined with home_position or home_position_xyz) was executed by set_wait, then release_wait leads to a corresponding home return (the INTERNAL-BUSY status is set while the home return is executed). The wait status can be queried by get_wait_status.
RTC4→RTC5	Essentially unchanged functionality, however: The RTC5 also provides a PAUSED status. It is set with set_wait and reset with release_wait .
References	set_wait, get_wait_status, restart_list

Ctrl Command	reset_error
Function	Resets the cumulative error code.
Call	<code>reset_error(Code)</code>
Parameter	Code OR connection of the error codes = sum by means of $2^{\text{Bitnumber}}$ of all bits to be reset as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> For error handling see chapter 6.8 "Error Handling", page 98. The cumulative error code can be reset bitwise (individually for each error type, e.g. bit #5 and bit #6 by <code>Code = 2^5 2^6</code> or by <code>Code = RTC5_BUSY RTC5_REJECTED</code>) or completely (by <code>Code = "-1"</code>, therefore by <code>Code = 2^32 - 1</code>). The meanings of bit numbers, error types and error constants is described in the command get_error. reset_error does not delete the error code of another user program currently assigned access rights to the board. The commands reset_error and n_reset_error are also available without explicit access rights to a specific RTC5 Board. The board-specific error variable <code>LastErrorHandler</code> (see chapter 6.8 "Error Handling", page 98) is neither generated nor altered by reset_error. In contrast, <code>AccErrorHandler</code> is altered as specified by <code>Code</code>.
RTC4→RTC5	New command.
References	get_error, get_last_error



Ctrl Command	restart_list
Function	Reenables “laser active” laser control signals and resumes execution of a list that was interrupted by the command pause_list or stop_list .
Call	<code>restart_list()</code>
Comments	<ul style="list-style-type: none"> The restart_list command is only executed if a list was previously halted by pause_list or stop_list and if the BUSY and PAUSED statuses (queryable with get_status) are set. Otherwise (e.g. if a list was halted with set_wait), the command is ignored (get_last_error return code RTC5_BUSY). restart_list resets the PAUSED status. The BUSY status is left unchanged.
RTC4→ RTC5	<p>Essentially unchanged functionality, however:</p> <p>The RTC5 also provides a PAUSED status that is set with pause_list and stop_list and reset with restart_list.</p>
References	pause_list , stop_list , release_wait

Ctrl Command	rs232_config
Function	Configures the RS232 interface for the specified baud rate.
Call	<code>rs232_config(BaudRate)</code>
Parameter	BaudRate Baud rate as an unsigned 32-bit value. Allowed value range: [300 ... +115200].
Comments	<ul style="list-style-type: none"> The default value is 9600 baud. The other RS232 interface parameters cannot be altered (data bits: 8, start bits: 1, stop bits: 1, parity: none). See also chapter 4.4.5 “RS232 Socket Connector”, page 54.
RTC4→ RTC5	New command.
References	rs232_write_data , rs232_read_data



Ctrl Command	rs232_read_data
Function	Reads a value from the input buffer of the RS232 interface (see page 238).
Call	<code>RS232Data = rs232_read_data()</code>
Result	<p>unsigned 32-bit value:</p> <p>Bit #0 Next (not yet read by the user program) value of the input buffer. (LSB)... Bit #7 Bit #8 "New" bit: = 1: The value is new (was not previously read). = 0: The value is old (was already read once by rs232_read_data). Bits #9... #15 Bits #16... #23 Bits #24... #31 Number of further (not yet read) characters. Number of buffer overruns.</p>
Comments	<ul style="list-style-type: none"> The RS232 interface is internally read every 10 µs (one character at a time). This character, if new, is stored in a 256-character ring buffer that can be transferred to the user program by rs232_read_data (asynchronously to reading). Byte #0 (bits #0...#7) returns only one character from the current reading position. Byte #1 (bit #8) indicates if the character has already been read by the user program. Byte #2 (bits #16...#23) indicates the number of characters in the input buffer that still have not been read by the user program. If no unread characters are present (byte #2 = 0), then the most recently read character is transferred with "new bit" = 0. Byte #3 (bits #24...#31) indicates the number of overruns of the input buffer (a corresponding number of characters were overwritten and therefore irretrievably lost). To reset the overrun counter, call rs232_read_data that many times. Example: return value 459098 = 0x0007015A = (0, 7, 1, 90) means: character 90 ('Z') was read and is new (1), 7 additional characters remain to be read, the buffer was never overrun (0).
RTC4→ RTC5	New command.
Version info	Last change with version RBF 513.
References	rs232_write_data



Ctrl Command	rs232_write_data
Function	Sends a data word (byte) to the RS232 interface.
Call	<code>rs232_write_data(Data)</code>
Parameter	<p>Data Data word as an unsigned 32-bit value. Only the least significant byte is transferred to the RS232 interface.</p>
Comments	<ul style="list-style-type: none"> The complete transmission of any previous data words is waited for. An overrun at the interface is not possible. See also chapter 9.1.6 "RS232 Interface", page 237.
RTC4→ RTC5	New command.
Version info	Last change with version DLL 516.
References	rs232_write_text , rs232_read_data

Ctrl Command	rs232_write_text
Function	Sends a text string (character-by-character) to the RS232 interface.
Call	<code>rs232_write_text(pData)</code>
Parameter	pData PC memory address of the first character (byte) of the to-be-sent text string as a pointer to a null-terminated string.
Comments	<ul style="list-style-type: none"> This command is split into an appropriate number of rs232_write_data commands. During execution of this command no further control commands can execute, but the board's list execution is not affected. If an executing list itself contains rs232_write_text_list commands, rs232_write_text should preferably not be used so as to avoid conflicts (race conditions). See also chapter 9.1.6 "RS232 Interface", page 237.
RTC4→ RTC5	New command.
Version info	Last change with version DLL 516, OUT 515.
References	rs232_write_data , rs232_write_text_list



Variable List Command	rs232_write_text_list
Function	Sends a text string (character-by-character) to the RS232 interface.
Call	<code>rs232_write_text_list(pData)</code>
Parameter	<code>pData</code> PC memory address of the first character (byte) of the to-be-sent text string as a pointer to a null-terminated ANSI string.
Comments	<ul style="list-style-type: none"> When a <code>rs232_write_text_list</code> command is loaded, the to-be-sent text (if more than 12 characters in length, \0 not included) is split into blocks of 12 characters, with each block receiving its own <code>rs232_write_text_list</code> command in the list memory (keep this in mind to prevent unintended overflow of the corresponding buffer area). Processing of the individual <code>rs232_write_text_list</code> commands is similar to that of the <code>rs232_write_text</code> command (the 12-character block is split into individual characters and sent sequentially to the RS232 interface). The command takes some time for execution, depending on the baud rate and text length. See also chapter 9.1.6 "RS232 Interface", page 237.
RTC4→ RTC5	New command.
Version info	Last change with version DLL 516, OUT 515.
References	rs232_write_text

Ctrl Command	rtc5_count_cards
Function	Returns the number of RTC5 Boards detected during initialization.
Call	<code>NoOfCards = rtc5_count_cards()</code>
Result	Number of RTC5 Boards as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> Initialization of the installed RTC5 Boards is to be made separately for each user program by calling <code>init_RTC5_dll</code>. The command <code>rtc5_count_cards</code> is also available without explicit access rights to a specific RTC5 Board. <code>rtc5_count_cards</code> is not available as a multi-board command. The board-specific error variables <code>LastError</code> and <code>AccError</code> (see chapter 6.8 "Error Handling", page 98) are neither generated nor altered by <code>rtc5_count_cards</code>.
RTC4→ RTC5	New command. Similar in functionality to its predecessor version <code>rtc4_count_cards</code> .



Delayed Short List Command	save_and_restart_timer
Function	Stores the current value of the RTC5 timer and resets it to zero.
Call	<code>save_and_restart_timer()</code>
Comments	<ul style="list-style-type: none"> The stored RTC5 timer value can be read by the get_time command. The command is useful for measuring the marking time of a marking process, see chapter 8.13 "Time Measurements", page 231. The RTC5 timer only counts list-command clock cycles. Counting is paused during interruptions by set_wait or pause_list. By get_lap_time the number of elapsed list-command clock cycles since the reset to zero can be queried. To compare RTC5-internal save_and_restart_timer time measurements to external time measurements by the BUSY pin, you should insert a list_nop between save_and_restart_timer and set_end_of_list. This ensures that any scanner delay completes before set_end_of_list. Without list_nop, save_and_restart_timer includes the scanner delay in its measurement even though it completes only after set_end_of_list (and therefore the BUSY pin is already low). As of DLL 543, OUT 543, RBF 524 a time stamp counter is available in addition to the RTC5 timer. It starts counting at 0 with load_program_file and counts uninterruptible all 10 µs clock periods. See also chapter 8.13 "Time Measurements", page 231.
RTC4→ RTC5	Unchanged.
References	get_lap_time , get_time



Ctrl Command	save_disk
Function	Stores all indexed characters, text strings and/or subroutines to a binary file on a PC storage medium, ordered by index, and returns the number of thereby stored list commands.
Call	NoOfSavedCommands = save_disk(Name, Mode)
Parameters	<p>Name File name as a pointer to a null-terminated ANSI string</p> <p>Mode This parameter (unsigned 32-bit value) specifies what should be stored. Bit #0 = 1: All indexed characters and text strings are stored. Bit #1 = 1: All indexed subroutines are stored. Bits #2...31: Are not evaluated.</p>
Result	The number of list commands saved by save_disk , as an unsigned 32-bit value
Comments	<ul style="list-style-type: none"> • The save_disk command can be used together with load_disk, for example, to perform defragmentation or to apply subsequent protection to subroutines (see page 84 and page 85). • The save_disk command stores complete sets (no individual characters, text strings or subroutines!) in the specified file. Indexed characters, text strings or subroutines that are referenced multiple times with copy_dst_src are correspondingly duplicated by save_disk, i.e. also stored multiple times. The save_disk command ignores unreferenced non-indexed (not subsequently referenced by set_char_pointer, ...) characters, text strings and subroutines. • save_disk stores to the binary file not only the list commands for characters, text strings and/or subroutines, but also their indexes and the number of commands. • The number of list commands stored with save_disk can differ considerably from the number of the list commands stored in the protected-area "List 3" ($= 2^{20} - \text{Mem1} - \text{Mem2} - \text{get_list_space}$), due to possible multiple referencing of an indexed character, text string or subroutine or referencing of an indexed character, text string or subroutine in the unprotected memory area ("List 1" or "List 2"). Prior to a subsequent load_disk, the returned number must be compared to the size of the protected memory area ($= 2^{20} - \text{Mem1} - \text{Mem2}$). • No-longer-needed characters (or text strings or subroutines) should be dereferenced by load_char (or load_text_table or load_sub) directly followed by list_return previously to save_disk (see page 85). • save_disk always stores all characters, text strings and/or subroutines from the referenced address to the next possible list_return. Jump commands (also branches to various list_return commands) are thereby neither evaluated nor executed. • The save_disk command automatically replaces unallowed commands (e.g. set_end_of_list) with list_nop commands; missing commands (e.g. list_return upon reaching the last memory position of "List 3") are added.



Ctrl Command	save_disk
Comments (cont'd)	<ul style="list-style-type: none"> If Mode = 0 or Name = 0, then save_disk is ignored (get_last_error return code: RTC5_PARAM_ERROR). The command is also ignored (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list) or the board's INTERNAL-BUSY status is currently set. In contrast, if a list has only been paused by set_wait (PAUSED status set), then save_disk can be executed. While the save_disk command is running, no further commands can be executed. During execution of this command, external starts are suppressed.
RTC4→ RTC5	New command.
References	load_disk

Undelayed Short List Command	select_char_set
Function	Selects one of the available character sets (exclusively) for the execution of subsequent mark_text and mark_text_abs commands.
Call	<code>select_char_set(No)</code>
Parameter	No Number of the desired character set as an unsigned 32-bit value. Allowed range: [0 ... 3].
Comments	<ul style="list-style-type: none"> The selection remains valid until another is encountered. The default value (after initialization) is No = 0. If No > 3, then the command is replaced with a list_nop. The current character set then remains valid (get_last_error return code RTC5_PARAM_ERROR). A character set change <i>within</i> a mark_text or mark_text_abs command is only possible if a select_char_set is located within a (called) indexed character. The selection encountered there then applies to all subsequent characters. If the selected character set is not (fully) defined, then missing characters are not marked (with mark_text or mark_text_abs).
RTC4→ RTC5	New command.
References	mark_text , mark_text_abs



Ctrl Command	select_cor_table
Function	Assigns the previously loaded correction tables to the scan head control ports and activates image field correction.
Call	<code>select_cor_table(HeadA, HeadB)</code>
Parameters	<p>HeadA = 0: Turns off the signals for scan head A (primary scan head connector) = 1...4: Assigns correction table number HeadA to scan head A. (as an unsigned 32-bit value)</p> <p>HeadB = 0: Turns off the signals for scan head B (secondary scan head connector). = 1...4: Assigns correction table number HeadB to scan head B (activation required). (as an unsigned 32-bit value)</p>
Comments	<ul style="list-style-type: none"> The command select_cor_table or select_cor_table_list should be called directly after loading the desired correction table(s) by load_correction_file and/or load_z_table (or after a subsequent load_program_file command – see below) in order to assign the correction table(s) to the corresponding control ports (as of version DLL 521, OUT 521, select_cor_table is automatically called by load_correction_file; see notes on page 138). select_cor_table and select_cor_table_list issue a jump with jump_speed (no <i>hard</i> jump) to the corrected galvanometer scanner position, so that image field correction is immediately applied to the currently set galvanometer scanner position. select_cor_table does this automatically and immediately, whereas select_cor_table_list inserts the jump in front of the next list command. Depending on the table contents and galvanometer scanner position, this can take a few clock cycles (and at least one clock cycle). Correction tables should be loaded and assigned prior to first-time starting of a list or issuance of a control command (e.g. goto_xy) that sets the scan system's galvanometer scanners in motion, and the command select_cor_table or select_cor_table_list should only be started <i>after</i> loading of the desired correction table(s). Otherwise, the galvanometers can, in some circumstances, be driven to an unexpected position and the laser beam deflected in an unintended direction. Correction tables, loaded (by load_correction_file) <i>prior</i> to load_program_file, are not fully effective before select_cor_table or select_cor_table_list is called (with versions older than DLL 521, OUT 521, the same might apply to correction tables loaded after load_program_file; see notes on page 138). load_program_file deletes the correction tables number 3 and 4. The select_cor_table command is ignored (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list). In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set). The list command select_cor_table_list can be used without restrictions. If the board's INTERNAL-BUSY status is currently set, select_cor_table is only executed with a delay (after INTERNAL-BUSY has been reset again). The INTERNAL-BUSY status is set while the jump to the corrected galvanometer scanner position is executed.

Ctrl Command	<code>select_cor_table</code>
Comments (cont'd)	<ul style="list-style-type: none"> If the values for parameters <code>HeadA</code> or <code>HeadB</code> are invalid (all values > 4) or 0, then <code>select_cor_table</code> turns off the scan head signals. The galvanometer scanners then remain in their last position. If the “second scan head control” option has not been enabled, <code>select_cor_table</code> does not assign a correction table to the secondary scan head connector. The default setting (after the command <code>load_program_file</code>) is (1, 0), i.e. correction table #1 is used for scan head A, whereas the output signals for scan head B are turned off (this corresponds to parameter values <code>HeadA</code> = 1 and <code>HeadB</code> = 0). Initially however, after <code>load_program_file</code> and until <code>select_cor_table</code> is called or the galvanometer scanners are explicitly moved, the galvanometer scanners stay in the position (0 0), even if the correction table content is different. If the “second scan head control” option is disabled, then signals of the second scan head remain permanently turned off; even so, multiple correction tables can still be loaded and used at any time by the primary scan head. Even with one scan head, you can thereby quickly switch back and forth between several correction tables, e.g. one for a pointer laser and one for the main laser with a different wavelength (use <code>select_cor_table(1, 0)</code> and <code>select_cor_table(2, 0)</code>, see also chapter 8.5 “Using Several Different Correction Tables”, page 192). In a double scan head system, table #1 is typically used for scan head A, and table #2 is used for scan head B: <code>select_cor_table(1, 2)</code>. But you can make a different assignment at any time. For 3D systems, the 3D option must be enabled. Provided the RTC5’s 3D option is enabled, you can load several (2D or 3D) correction tables. If the “second scan head control” option is disabled, then the XY axis must be connected to the primary scan head connector, the Z axis to the X or Y channel of the secondary scan head connector. If a 3D correction table is assigned to the primary scan head connector, corrected signals for an XY scan head are then transmitted by the primary scan head connector and corrected signals for the Z axis are transmitted by both channels of the secondary scan head connector. If multiple RTC5 Boards with enabled 3D option are installed in a PC, then that many 3-axis systems can be simultaneously controlled. Provided the RTC5’s 3D option and the “second scan head control” option are enabled, users specify which signals (XY or Z) are to be outputted by which connector when assigning the correction table (see also “Scan Head Connectors and Transfer Protocol”, page 42 and “2D and 3D Correction Files”, page 137). 2D correction tables should and 3D correction tables can be thereby assigned only to the XY axes and <i>not</i> to the Z axis (e.g. by <code>select_cor_table(0, 1)</code> or <code>select_cor_table(0, 2)</code> for XY at the scan head B connector and Z at the scan head A connector). Because unexpected system behavior might otherwise occur and the system would not know where the XY and Z axes are connected, the signals of both connectors are turned off if both connectors have been assigned a correction table and (at least) one of them is a 3D table (e.g. for <code>select_cor_table(1, 1)</code>). Parameters from currently loaded correction tables can be read by <code>get_table_para</code>, or from currently assigned correction tables by <code>get_head_para</code> (see also load_correction_file).



Ctrl Command	select_cor_table
RTC4→ RTC5	Unchanged functionality. Exception: now up to four correction tables can be stored in RTC5 memory, and – if the 3D option is enabled – even up to four 3D correction tables. However, two 3D correction tables can <i>not</i> be simultaneously assigned to the scan head control ports.
Version info	Last change (with version DLL 533, OUT 534): the parameter's value range has been increased to [0...4] (load_correction_file lets you load up to 4 correction files).
References	select_cor_table_list , load_correction_file , load_program_file

Variable List Command	select_cor_table_list								
Function	Similar to select_cor_table , but a list command.								
Call	<code>select_cor_table_list(HeadA, HeadB)</code>								
Parameters	<table> <tr> <td>HeadA</td> <td>= 0: turns off the signals for scan head A (primary scan head connector)</td> </tr> <tr> <td></td> <td>= 1...4: assigns correction table number HeadA to scan head A (as an unsigned 32-bit value)</td> </tr> <tr> <td>HeadB</td> <td>= 0: turns off the signals for scan head B (secondary scan head connector)</td> </tr> <tr> <td></td> <td>= 1...4: assigns correction table number HeadB to scan head B (activation required) (as an unsigned 32-bit value)</td> </tr> </table>	HeadA	= 0: turns off the signals for scan head A (primary scan head connector)		= 1...4: assigns correction table number HeadA to scan head A (as an unsigned 32-bit value)	HeadB	= 0: turns off the signals for scan head B (secondary scan head connector)		= 1...4: assigns correction table number HeadB to scan head B (activation required) (as an unsigned 32-bit value)
HeadA	= 0: turns off the signals for scan head A (primary scan head connector)								
	= 1...4: assigns correction table number HeadA to scan head A (as an unsigned 32-bit value)								
HeadB	= 0: turns off the signals for scan head B (secondary scan head connector)								
	= 1...4: assigns correction table number HeadB to scan head B (activation required) (as an unsigned 32-bit value)								
Comments	<ul style="list-style-type: none"> See select_cor_table. Even though select_cor_table_list is a short list command, execution of the directly following list command is delayed by a few clock cycles due to the intermediate jump to the corrected galvanometer position. The extent of this delay depends on the assigned correction table's content for the current galvanometer position (e.g. (0 0) after load_program_file); but it is at least 10 µs, even if the correction table does not specify a correction. 								
RTC4→ RTC5	New command.								
Version info	Last change (with version DLL 533, OUT 534): the parameter's value range has been increased to [0...4] (load_correction_file lets you load up to 4 correction files).								



Ctrl Command	select_rtc
Function	Defines, in a multi-board system, the active RTC5 Board for a user program. See chapter 6.6, page 92 .
Call	NoOfSelectedCard = select_rtc(CardNo)
Parameter	CardNo (DLL-internal) number of the RTC5 Board as an unsigned 32-bit value.
Result	<p>The returned (unsigned 32-bit) value is:</p> <ul style="list-style-type: none"> • CardNo if access rights exist for the specified board or if the specified board is not allocated to another user program (in this latter case, access rights are acquired through select_rtc – as by acquire_rtc, see below). • The number of the active board if CardNo exceeds the number of RTC5 Boards found during initialization or if CardNo = 0. • 0 otherwise (particularly when the specified board is reserved by another user program or if a version compatibility error is detected and activation of the board therefore cannot succeed) (get_last_error return code RTC5_ACCESS_DENIED and possibly RTC5_VERSION_MISMATCH).
Comments	<ul style="list-style-type: none"> • Activation of a board for a user program already occurs when the DLL for this user program is initialized (see init_rtc5_dll). The select_rtc command offers the possibility of changing the active board at any time. All the user program's commands subsequent to select_rtc (except for multi-board commands) are forwarded to the corresponding active board. • If the specified board is reserved for another user program, select_rtc has no effect (return value 0, get_last_error return code RTC5_ACCESS_DENIED). • If no access rights are assigned for the specified board, and it is not in use by another user program, then the board is not only activated but also – if it passed the version compatibility check (see below) – automatically reserved for this user program (as by acquire_rtc) and therefore unavailable to other applications (return value: CardNo). If a board is acquired (as by acquire_rtc), a version compatibility check is performed. If a version error is detected, then access to the board is denied and select_rtc has no effect (return code 0, get_last_error return code RTC5_ACCESS_DENIED RTC5_VERSION_MISMATCH). • If the specified board is already the active board for this user program, select_rtc has no effect (return value: CardNo). • The select_rtc command also has no effect if CardNo exceeds the number of RTC5 Boards found during initialization (see rtc5_count_cards) or if CardNo = 0 (real boards begin at 1). The return value is then the number of the active board (see above). Therefore, the command select_rtc(0) can be used to determine the number of the active board at any time. • The select_rtc command is also available without explicit access rights to a particular RTC5 Board. • select_rtc is not available as a multi-board command.
RTC4→ RTC5	With the RTC4, this command only activates the specified board (unconditionally). With the RTC5, this command additionally returns a value and either assigns access rights in some circumstances (as by acquire_rtc) for the specified board when none had existed or it has no effect.



Ctrl Command	select_serial_set
Function	Selects a serial-number-set for serial number control commands.
Call	<code>select_serial_set(No)</code>
Parameter	No Number of the desired serial-number-set as an unsigned 32-bit value. Allowed range: [0 ... 3]; only the two least significant bits are evaluated.
Comments	<ul style="list-style-type: none"> After initialization with load_program_file, serial-number-set 0 is selected. <code>select_serial_set</code> does not assign a serial-number-set for serial number marking. The list command select_serial_set_list is intended for that purpose. For command usage, see chapter 7.5.2 "Marking Serial Numbers", page 170.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 537, OUT 537.
References	select_serial_set_list , set_serial_step , get_serial

Undelayed Short List Command	select_serial_set_list
Function	Selects a serial-number-set for serial number list commands (as well as for serial number marking).
Call	<code>select_serial_set_list(No)</code>
Parameter	No Number of the desired serial-number-set as an unsigned 32-bit value. Allowed range: [0 ... 3]; only the two least significant bits are evaluated.
Comments	<ul style="list-style-type: none"> After initialization with load_program_file, serial-number-set 0 is selected. For command usage, see chapter 7.5.2 "Marking Serial Numbers", page 170.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 537, OUT 537.
References	select_serial_set , set_serial_step_list , get_list_serial , mark_serial , mark_serial_abs

Ctrl Command	send_user_data
Function	Sends data values by the user data bit to the specified scan system.
Comments	<ul style="list-style-type: none"> Up to now, this command has no effect, but it is reserved for future use according to chapter 7.3.6 "Status Monitoring and Diagnostics", page 143 and chapter 8.1.2 "Configuring Status Return Behavior", page 173.
References	read_user_data



Ctrl Command	set_angle
Function	Uses a specified rotation angle to define the rotation matrix M_R for all subsequent coordinate transformations (see chapter 8.2 "Coordinate Transformations", page 183).
Call	set_angle(HeadNo, Angle, at_once)
Parameters	<p>HeadNo Number of the scan head connector as an unsigned 32-bit value. = 1: The definition only affects the <i>primary</i> scan head connector. = 2: The definition only affects the <i>secondary</i> scan head connector (activation required) = 0, 3: The definition affects <i>both</i> scan head connectors. Only the two least significant bits are evaluated.</p> <p>Angle Rotation angle in ° (as a 64-bit IEEE floating point value); positive angles result in a counterclockwise rotation around the centerpoint of the image field.</p> <p>at_once This parameter (unsigned 32-bit value) determines when the defined transformation becomes effective. = 0: The new total transformation (total matrix and offset) is only calculated and applied to the current position when the next list command is executed. = 1: The new total transformation is calculated immediately (or before the next list command if a list is currently BUSY or the board is INTERNAL-BUSY) and applied to the current position. The "laser active" laser control signals are first switched off if necessary. = 2: The new total transformation (total matrix and offset) is only calculated and applied to the current position when the next jump_abs, jump_rel, goto_xy or goto_xyz command is executed. = 3: As with at_once = 1, but the laser control signals remain unaffected. > 3: As with at_once = 2.</p>
Comments	<ul style="list-style-type: none"> • See also chapter 8.2 "Coordinate Transformations", page 183.
RTC4 → RTC5	New command.
Version info	Last change with version DLL 525, OUT 527: behavior for at_once = 3.
References	set_angle_list , set_matrix , set_offset , set_scale



Variable List Command	set_angle_list
Function	Same as set_angle , but a list command.
Call	set_angle_list(HeadNo, Angle, at_once)
Parameters	<p>HeadNo See set_angle.</p> <p>Angle See set_angle.</p> <p>at_once This parameter (unsigned 32-bit value) determines when the defined transformation becomes effective.</p> <ul style="list-style-type: none"> = 0: The transformation settings are only collected and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (e.g. by a list command with at_once = 1 or a corresponding control command). = 1: The transformation is immediately calculated (including all transformation settings that were collected until then) and processed prior to the next list command. The “laser active” laser control signals are first switched off if necessary. = 2: The transformation settings are only collected and intermediately stored (as with at_once = 0). However, The transformation is immediately calculated (including all transformation settings that were collected and intermediately stored until then) and applied to the current position when the next jump_abs, jump_rel, goto_xy or goto_xyz command is executed. = 3: As with at_once = 1, but the laser control signals remain unaffected. > 3: As with at_once = 2.
RTC4→ RTC5	New command.
Version info	Last change with version DLL 525, OUT 527: behavior for at_once = 3.
References	set_angle



Ctrl Command	set_auto_laser_control	
Function	Initializes or deactivates position- and/or speed-dependent or encoder-speed-dependent laser control.	
Call	ErrorCode = set_auto_laser_control(Ctrl, Value, Mode, MinValue, MaxValue)	
Parameters	<p>Ctrl</p> <p>Control parameter (unsigned 32-bit value) for initializing or deactivating position- and/or speed-dependent or encoder-speed-dependent laser control.</p> <p>= 1 ... 6: Defines which signal parameter is automatically corrected by position- and/or speed-dependent or encoder-speed-dependent laser control.</p> <ul style="list-style-type: none"> = 1: 12-bit output value at the ANALOG OUT1 output port. = 2: 12-bit output value at the ANALOG OUT2 output port. = 3: Output value at the 8-bit digital output port. = 4: Pulse length (<i>PulseLength</i>) of the laser signals LASER1 and LASER2. = 5: Output period (<i>HalfPeriod</i>) of the laser signals LASER1 and LASER2. = 6: Output value at the 16-bit digital output. <p>= 0 or > 6: deactivates position-, speed- and encoder-speed-dependent laser control (for Ctrl > 6: get_last_error return code RTC5_PARAM_ERROR).</p>	
Value	<p>Defines the 100% value for the parameter selected by Ctrl (unsigned 32-bit value). Allowed values:</p> <p>For Ctrl = 1/2: 12-bit values [0 ... 4095], higher bits are ignored.</p> <p>For Ctrl = 3: 8-bit values [0 ... 255], higher bits are ignored.</p> <p>For Ctrl = 4: [0 ... (2³²-1)], 1 bit equals 1/64 µs.</p> <p>For Ctrl = 5: [0 ... (2³²-1)], 1 bit equals 1/64 µs.</p> <p>For Ctrl = 6: 16-bit values [0 ... (2¹⁶-1)], higher bits are ignored.</p>	
Mode	<p>Control parameter (unsigned 32-bit value) for initializing or deactivating speed-dependent or encoder-speed-dependent (but not position-dependent) laser control. Speed-dependent laser control (Mode = 1...5) lets you select which data is to be used for calculating speed-dependent correction.</p> <ul style="list-style-type: none"> =1: Set velocity (for all scan systems). =2: Queried actual velocity (only with iDRIVE scan systems: intelliSCAN, intelliSCAN_{de}, intelliDRILL, intellicube, intelliWELD, varioSCAN_{de}). =3: Reserved. =4: Reserved. =5: Encoder speed (counter pulse rate). =6: Combined speed of 2 and 5. <p>= 0 or > 6: Deactivates speed-dependent and encoder-speed-dependent laser control (for Mode > 6: get_last_error return code RTC5_PARAM_ERROR).</p>	



Ctrl Command	set_auto_laser_control	
	MinValue	Defines the <i>lower</i> range limit of the parameter selected by Ctrl for automatic correction and that cannot be undercut (unsigned 32-bit value). Allowed values: see Value.
	MaxValue	Defines the <i>upper</i> range limit of the parameter selected by Ctrl for automatic correction and that cannot be exceeded (unsigned 32-bit value). Allowed values: see Value.
Result	ErrorCode as an unsigned 32-bit value: 0 No error. 1 No primary scan head active. 2 No iDRIVE scan system (intelliSCAN, intelliSCAN _{de} , intelliDRILL, intellicube, intelliWELD, varioSCAN _{de}) active. 3 Invalid Ctrl value. 4 Invalid Mode value. 5 Access denied (board locked, get_last_error return code RTC5_ACCESS_DENIED).	
Comments	<ul style="list-style-type: none"> For information on using the command, see chapter 7.4.9 "Automatic Laser Control", page 159. MinValue and MaxValue are automatically exchanged if MinValue > MaxValue; if necessary, MinValue is clipped to ≤ Value and MaxValue clipped to ≥ Value. The control data for position- and/or speed-dependent laser control is exclusively derived from the scan system data at the primary scan head connector and then also applied for the secondary scan head connector (if that connector has been activated and a scan system is attached). At the time the command is issued, the primary scan head connector must already have been assigned a correction table, otherwise error code 1 is returned and Ctrl is set to 0. If only the secondary scan head connector is being used and/or the scan system at the primary scan head connector is shut off, then speed-dependent laser control does not function. For Mode = 2, a functioning iDRIVE scan system must be attached to the primary scan head connector. If this is not the case, then error code 2 is returned and Ctrlset to 0. If the iDRIVE scan system has reacted, then the queried data type for the primary scan head connector and both axes are set to "actual velocity" by control_command (Data = 0506_H). Only thereafter the command is sent to the RTC5, whereby the other parameters are applied. As a result, control_command is unavailable for the primary scan head connector as long as this mode selection is in effect. For Mode = 0 or 1, control_command is available without any restriction. Encoder-speed-dependent laser control (Mode = 5) functions even if no scan heads are attached to the scan-head connectors at runtime. The Processing-on-the-fly option does not need to be activated here either. 	



Ctrl Command	<code>set_auto_laser_control</code>
Comments (cont'd.)	<ul style="list-style-type: none"> Basically, Mode = 6 is the same as Mode = 2. However, encoder speeds are converted to galvanometer scanner units (bits/ms) by using the scaling factors from <code>set_fly_x</code> and <code>set_fly_y</code> or <code>set_fly_2d</code>. The result is then added to the current galvanometer scanner speeds. This requires an enabled Processing-on-the-fly option (in contrast to Mode = 5). As with Mode = 2 the current marking speed is used as reference speed. The <code>set_encoder_speed</code> command (which is used for Mode = 5) is not taken into account in Mode = 6. If an axis is not activated for Processing-on-the-fly at runtime, its encoder speed is not taken into account. If both axes are not activated for Processing-on-the-fly Mode = 6 functions similarly to Mode = 2. <code>set_fly_rot</code>, <code>set_fly_rot_pos</code>, <code>set_fly_x_pos</code> and <code>set_fly_y_pos</code> cannot be combined with the galvanometer scanner speed. If the values for Ctrl or Mode are invalid, then <code>set_auto_laser_control</code> is not executed (return value 3 or 4, <code>get_last_error</code> return code: RTC5_PARAM_ERROR). Automatic laser control should not be combined with the variable laser power of <code>set_multi_mcbsp_in</code>.
RTC4→RTC5	New command. In RTC4 compatibility mode for Ctrl = 1/2, the specified values for Value, MinValue and MaxValue are internally multiplied by 4. For Ctrl = 4/5 in RTC4 compatibility mode, the parameter values for Value, MinValue and MaxValue must be specified in units of 1/8 µs. They are then internally converted to 1/64 µs units (i.e. multiplied by 8). The allowed range of values is correspondingly smaller.
Version info	Changed with version DLL 540, OUT 540: Mode = 6.
References	<code>load_position_control</code> , <code>load_auto_laser_control</code> , <code>set_auto_laser_params</code> , <code>set_auto_laser_params_list</code> , <code>set_fly_x</code> , <code>set_fly_y</code> , <code>set_fly_2d</code> , <code>set_fly_rot</code> , <code>set_fly_rot_pos</code> , <code>set_fly_x_pos</code> , <code>set_fly_y_pos</code> , <code>set_multi_mcbsp_in</code>



Ctrl Command	set_auto_laser_params		
Function	Specifies the automatic laser control's to-be-controlled signal parameter, the 100% value and its corresponding limit values.		
Call	set_auto_laser_params(Ctrl, Value, MinValue, MaxValue)		
Parameters	Ctrl	Specifies the to-be-controlled signal parameter (see set_auto_laser_control). Allowed values: [1 ... 6].	
	Value	100% value (see set_auto_laser_control).	
	MinValue	Lower range limit (see set_auto_laser_control).	
	MaxValue	Upper range limit (see set_auto_laser_control).	
Comments	<ul style="list-style-type: none"> For information on using the command, see "Automatic Laser Control", page 159. The parameter's meaning is identical to that of the command set_auto_laser_control (see also comments there). However, set_auto_laser_params can neither start nor terminate automatic laser control. If the value for Ctrl is invalid (0 or >6), then set_auto_laser_params is not executed (get_last_error return code: RTC5_PARAM_ERROR). 		
RTC4→ RTC5	New command. RTC4 compatibility mode: see set_auto_laser_control .		
Version info	Available as of version DLL 517, OUT 516, RBF 512.		
References	set_auto_laser_control		

Delayed Short List Command	set_auto_laser_params_list		
Function	Same as set_auto_laser_params , but a list command.		
Call	set_auto_laser_params_list(Ctrl, Value, MinValue, MaxValue)		
Parameters	Ctrl	specifies the to-be-controlled signal parameter (see set_auto_laser_control). Allowed values: [1 ... 6].	
	Value	100% value (see set_auto_laser_control).	
	MinValue	lower range limit (see set_auto_laser_control).	
	MaxValue	upper range limit (see set_auto_laser_control).	
Comments	<ul style="list-style-type: none"> If the value for Ctrl is invalid (0 or >6), then set_auto_laser_params_list is replaced with a list_nop (get_last_error return code RTC5_PARAM_ERROR). 		
RTC4→ RTC5	New command. RTC4 compatibility mode: see set_auto_laser_control .		
Version info	Available as of version DLL 517, OUT 516, RBF 512.		
References	set_auto_laser_params , set_auto_laser_control		



Ctrl Command	set_char_pointer
Function	Stores the absolute start address of a command list in the internal management table for indexed characters.
Call	<code>set_char_pointer(Char, Pos)</code>
Parameters	<p>Char Index of the indexed character whose starting address <code>Pos</code> should be entered in the management table as an unsigned 32-bit value. Allowed range: [0 ... 1023]. The same applies as for the command load_char: <code>Char</code> = character set number * 256 + ASCII number of the character (character sets are numbered 0 to 3).</p> <p>Pos Absolute start address as an unsigned 32-bit value. Allowed range: [0 ... $(2^{20}-1)$].</p>
Comments	<ul style="list-style-type: none"> If <code>Char > 1023</code> and/or <code>Pos > (2²⁰-1)</code>, then the command is <i>not</i> executed (get_last_error return code <code>RTC5_PARAM_ERROR</code>). The set_char_pointer command can be used for referencing an indexed character. It thereby becomes an indexed character that is protectable by save_disk/load_disk and/or callable by the index. set_char_pointer can also be used to reference anew an indexed subroutine, character or text string so that it can also be called by a second index. Here, it is preferable to use the copy_dst_src command for index management. The start addresses of command lists that are to be referenced with set_char_pointer can be queried by get_input_pointer before loading the command lists. set_char_pointer only stores <i>starting addresses</i> in the internal management table. An indexed character only gains protection by a subsequent save_disk/load_disk command. <code>Pos</code> should not be an arbitrary address within a list. Instead, it should be the starting address of an actually existing subroutine that was finalized by list_return and does not contain set_end_of_list.
RTC4→ RTC5	New command.
References	load_char



Ctrl Command	set_char_table	
Function	Stores the absolute start address of a command list in the internal management table for indexed text strings.	
Call	set_char_table(Index, Pos)	
Parameters	<p>Index Index of the indexed text string whose starting address Pos should be entered in the management table as an unsigned 32-bit value. Allowed range: [0 ... 41]). The same ordering applies as for the load_text_table command: = 0...9: Digits for marking the time and date [0 ... 9]. = 10...21: Months [January ... December]. = 22...28: Days-of-the-week [Sunday ... Saturday]. = 29: Blank character for marking serial numbers. = 30...39: Digits for marking serial numbers [0 ... 9]. = 40: Text for "a.m." = 41: Text for "p.m."</p> <p>Pos Absolute start address as an unsigned 32-bit value. Allowed range: [0 ... (2^{20}-1)].</p>	
Comments	<ul style="list-style-type: none"> The command is synonymous with set_text_table_pointer. 	RTC4→RTC5 Unchanged functionality (except for the extended range of values). The command was previously only available for the RTC SCANalone Board, i.e. the standalone version of the RTC4 Board.
References		
	set_text_table_pointer	



Ctrl Command	set_control_mode																									
Function	Enables or disables the external control input for external list starts and resets the counter for external list starts to zero.																									
Call	set_control_mode(Mode)																									
Parameter	<p>Mode (as an unsigned 32-bit value):</p> <table> <thead> <tr> <th>Bit #</th> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>Bit #0 (LSB)</td> <td>= 1: = 0:</td> <td>The external start input (by /START, /START2 or /Slave-START) is enabled. The external start input is disabled.</td> </tr> <tr> <td>Bit #1</td> <td>= 1: = 0:</td> <td>An external list stop (/STOP, /STOP2, /Slave-STOP or simulate_ext_stop) causes explicit cancellation of the external start queue's entries (/START, /START2, /Slave-START or simulate_ext_start). No effect.</td> </tr> <tr> <td>Bit #2</td> <td>= 1: = 0:</td> <td>The track delay (defined by simulate_ext_start, set_ext_start_delay or set_ext_start_delay_list) that postpones execution of the list start relative to the triggering input signal or simulate_ext_start or simulate_ext_start_ctrl command (see "External List Start", page 240) is deactivated. No effect. To define and activate the track delay (e.g. for Processing-on-the-fly applications), use the command simulate_ext_start, set_ext_start_delay or set_ext_start_delay_list.</td> </tr> <tr> <td>Bit #3</td> <td>= 1: = 0:</td> <td>The external start input is <i>not</i> disabled by an external stop request. The external start input is disabled by an external stop request.</td> </tr> <tr> <td>Bit #4</td> <td></td> <td>Disables simulate_ext_start_ctrl.</td> </tr> <tr> <td>Bits #5...8</td> <td></td> <td>Reserved.</td> </tr> <tr> <td>Bit #9</td> <td>= 1: = 0:</td> <td>Encoder resets of the two internal encoder counters (initiated by the Processing-on-the-fly commands set_fly_x, set_fly_y or set_fly_rot) occur after the subsequent start trigger (i.e. the subsequent external start signal or simulate_ext_start or simulate_ext_start_ctrl command, possibly postponed by a track delay defined by simulate_ext_start, set_ext_start_delay or set_ext_start_delay_list, see also bit #2). Encoder resets occur immediately with each initiating Processing-on-the-fly command.</td> </tr> </tbody> </table>	Bit #	Value	Description	Bit #0 (LSB)	= 1: = 0:	The external start input (by /START, /START2 or /Slave-START) is enabled. The external start input is disabled.	Bit #1	= 1: = 0:	An external list stop (/STOP, /STOP2, /Slave-STOP or simulate_ext_stop) causes explicit cancellation of the external start queue's entries (/START, /START2, /Slave-START or simulate_ext_start). No effect.	Bit #2	= 1: = 0:	The track delay (defined by simulate_ext_start , set_ext_start_delay or set_ext_start_delay_list) that postpones execution of the list start relative to the triggering input signal or simulate_ext_start or simulate_ext_start_ctrl command (see "External List Start", page 240) is deactivated. No effect. To define and activate the track delay (e.g. for Processing-on-the-fly applications), use the command simulate_ext_start , set_ext_start_delay or set_ext_start_delay_list .	Bit #3	= 1: = 0:	The external start input is <i>not</i> disabled by an external stop request. The external start input is disabled by an external stop request.	Bit #4		Disables simulate_ext_start_ctrl .	Bits #5...8		Reserved.	Bit #9	= 1: = 0:	Encoder resets of the two internal encoder counters (initiated by the Processing-on-the-fly commands set_fly_x , set_fly_y or set_fly_rot) occur after the subsequent start trigger (i.e. the subsequent external start signal or simulate_ext_start or simulate_ext_start_ctrl command, possibly postponed by a track delay defined by simulate_ext_start , set_ext_start_delay or set_ext_start_delay_list , see also bit #2). Encoder resets occur immediately with each initiating Processing-on-the-fly command.	
Bit #	Value	Description																								
Bit #0 (LSB)	= 1: = 0:	The external start input (by /START, /START2 or /Slave-START) is enabled. The external start input is disabled.																								
Bit #1	= 1: = 0:	An external list stop (/STOP, /STOP2, /Slave-STOP or simulate_ext_stop) causes explicit cancellation of the external start queue's entries (/START, /START2, /Slave-START or simulate_ext_start). No effect.																								
Bit #2	= 1: = 0:	The track delay (defined by simulate_ext_start , set_ext_start_delay or set_ext_start_delay_list) that postpones execution of the list start relative to the triggering input signal or simulate_ext_start or simulate_ext_start_ctrl command (see "External List Start", page 240) is deactivated. No effect. To define and activate the track delay (e.g. for Processing-on-the-fly applications), use the command simulate_ext_start , set_ext_start_delay or set_ext_start_delay_list .																								
Bit #3	= 1: = 0:	The external start input is <i>not</i> disabled by an external stop request. The external start input is disabled by an external stop request.																								
Bit #4		Disables simulate_ext_start_ctrl .																								
Bits #5...8		Reserved.																								
Bit #9	= 1: = 0:	Encoder resets of the two internal encoder counters (initiated by the Processing-on-the-fly commands set_fly_x , set_fly_y or set_fly_rot) occur after the subsequent start trigger (i.e. the subsequent external start signal or simulate_ext_start or simulate_ext_start_ctrl command, possibly postponed by a track delay defined by simulate_ext_start , set_ext_start_delay or set_ext_start_delay_list , see also bit #2). Encoder resets occur immediately with each initiating Processing-on-the-fly command.																								



Ctrl Command	set_control_mode	
Parameter (cont'd)	Bits #10 = 1: Bits #12-31	<p>Track delay configured by <code>simulate_ext_start</code>, <code>set_ext_start_delay</code> or <code>set_ext_start_delay_list</code> is counted beginning with the most recent externally (but not with <code>execute_list_pos</code> etc.) triggered or simulated external list start. The interval between subsequent external list starts (in encoder pulses) is thus constant (see also page 242). For <code>stop_execution</code> or an external stop signal, bit #10 gets reset to "0". This bit has no effect if the firmware version is 506 or lower (see <code>get_RTC_version</code>).</p> <p>= 0:</p> <p>Track delay configured by <code>simulate_ext_start</code>, <code>set_ext_start_delay</code> or <code>set_ext_start_delay_list</code> is counted beginning with the time point an external list start was requested (i.e. with the corresponding <code>simulate_ext_start</code> or <code>simulate_ext_start_ctrl</code> command or external start signal). The interval between subsequent external list starts (in encoder pulses) can thus vary. This is standard for firmware version 506 or lower (see <code>get_RTC_version</code>).</p> <p>Reserved.</p>
Comments	<ul style="list-style-type: none"> If execution is aborted by the command <code>stop_execution</code>, then bit #0 and bit #10 gets reset to zero, thus deactivating external start inputs and the counting of track delays with respect to a possible triggering event. If bit #9 = 0, then there is generally a small (random) time offset (10 µs jitter) between the start signal at /START, /START2 and the actual list start. If bit #9 = 1, then this 10 µs jitter is not present because the encoder reset then occurs synchronously with the start signal. See also "External List Stop", page 239 and "External List Start", page 240. 	
RTC4→ RTC5	<p>In contrast to the RTC4, bit#8 cannot not be used to lock or unlock the upper 8 bits of the 16-bit digital output port for I/O commands. The RTC5 automatically reserves (locks) these bits for varioSCAN FLEX control by <code>move_to</code>. It is not possible to explicitly release these bits (release automatically occurs by <code>load_program_file</code>).</p> <p>Otherwise: unchanged functionality for the bits that are also used on the RTC4.</p>	
Version info	Last change (DLL 543, OUT 543): bit #4.	
References	set_extstartpos , get_counts , set_max_counts , get_startstop_info , move_to	



Normal List Command	set_control_mode_list
Function	Similar to set_control_mode , but a list command.
Call	<code>set_control_mode_list(Mode)</code>
Parameter	Mode (as an unsigned 32-bit value).
Comments	<ul style="list-style-type: none"> The counter for external list starts is <i>not</i> reset by this command.
RTC4→ RTC5	Unchanged functionality for the bits that are also used on the RTC4.
Version info	Last change (DLL 543, OUT 543): bit #4.
References	set_control_mode

Ctrl Command	set_default_pixel
Function	Defines the pulse length default value for the default pixel that terminates pixel output mode.
Call	<code>set_default_pixel(PulseLength)</code>
Parameters	<p>PulseLength default pixel pulse length (as an unsigned 32-bit value). <i>1 bit equals 1/64 μs.</i> Allowed range: [0 ... (2³²-1)]</p>
Comments	<ul style="list-style-type: none"> In pixel output mode, the pixel pulse length at the end of an image line (at the beginning of the default pixel) gets set to the specified default value (see page 219). When the RTC5 is initialized (by load_program_file), the pulse length default value is set to 0.
RTC4→ RTC5	New command.
References	set_port_default

Undelayed Short List Command	set_default_pixel_list
Function	Defines the pulse length default value for the default pixel that terminates pixel output mode.
Call	<code>set_default_pixel_list(PulseLength)</code>
Parameters	<p>PulseLength default pixel pulse length (as an unsigned 32-bit value). <i>1 bit equals 1/64 μs.</i> Allowed range: [0 ... (2³²-1)].</p>
Comments	<ul style="list-style-type: none"> Same as set_default_pixel, but a list command
RTC4→ RTC5	New command.
References	set_default_pixel , set_port_default



Ctrl Command	set_defocus
Function	Determines a focus shift for all subsequent 3D vector outputs.
Restriction	If the 3D option has not been enabled or if no 3D correction table has been assigned (see select_cor_table), then the command has no effect. Nevertheless, the supplied focus shift value is stored internally and takes effect as soon as a 3D correction table is assigned.
Call	<code>set_defocus(Shift)</code>
Parameter	<p>Shift Focus shift in <i>bits</i> (as a signed 32-bit value). Allowed range: [-32768 ... 32767]. Out-of-range values are edge-clipped.</p>
Comments	<ul style="list-style-type: none"> A focus shift causes the Z axis (varioSCAN) to defocus the laser spot on the target surface. The focus shift is a <i>signed</i> parameter. A positive value <i>increases</i> the focal length of the Z axis (varioSCAN) and shifts, for example, the focus position with the control value (0 0 0) by approx. $d = \text{Shift} / K_z$ to the plane $z = -d$ (the focus is shifted in the opposite direction of the Z coordinate axis and in the opposite direction of a working-plane offset specified by set_offset_xyz). Here, K_z is the calibration factor for the Z direction (see also "3D Marking Commands", page 194). If the resulting total Z output value is too large, the Z axis moves to the limit stop. Users should take care to avoid this situation. If the command is called during output of a vector, then it is only executed directly before the next list command. To avoid hard jumps, a jump to the changed Z coordinate is performed at jump speed. This might take a few clock cycles, depending on the jump speed setting and desired focus shift. If no list is currently BUSY, then the jump executes immediately, whereby no downtime occurs at the next list start. If the board's INTERNAL-BUSY status is currently set, set_defocus is only executed with a delay (after INTERNAL-BUSY has been reset again). The INTERNAL-BUSY status is set while the jump to the changed Z coordinate is executed. After vector-defined laser control is activated by set_vector_control(ctrl = 7), the focus shift might change with parameterized mark or jump commands, too.
RTC4 → RTC5	Essentially unchanged functionality, except for avoidance of hard jumps.
Version info	Last change with version OUT 515.
References	set_defocus_list , set_offset_xyz



Variable List Command	set_defocus_list
Function	Similar to set_defocus , but a list command.
Restriction	See set_defocus .
Call	set_defocus_list(Shift)
Parameter	Shift See set_defocus .
Comments	<ul style="list-style-type: none">• See set_defocus.• Even though set_defocus_list is a short list command, execution of the directly following list command is delayed by a few clock cycles due to the intermediate jump to the changed Z-position. The extent of this delay depends on the size of the specified focus shift; but it is at least 10 µs, even if Shift = 0.
RTC4→ RTC5	See set_defocus .

Ctrl Command	set_delay_mode		
Function	Turns the variable polygon delay mode and the variable jump delay mode on or off and sets some special scanner-delay related parameters as well as the 3D Z-Move mode.		
Call	set_delay_mode(VarPoly, DirectMove3D, EdgeLevel, MinJumpDelay, JumpLengthLimit)		
Parameters	All parameters must be unsigned 32-bit values.		
	Parameter	Allowed Values	Description
	VarPoly	> 0 = 0	Enables the variable polygon delay mode. See page 118 . Disables the variable polygon delay mode (default setting).
	DirectMove3D		This parameter effects only 3D-applications.
	D	> 0 = 0	The Z value is changed directly (linearly) to its end value during a jump. The Z value is changed to its end-value in such a way that the focus is kept in one plane during the entire jump.
	EdgeLevel	0 ... (2 ³² -1). 1 bit equals 10 µs.	This parameter defines a maximum "laser on" time for the corners of a polyline. If the polygon delay is longer than or equal to this value (because the angle ϕ is close to 180°, for instance), the laser is turned off (after a LaserOff delay) and a new polyline is started. This can be useful for preventing burn-in effects. The edgelevel must be smaller than twice the set value for the polygon delay, otherwise it has no effect. See also figure 36 . Note: To disable this feature, set the edgelevel to (2 ³² -1) (default value).
	MinJumpDelay	0 ... (2 ³² -1). 1 bit equals 10 µs.	Minimum jump delay that cannot be undercut for jumps shorter than JumpLengthLimit (even for jump vectors of zero length, see figure 33). For jumps longer than JumpLengthLimit, the MinJumpDelay has no relevance. To avoid anomalies in the range of MinJumpDelay, define a value for MinJumpDelay that is not larger than JumpDelay.
	JumpLengthLimit	0 ... (2 ³² -1)	Jump length limit in bits. If the jump vector is longer than this value, then the fixed jump delay (see set_scanner_delays) is inserted. For all shorter jump lengths, a linearly interpolated Variable Jump Delay between MinJumpDelay and the jump delay is calculated and inserted (see figure 33). To disable the Variable Jump Delay mode, set the JumpLengthLimit to 0.



Ctrl Command	set_delay_mode
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, the RTC5 multiplies the specified value for <code>JumpLengthLimit</code> by 16 (the allowed range of values is correspondingly reduced to [0 ... (2 ²⁸ -1)]).
References	set_scanner_delays , load_varpolydelay , set_delay_mode_list

List Multi-Command	set_delay_mode_list
Function	Identical with set_delay_mode , but a list command.
Call	<code>set_set_delay_mode_list(VarPoly, DirectMove3D, EdgeLevel, MinJumpDelay, JumpLengthLimit)</code>
Parameters	See set_delay_mode .
Comments	<ul style="list-style-type: none"> • See set_delay_mode. • The command requires two list buffer positions and executes as two undelayed short list commands. • Any still-pending delayed short list command executes first.
RTC4→ RTC5	New command. RTC4 compatibility mode: see set_delay_mode .
Version info	Available as of version DLL 536, OUT 536.
References	set_delay_mode

Ctrl Command	set_dsp_mode
Function	Sets a DSP mode for short-command count.
Call	<code>initial_dsp_mode = set_dsp_mode(Mode)</code>
Parameter	Mode Desired DSP mode as an unsigned 32-bit value
Result	DSP mode for which the board was set during initialization (as an unsigned 32-bit value).
Comments	<ul style="list-style-type: none"> • This command is only needed when multiple RTC5 Boards with differing DSP versions are to be operated synchronously in a master/slave chain (see page 94), and then only if matching of short list counts is not to be performed by sync_slaves. • Even if another mode was already set by set_dsp_mode, the set_dsp_mode command always returns the DSP mode that the board was set to during initialization. This return value (<code>initial_dsp_mode</code>) is identical to Byte #2 of get_RTC_version (DSP version number). • No mode can be set that is larger than <code>initial_dsp_mode</code> (if necessary, <code>Mode</code> is clipped to <code>initial_dsp_mode</code>). Thus, faster boards can only be matched to slower boards, not the other way around.
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> • Available as of version DLL 523, OUT 524.
References	sync_slaves , get_RTC_version



Undelayed Short List Command	set_ellipse
Function	Defines the shape of an elliptical arc that can subsequently be marked by mark_ellipse_abs or mark_ellipse_rel .
Call	<code>set_ellipse(a, b, Phi0, Phi)</code>
Parameters	<p>a, b Lengths of the elliptical half-axes in <i>bits</i> as unsigned 32-bit values. Allowed range: [1 ... 8388607]. Out-of-range positive values are edge-clipped.</p> <p>Phi0 Beginning phase angle in ° (the arc starting point's position relative to the end point of half-axis a) as a 64-bit IEEE floating point value. Positive angle values correspond to clockwise angles. <code>Phi0</code> gets normalized to the value range [0...<360°].</p> <p>Phi Arc angle in ° (to-be-marked ellipse section) as a 64-bit IEEE floating point value. Positive angle values correspond to clockwise angles. Allowed range: [-3600.0° ... +3600.0°] (± 10 full ellipses); An out-of-range value is edge-clipped.</p>
Comments	<ul style="list-style-type: none"> Specify the to-be-marked elliptical arc's position and orientation in the two-dimensional image field by mark_ellipse_abs or mark_ellipse_rel. For descriptions of the individual parameters, see also page 105. For <code>a < 1</code> or/and <code>b < 1</code>, set_ellipse is replaced with a list_nop (get_last_error return code: <code>RTC5_PARAM_ERROR</code>).
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the length values specified for the elliptical half-axes by 16 (the allowed range of values is correspondingly reduced to [1 ... 524287]).
Version info	<ul style="list-style-type: none"> Available as of version DLL 518, OUT 517, RBF 515. For older RTC5 Boards with DSP version numbers < 2 (get_RTC_version bits #16-23), the command is neither executed nor replaced by list_nop (get_last_error return code: <code>RTC5_TYPE_REJECTED</code>).
References	mark_ellipse_abs , mark_ellipse_rel



Delayed Short List Command	set_encoder_speed
Function	Defines the target encoder speed and further parameters for encoder-speed-dependent automatic laser control.
Call	<code>set_encoder_speed(EncoderNo, Speed, Smooth)</code>
Parameters	<p>EncoderNo Number of the encoder counter to be used for speed measurement as an unsigned 32-bit value. Allowed values: = 0: Encoder counter Encoder0. = 1: Encoder counter Encoder1. = 2 and 3: vectorial encoder velocity: a vectorial velocity is calculated from both encoder speeds (by the pulse rates of both encoder counters) for use with encoder-speed-dependent automatic laser control in Mode = 5. Higher bits are ignored.</p> <p>Speed Target encoder speed (counter pulse rate) in [counter pulses/ms] as a 64-bit IEEE floating point value. Allowed range: [100.0 ... 16000.0]; for Speed > 16000.0, Speed is clipped to 16000.0, for 0.0 < Speed < 100.0 to 100.0.</p> <p>Smooth Smoothing factor for a 2-stage low-pass filter as a 64-bit IEEE floating point value. Allowed range: [0.0 ... 1.0]. Larger values are clipped.</p>
Comments	<ul style="list-style-type: none"> If Smooth = 0.0, then only the current encoder speed CurrentSpeed (based on counter pulses received in the most recent 10 µs) is used; if Smooth = 1.0, then the speed from the previous clock cycle PreviousSpeed. In general, the used speed is: $\text{Speed} = \text{PreviousSpeed} \times \text{Smooth} + \text{CurrentSpeed} \times (1.0 - \text{Smooth})$. If Speed ≤ 0.0 or Smooth < 0.0, then set_encoder_speed is, already during loading, replaced by a list_nop (get_last_error) return code RTC5_PARAM_ERROR). One encoder increment results in four counter pulses. The maximum value for Speed (16000.0) corresponds to a counting rate of 16 MHz. The minimum value for Speed (100.0) corresponds to a counting rate of 1/(10 µs), i.e. one counter pulse per output period. Beware of the low resolution of encoder-speed-dependent laser control for low speed values! Encoder-speed-dependent correction is only recommended if substantially more than one counter pulse per output period (10 µs) are received. See also "Encoder-Speed-Dependent Laser Control", page 168.
RTC4→RTC5	New command.
Version info	Last change with version DLL 536, OUT 536: EncoderNo = 2 and 3.
References	set_encoder_speed_ctrl, set_auto_laser_control



Ctrl Command	set_encoder_speed_ctrl
Function	Same as set_encoder_speed , but a control command.
Call	<code>set_encoder_speed_ctrl(EncoderNo, Speed, Smooth)</code>
Parameters	<p>EncoderNo Number of the encoder counter to be used for speed measurement as an unsigned 32-bit value. Allowed values: = 0: Encoder counter Encoder0. = 1: Encoder counter Encoder1. = 2 and 3: vectorial encoder velocity: a vectorial velocity is calculated from both encoder speeds (by the pulse rates of both encoder counters) for use with encoder-speed-dependent automatic laser control in Mode= 5. Higher bits are ignored.</p> <p>Speed Target encoder speed (counter pulse rate) in [counter pulses/ms] as a 64-bit IEEE floating point value. Allowed range: [100.0 ... 16000.0]; for Speed > 16000.0, Speed is clipped to 16000.0, for 0.0 < Speed < 100.0 to 100.0.</p> <p>Smooth Smoothing factor for a 2-stage low-pass filter as a 64-bit IEEE floating point value. Allowed range: [0.0 ... 1.0]. Larger values are clipped.</p>
Comments	<ul style="list-style-type: none"> For Speed ≤ 0.0 or Smooth < 0.0, the command is ignored (get_last_error return code RTC5_PARAM_ERROR). The command is also ignored (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list). In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set) or the board's INTERNAL-BUSY status is currently set.
RTC4→RTC5	New command.
Version info	Last change with version DLL 536, OUT 536: EncoderNo = 2 and 3.
References	set_encoder_speed , set_auto_laser_control



Normal List Command	set_end_of_list
Function	Ends execution of a list.
Call	set_end_of_list()
Comments	<ul style="list-style-type: none"> If, during processing of a list, the set_end_of_list command is encountered and no automatic list change was previously activated (see "Automatic List Changing", page 79), then list execution ends. The "laser active" laser control signals are then switched off and a home jump, if defined by home_position or home_position_xyz, is executed (the INTERNAL-BUSY status is set while the home jump is executed). In contrast, upon reaching a set_end_of_list command, execution continues at the other list if an automatic list change was previously activated. The other list can also be "List 1" if "List 2" was not configured (<code>Mem2 = 0</code>, see config_list). Upon processing of the set_end_of_list command, the USED status of the respective list (USED1 or USED2) is always set and the list's BUSY status (BUSY1 or BUSY2) reset (see also "List Status", page 76). The BUSY list execution status, on the other hand, is only reset if no automatic list change was previously activated. An automatic list change of the <i>input</i> pointer never occurs during <i>loading</i> of the set_end_of_list command (in contrast to an automatic list change of the <i>output</i> pointer during <i>execution</i> of the set_end_of_list command, if previously activated by auto_change_pos or start_loop). Upon loading the set_end_of_list command, the list's READY status (READY1 or READY2) is set and the list's LOAD status (LOAD1 or LOAD2) is reset. Additionally, flushing of the list input buffer is triggered (see page 75). The set_end_of_list command is ignored during loading and execution, i.e. replaced with a list_nop if an indexed subroutine is currently being loaded or executed (get_last_error return code <code>RTC5_IGNORED</code>).
RTC4→ RTC5	<p>Essentially unchanged functionality, however:</p> <p>The RTC5 also provides a USED status that is reset upon loading the set_end_of_list command.</p>
Version info	Last change with version OUT 517.
References	set_start_list



Ctrl Command	set_extstartpos
Function	Defines the start address (within the range of "List 1" or "List 2") where execution should continue upon subsequent external list starts.
Call	<code>set_extstartpos(Pos)</code>
Parameter	Pos Absolute address of the first list command to be executed as an unsigned 32-bit value. Allowed range: [0 ... (2^{20} –1)].
Comments	<ul style="list-style-type: none"> By default, an external list start results in a continuation or start at the beginning of "List 1" (<code>Pos = 0</code>). <code>Pos</code> must be within the range of "List 1" or "List 2". Otherwise, <code>Pos = 0</code> is set. The specified start address is used for all subsequent external list starts until a new address is specified by <code>set_extstartpos</code> or <code>set_extstartpos_list</code>. An address range validity check is performed on <code>Pos</code> before each external list start; <code>Pos</code> might therefore become set to 0 at a later point (and remain at this value) if the configuration was correspondingly changed in the meantime (see config_list). See also "External List Start", page 240.
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). The command was previously only available for the RTC SCANalone Board, i.e. the standalone version of the RTC4 Board.
References	set_extstartpos_list , set_control_mode

Undelayed Short List Command	set_extstartpos_list
Function	Same as <code>set_extstartpos</code> , but a list command.
Call	<code>set_extstartpos_list(Pos)</code>
Parameter	Pos Absolute address of the first list command to be executed as an unsigned 32-bit value. Allowed range: [0 ... (2^{20} –1)].
Comments	<ul style="list-style-type: none"> This list command can be used within a list, to "link" it to the list that follows. See <code>set_extstartpos</code>.
RTC4→ RTC5	Unchanged functionality (except for the extended range of values).
References	set_extstartpos



Ctrl Command	set_ext_start_delay
Function	Sets a track delay for subsequent external list starts, so that the lists are started with a delay relative to the triggering input signal or simulate_ext_start or simulate_ext_start_ctrl command.
Call	<code>set_ext_start_delay(Delay, EncoderNo)</code>
Parameters	<p>Delay Track delay (counter steps of the selected encoder counter <code>EncoderNo</code>) as a signed 32-bit value. Allowed range: $[-2^{31} \dots + (2^{31}-1)]$</p> <p>EncoderNo Number of the to-be-used encoder counter as an unsigned 32-bit value. Allowed values: = 0: Encoder counter Encoder0. = 1: Encoder counter Encoder1.</p>
Comments	<ul style="list-style-type: none"> For external list starts, see page 240. The track delay is specified in (relative) counting units of the selected encoder counter (the RTC5's encoder counter is triggered by an external or simulated encoder signal, see page 246). Ensure that the sign of the track delay (<code>Delay</code> parameter) is appropriate for the selected encoder's counting direction (for external triggering, this corresponds to the work-piece's direction of motion). For <code>Delay = 0</code>, the track delay is deactivated. If a track delay is specified, that causes a subsequent start trigger (initiated by simulate_ext_start or an external start signal) to occur when the BUSY status or INTERNAL-BUSY status is set (e.g. when outputting a list or during goto_xy), then no list starts are get triggered by this start trigger (in this case, bit#11 of the get_startstop_info return value is set). Track delays can also be set with simulate_ext_start. Track delays are deactivated by initialization (with load_program_file), by external list stops and by stop_execution. They can also be deactivated with set_control_mode (bit#2). The command set_ext_start_delay cancels already externally triggered list starts that have not yet executed and are still being held in a queue that accommodates up to 8 starts. If <code>EncoderNo > 1</code>, then set_ext_start_delay is ignored (get_last_error return code <code>RTC5_PARAM_ERROR</code>).
RTC4→ RTC5	<p>Unchanged functionality (except for the extended range of values). The RTC5 allows this command to be used not only (as with the RTC4) together with a real encoder (hardware encoder), but also with an encoder simulation started by simulate_encoder. In RTC4 compatibility mode, the RTC5 multiplies the specified value for <code>Delay</code> by 16 (the allowed range of values is correspondingly reduced).</p>
References	simulate_ext_start , set_ext_start_delay_list set_extstartpos , set_extstartpos_list , set_control_mode



Normal List Command	set_ext_start_delay_list
Function	Same as set_ext_start_delay , but a list command.
Call	<code>set_ext_start_delay_list(Delay, EncoderNo)</code>
Parameters	Delay See set_ext_start_delay . EncoderNo See set_ext_start_delay .
Comments	<ul style="list-style-type: none"> If <code>EncoderNo > 1</code>, then <code>set_ext_start_delay_list</code> is replaced by a list_nop (get_last_error return code <code>RTC5_PARAM_ERROR</code>).
RTC4→ RTC5	See set_ext_start_delay .
References	set_ext_start_delay .

Ctrl Command	set_firstpulse_killer
Function	Defines the length of the FirstPulseKiller signal for a YAG laser.
Call	<code>set_firstpulse_killer(Length)</code>
Parameter	Length Length of the FirstPulseKiller signal as an unsigned 32-bit value. <i>1 bit equals 1/64 μs.</i> Allowed range: [0 ... +(2 ²⁶ -1)].
Comments	<ul style="list-style-type: none"> Values over (2²⁶-1) are clipped. The time base for the FirstPulseKiller signal is always 64 MHz. 1 bit equals 1/64 μs. The signal level is defined by set_laser_control. For YAG mode 2, the Q-Switch delay is also correspondingly altered (see page 150). In CO₂ mode, the command set_firstpulse_killer has no effect. The laser control mode has to be set by the command set_laser_mode. Refer to chapter 7.4 "Laser Control", page 144 for details. To set the Q-Switch pulse length and period, use the commands set_laser_pulses_ctrl, set_laser_pulses or set_laser_timing.
RTC4→ RTC5	Essentially identical functionality (except for the extended range of values), however: <code>Length</code> must be specified in units of 1/64 μs with the RTC5; in RTC4 compatibility mode it must be specified in units of 1/8 μs (as with the RTC4). In RTC4 compatibility mode, the RTC5 multiplies the specified value for <code>Length</code> by 8 (the allowed range of values is correspondingly reduced).
References	set_laser_mode , set_laser_timing

Undelayed Short List Command	set_firstpulse_killer_list
Function	Same as set_firstpulse_killer , but a list command.
Call	<code>set_firstpulse_killer_list(Length)</code>
Parameter	Length Length of the FirstPulseKiller signal as an unsigned 32-bit value. <i>1 bit equals 1/64 μs.</i> Allowed range: [0 ... +(2 ²⁶ -1)].
RTC4→ RTC5	as set_firstpulse_killer

Normal List Command	set_fly_2d				
Function	Activates Processing-on-the-fly correction for compensation of a linear workpiece-movement in two dimensions (based on the encoder values transferred to the RTC5 by encoder counters Encoder0 and Encoder1 and corrected by a special 2D table) and sets the corresponding scaling factors. The current encoder values get added to the previous reference values of 2D encoder compensation and the sums get stored as the new reference values (see notes). The encoder counters get reset to zero.				
Restriction	If the Processing-on-the-fly option is not enabled, then the command terminates the Processing-on-the-fly process (even though it could not have been activated).				
Call	<code>set_fly_2d(ScaleX, ScaleY)</code>				
Parameters	<table> <tr> <td>ScaleX</td> <td>Scaling factor for the X direction (encoder counter Encoder0) in <i>bits/count</i> (as a 64-bit IEEE floating point value). Allowed range: $1/256 \leq \text{ScaleX} \leq 16000.0$.</td> </tr> <tr> <td>ScaleY</td> <td>Scaling factor for the Y direction (encoder counter Encoder1) in <i>bits/count</i> (as a 64-bit IEEE floating point value). Allowed range: $1/256 \leq \text{ScaleY} \leq 16000.0$.</td> </tr> </table>	ScaleX	Scaling factor for the X direction (encoder counter Encoder0) in <i>bits/count</i> (as a 64-bit IEEE floating point value). Allowed range: $1/256 \leq \text{ScaleX} \leq 16000.0$.	ScaleY	Scaling factor for the Y direction (encoder counter Encoder1) in <i>bits/count</i> (as a 64-bit IEEE floating point value). Allowed range: $1/256 \leq \text{ScaleY} \leq 16000.0$.
ScaleX	Scaling factor for the X direction (encoder counter Encoder0) in <i>bits/count</i> (as a 64-bit IEEE floating point value). Allowed range: $1/256 \leq \text{ScaleX} \leq 16000.0$.				
ScaleY	Scaling factor for the Y direction (encoder counter Encoder1) in <i>bits/count</i> (as a 64-bit IEEE floating point value). Allowed range: $1/256 \leq \text{ScaleY} \leq 16000.0$.				
Comments	<ul style="list-style-type: none"> • ScaleX and ScaleY can be negative depending on the motion direction of the work-piece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction (e.g. determination of the scaling factor or deactivating Processing-on-the-fly correction), see the chapter 8.7 "Processing-on-the-fly (Optional)", page 199. For set_fly_2d command usage, see the chapter 8.7.4 "Compensation of 2D Motions", page 206. • If unallowed parameter values are supplied (e.g. for <code>ScaleX = 0</code>), then set_fly_2d does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by set_fly_2d (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed vector or arc command (without "set_fly_2d" Processing-on-the-fly correction). However, Processing-on-the-fly correction successfully activated by set_fly_2d switches off any other Processing-on-the-fly correction and does itself get switched off by any other Processing-on-the-fly command, even if that other command contains unallowed parameters (see section "Overview", page 200). • When using set_control_mode, do not set bit#9, which would result in the encoder values getting reset only after the subsequent external start trigger. Otherwise the reference values of 2D encoder compensation are lost. • Do not intermediately call set_fly_x or set_fly_y to switch on the Processing-on-the-fly application if you intend to use set_fly_2d in conjunction with 2D encoder compensation for an XY stage, because here too the reference values are lost. • If no correction table for 2D encoder compensation has yet been loaded onto the board (see load_fly_2d_table), then the encoder values are used without correction. 				

Normal List Command	set_fly_2d
Comments (cont'd)	<ul style="list-style-type: none"> If a coordinate transformation in the virtual image field is active, then it is applied to the entire virtual image field <i>before</i> Processing-on-the-fly correction (see the section "Coordinate Transformations in the Virtual Image Field", page 208). You can also use the activate_fly_2d command to switch on set_fly_2d Processing-on-the-fly correction.
RTC4→ RTC5	<p>New command.</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified scaling factors by 16 (the allowed range of values is correspondingly reduced).</p>
Version info	Available as of version DLL 536, OUT 536.
References	init_fly_2d , load_fly_2d_table , get_fly_2d_offset , activate_fly_2d , activate_fly_xy

Undelayed Short List Command	set_fly_limits
Function	Defines the boundaries of the customer-defined monitoring area for Processing-on-the-fly applications.
Call	<code>set_fly_limits(Xmin, Xmax, Ymin, Ymax)</code>
Parameters	Xmin, Area boundaries as signed 32-bit values. Xmax, Allowed value range: [-524,288 ... 524,287]. Ymin, Ymax Out-of-range values are edge-clipped.
Comments	<ul style="list-style-type: none"> For command usage, see "Customer-Defined Monitoring Area and Conditional Command Execution (as of Version DLL 525, OUT 527)", Seite 214. During initialization (with load_program_file), the boundary limits get set to the following default values: Xmin = Ymin = -524288 and Xmax = Ymax = +524287. Boundary limits specified using the parameter value 0 also get set to the above-mentioned default values.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 525, OUT 527.
References	get_marking_info



Undelayed Short List Command	set_fly_limits_z
Function	Defines the boundaries of the customer-defined monitoring range for Z-axis Processing-on-the-fly applications.
Call	<code>set_fly_limits_z(Zmin, Zmax)</code>
Parameters	<p>Zmin, Boundaries as signed 32-bit values. Zmax Allowed value range: [-32768 ... 32767]. Out-of-range values are edge-clipped.</p>
Comments	<ul style="list-style-type: none"> For command usage, see also chapter 8.7.9 "Monitoring Processing-on-the-fly Corrections", page 213. During initialization (with load_program_file), the boundary limits get set to the following default values: Zmin = -32768 and Zmax = +32767. Boundary limits specified using the parameter value 0 also get set to the above-mentioned default values.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 531, OUT 532.
References	set_fly_limits , get_marking_info



Normal List Command	set_fly_rot
Function	Activates Processing-on-the-fly correction for compensation of workpiece rotary movement (based on angular position values transferred to the RTC5 by encoder counter Encoder0) and sets the corresponding <code>Resolution</code> parameter. Encoder counter Encoder0 may be reset (see comments).
Restriction	If the Processing-on-the-fly option is not enabled, then the command terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_rot(Resolution)</code>
Parameter	<code>Resolution</code> number of steps per revolution (as a 64-bit IEEE floating point value). Allowed range: $ Resolution > 100.0$.
Comments	<ul style="list-style-type: none"> • <code>Resolution</code> can be negative depending on the rotation direction of the workpiece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction and determining the <code>Resolution</code> parameter, see chapter 8.7 "Processing-on-the-fly (Optional)", page 199. • Before executing the command <code>set_fly_rot</code>, you should define the rotation center for Processing-on-the-fly correction by <code>set_rot_center</code> or <code>set_rot_center_list</code>. Otherwise, the default value (0 0) is applied. • If unallowed parameter values are supplied (e.g. for <code>Resolution = 0</code>), then <code>set_fly_rot</code> does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by <code>set_fly_rot</code> (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed vector or arc command (without "set_fly_rot" Processing-on-the-fly correction). • The various Processing-on-the-fly corrections cannot be arbitrarily combined (see the page 200). • For deactivating Processing-on-the-fly correction, see page 209. • By <code>set_control_mode</code>, bit#9 can be set in advance to specify whether encoder counter Encoder0 should reset immediately by <code>set_fly_rot</code> or only after the subsequent external start trigger (i.e. by the subsequent external start signal or <code>simulate_ext_start</code> or <code>simulate_ext_start_ctrl</code> command, possibly postponed by a track delay set by <code>simulate_ext_start</code>, <code>set_ext_start_delay</code> or <code>set_ext_start_delay_list</code>; see also <code>set_control_mode</code>, bit#2).
RTC4→RTC5	Unchanged functionality.
Version info	Last change with version OUT 515.
References	set_rot_center , set_rot_center_list , fly_return , get_encoder , set_fly_rot_pos

Normal List Command	set_fly_rot_pos
Function	Activates Processing-on-the-fly correction for compensation of workpiece or scan system rotary movement (based on angular position values transferred to the RTC5 by the McBSP/SPI interface). It thereby sets the corresponding <code>Resolution</code> parameter.
Restriction	If the Processing-on-the-fly option is not enabled, then the command terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_rot_pos(Resolution)</code>
Parameter	<code>Resolution</code> number of steps per revolution (as a 64-bit IEEE floating point value). Allowed range: $ Resolution > 100.0$.
Comments	<ul style="list-style-type: none"> • <code>Resolution</code> can be negative depending on the rotation direction of the workpiece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction and determining the <code>Resolution</code> parameter, see chapter 8.7 "Processing-on-the-fly (Optional)", page 199. • Before executing the command <code>set_fly_rot_pos</code>, you should define the rotation center for Processing-on-the-fly correction by <code>set_rot_center</code> or <code>set_rot_center_list</code>. Otherwise, the default value (0 0) is applied. • If unallowed parameter values are supplied (e.g. for <code>Resolution = 0</code>), then <code>set_fly_rot_pos</code> does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by <code>set_fly_rot_pos</code> (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed vector or arc command (without "set_fly_rot_pos" Processing-on-the-fly correction). • The various Processing-on-the-fly corrections cannot be arbitrarily combined (see the page 200). • For deactivating Processing-on-the-fly correction, see page 209. • The McBSP/SPI interface cannot be simultaneously used for both Processing-on-the-fly applications and online positioning (see page 188). • The McBSP/SPI interface always ignores the first FrameSync signal after a <code>load_program_file</code> or <code>mcbsp_init</code>, so available data is not transmitted (see page 56).
RTC4 → RTC5	Unchanged functionality.
Version info	Last change with version OUT 515.
References	set_rot_center , set_rot_center_list , fly_return , read_mcbsp , set_fly_rot



Ctrl Command	set_fly_tracking_error
Function	Activates or deactivates tracking error compensation of encoder values for Processing-on-the-fly applications.
Call	<code>set_fly_tracking_error(TrackingErrorX, TrackingErrorY)</code>
Parameters	TrackingErrorX, Tracking error in units of [10 µs] for each axis TrackingErrorY (signed 32-bit values). Allowed range: [0 ... 65535]. Excessive bits are ignored.
Comments	<ul style="list-style-type: none"> • Tracking error compensation is a first-approximation correction: $EC = E + (E - EP) * TrackingError,$ whereby E is the current encoder value, EP is that of the previous cycle and EC is the compensated encoder value (used for Processing-on-the-fly correction). • To avoid step-like galvanometer movements, the used encoders should have a sufficiently high resolution (counts per 10 µs cycle). • If TrackingErrorX, Y = 0 (initialization value), then compensation is switched off. • The commands store_encoder, read_encoder and get_encoder still use the original, uncorrected encoder values. • This compensation is practical only for linear motions (chapter 8.7.2 "Compensation of Linear Movements", page 201), not for rotary motions (chapter 8.7.3 "Compensation of Rotary Movements", page 204). For the latter, TrackingErrorX, Y should be set to 0. • If the encoders get reset by the start trigger (set_control_mode(bit #9 = 1)), then the output value of the first 10 µs cycle might be incorrect to an unforeseen extent. Resets by the set_fly_x and set_fly_y commands automatically wait internally for an empty cycle. • This compensation only functions for Processing-on-the-fly applications with encoders, but not with position signals by the McBSP/SPI interface.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 533, OUT 534, RBF 524.



Normal List Command	set_fly_x
Function	Activates Processing-on-the-fly correction for compensation of a linear workpiece-movement in the X direction (based on position values transferred to the RTC5 by encoder counter Encoder0) and sets the corresponding scaling factor. Encoder counter Encoder0 may be reset (see comments).
Restriction	If the Processing-on-the-fly option is not enabled, then the command terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_x(ScaleX)</code>
Parameter	ScaleX scaling factor for the X direction (encoder counter Encoder0) in <i>bits/count</i> (as a 64-bit IEEE floating point value). Allowed range: $1/256 \leq \text{ScaleX} \leq 16000.0$
Comments	<ul style="list-style-type: none"> • ScaleX can be negative depending on the motion direction of the workpiece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction and determination of the scaling factor, see chapter 8.7 "Processing-on-the-fly (Optional)", page 199. • If unallowed parameter values are supplied (e.g. for <code>ScaleX = 0</code>), then <code>set_fly_x</code> does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by <code>set_fly_x</code> (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed vector or arc command (without "<code>set_fly_x</code>" Processing-on-the-fly correction). • The various Processing-on-the-fly corrections cannot be arbitrarily combined (see the page 200). • For deactivating Processing-on-the-fly correction, see page 209. • By <code>set_control_mode</code>, bit#9 can be set in advance to specify whether encoder counter Encoder0 should reset immediately by <code>set_fly_x</code> or only after the subsequent external start trigger (i.e. by the subsequent external start signal or <code>simulate_ext_start</code> or <code>simulate_ext_start_ctrl</code> command, possibly postponed by a track delay set by <code>simulate_ext_start</code>, <code>set_ext_start_delay</code> or <code>set_ext_start_delay_list</code>; see also <code>set_control_mode</code>, bit#2). • If a coordinate transformation in the virtual image field is active, then it is applied to the entire virtual image field <i>before</i> Processing-on-the-fly correction (see the section "Coordinate Transformations in the Virtual Image Field", page 208). • You can also switch on <code>set_fly_x</code>/<code>set_fly_y</code> Processing-on-the-fly correction by the <code>activate_fly_xy</code> command.
RTC4→RTC5	Unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, the RTC5 multiplies the specified scaling factor by 16 (the allowed range of values is correspondingly reduced).
Version info	Last change with version OUT 515.
References	fly_return , set_fly_y , get_encoder , set_fly_x_pos , set_fly_y_pos , activate_fly_xy , set_fly_2d



Normal List Command	set_fly_x_pos
Function	Activates Processing-on-the-fly correction for compensation of a linear workpiece or scan system movement in the X direction (based on position values transferred to the RTC5 by the McBSP/SPI interface); thereby sets the corresponding scaling factor.
Restriction	If the Processing-on-the-fly option is not enabled, then the command terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_x_pos(ScaleX)</code>
Parameter	ScaleX scaling factor for the X direction in <i>(RTC5)bits/(McBSP)bit</i> (as a 64-bit IEEE floating point value). Allowed range: $1/256 \leq \text{ScaleX} \leq 16000.0$.
Comments	<ul style="list-style-type: none"> • ScaleX can be negative depending on the motion direction of the workpiece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction and determination of the scaling factor, see chapter 8.7 "Processing-on-the-fly (Optional)", page 199. • If unallowed parameter values are supplied (e.g. for ScaleX = 0), then <code>set_fly_x_pos</code> does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by <code>set_fly_x_pos</code> (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed vector or arc command (without "<code>set_fly_x_pos</code>" Processing-on-the-fly correction). • The various Processing-on-the-fly corrections cannot be arbitrarily combined (see the page 200). • For deactivating Processing-on-the-fly correction, see page 209. • For one-dimensional correction (when only <code>set_fly_x_pos</code> is used), the McBSP/SPI interface provides a (signed) 32-bit value. In contrast, only a (signed) 16-bit value per axis is supplied for two-dimensional correction (<code>set_fly_x_pos</code> and <code>set_fly_y_pos</code>). Here, <code>set_fly_x_pos</code> uses the McBSP/SPI interface's lower 16 bits for the X value and <code>set_fly_y_pos</code> uses its upper 16-bits for the Y value. • The McBSP/SPI interface cannot be simultaneously used for both Processing-on-the-fly applications and online positioning (see page 188). • The McBSP/SPI interface always ignores the first FrameSync signal after a <code>load_program_file</code> or <code>mcbsp_init</code>, so available data is not transmitted (see page 56).
RTC4→RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified scaling factor by 16 (the allowed range of values is correspondingly reduced).
Version info	Last change with version OUT 515.
References	fly_return , set_fly_y_pos , read_mcbsp , set_fly_x , set_fly_y



Normal List Command	set_fly_y
Function	Activates Processing-on-the-fly correction for compensation of a linear workpiece-movement in the Y direction (based on position values transferred to the RTC5 by encoder counter Encoder1) and sets the corresponding scaling factor. Encoder counter Encoder1 may be reset (see comments).
Restriction	If the Processing-on-the-fly option is not enabled, then the command terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_y(ScaleY)</code>
Parameter	<p>Scale Scaling factor for the Y direction (encoder counter Encoder1) in <i>bits/count</i> (as a 64-bit IEEE floating point value). Allowed range: $1/256 \leq \text{ScaleY} \leq 16000.0$.</p>
Comments	<ul style="list-style-type: none"> • <code>ScaleY</code> can be negative depending on the motion direction of the workpiece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction and determination of the scaling factor, see chapter 8.7 "Processing-on-the-fly (Optional)", page 199. • If unallowed parameter values are supplied (e.g. for <code>ScaleY = 0</code>), then <code>set_fly_y</code> does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by <code>set_fly_y</code> (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed vector or arc command (without "<code>set_fly_y</code>" Processing-on-the-fly correction). • The various Processing-on-the-fly corrections cannot be arbitrarily combined (see the page 200). • For deactivating Processing-on-the-fly correction, see page 209. • By <code>set_control_mode</code>, bit#9 can be set in advance to specify whether encoder counter Encoder1 should reset immediately by <code>set_fly_y</code> or only after the subsequent external start trigger (i.e. by the subsequent external start signal or <code>simulate_ext_start</code> or <code>simulate_ext_start_ctrl</code> command, possibly postponed by a track delay set by <code>simulate_ext_start</code>, <code>set_ext_start_delay</code> or <code>set_ext_start_delay_list</code>; see also <code>set_control_mode</code>, bit#2). • If a coordinate transformation in the virtual image field is active, then it is applied to the entire virtual image field <i>before</i> Processing-on-the-fly correction (see the section "Coordinate Transformations in the Virtual Image Field", page 208). • You can also switch on <code>set_fly_x</code>/<code>set_fly_y</code> Processing-on-the-fly correction by the <code>activate_fly_xy</code> command.
RTC4→RTC5	Unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, the RTC5 multiplies the specified scaling factor by 16 (the allowed range of values is correspondingly reduced).
Version info	Last change with version OUT 515.
References	fly_return , set_fly_x , get_encoder , set_fly_x_pos , set_fly_y_pos , activate_fly_xy , set_fly_2d

Normal List Command	set_fly_y_pos
Function	Activates Processing-on-the-fly correction for compensation of a linear workpiece or scan system movement in the Y direction (based on position values transferred to the RTC5 by the McBSP/SPI interface); thereby sets the corresponding scaling factor.
Restriction	If the Processing-on-the-fly option is not enabled, then the command terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_y_pos(ScaleY)</code>
Parameter	ScaleY Scaling factor for the Y direction in <i>(RTC5)bits/(McBSP)bit</i> (as a 64-bit IEEE floating point value). Allowed range: $1/256 \leq \text{ScaleY} \leq 16000.0$.
Comments	<ul style="list-style-type: none"> • ScaleY can be negative depending on the motion direction of the workpiece. The restricted value range applies only to the absolute value. • For Processing-on-the-fly correction and determination of the scaling factor, see chapter 8.7 "Processing-on-the-fly (Optional)", page 199. • If unallowed parameter values are supplied (e.g. for ScaleY = 0), then set_fly_y_pos does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by set_fly_y_pos (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed vector or arc command (without "set_fly_y_pos" Processing-on-the-fly correction). • The various Processing-on-the-fly corrections cannot be arbitrarily combined (see page 200). • For deactivating Processing-on-the-fly correction, see chapter 8.7.5 "Deactivating Processing-on-the-fly Corrections", page 209. • For one-dimensional correction (when only set_fly_y_pos is used), the McBSP/SPI interface provides a (signed) 32-bit value. In contrast, only a (signed) 16-bit value per axis is supplied for two-dimensional correction (set_fly_x_pos and set_fly_y_pos). Here, set_fly_x_pos uses the McBSP/SPI interface's lower 16 bits for the X value and set_fly_y_pos uses its upper 16-bits for the Y value. • The McBSP/SPI interface cannot be simultaneously used for both Processing-on-the-fly applications and online positioning (see page 188). • The McBSP/SPI interface always ignores the first FrameSync signal after a load_program_file or mcbsp_init, so available data is not transmitted (see page 56).
RTC4 → RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified scaling factor by 16 (the allowed range of values is correspondingly reduced).
Version info	Last change with version OUT 515.
References	fly_return , set_fly_x_pos , read_mcbsp , set_fly_x , set_fly_y

Normal List Command	set_fly_z
Function	Activates Processing-on-the-fly correction for compensation of a linear workpiece-movement in the Z direction (based on position values transferred to the RTC5 by the specified encoder counter) and sets the corresponding scaling factor. The specified encoder counter may be reset (see comments).
Restriction	If the Processing-on-the-fly option is not enabled, then the command terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_fly_z(ScaleZ, EncoderNo)</code>
Parameters	<p>ScaleZ Scaling factor for the Z direction in <i>bits/count</i>. As a 64-bit IEEE floating point value. Allowed value range: $1/256 \leq \text{ScaleZ} \leq 16,000.0$.</p> <p>EncoderNo Number of the to-be-used encoder counter. As an unsigned 32-bit value. Allowed values: = 0: Encoder counter Encoder0. = 1: Encoder counter Encoder1.</p>
Comments	<ul style="list-style-type: none"> • ScaleZ can be negative depending on the motion direction of the workpiece. The restricted value range applies only to the absolute value. • The scaling factor ScaleZ is a 20 bit value (unlike the 16-bit full scale z-coordinates typical of marking and jump commands) because the RTC5 internally handles Z (and X and Y) coordinates as 20 bits full scale. Correspondingly, an activated RTC4 compatibility mode is also effective for ScaleZ. For Processing-on-the-fly correction and determination of the scaling factor, see chapter 8.7 "Processing-on-the-fly (Optional)", page 199. • If unallowed ScaleZ parameter values are supplied (for example, for ScaleZ = 0), then set_fly_z does not activate a Processing-on-the-fly correction or deactivates a Processing-on-the-fly correction previously activated by set_fly_z (but does not deactivate any other Processing-on-the-fly correction). The latter case leads to a jump (at jump speed) to the endpoint of the most recently executed vector or arc command (without "set_fly_z" Processing-on-the-fly correction). • For deactivating Processing-on-the-fly correction, see section "Notes on Usage", page 216. • By set_control_mode, bit #9 can be set in advance to specify whether the given encoder counter should reset immediately by set_fly_z or only after the subsequent external start trigger (that is, by the subsequent external start signal or simulate_ext_start or simulate_ext_start_ctrl command, possibly postponed by a track delay set by simulate_ext_start, set_ext_start_delay or set_ext_start_delay_list; see also set_control_mode, bit #2). • If EncoderNo > 1, set_fly_z is replaced by a list_nop (get_last_error return code RTC5_PARAM_ERROR).
RTC4→RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified scaling factor by 16 (the permissible range of values is correspondingly reduced).
Version info	Available as of version DLL 530, OUT 531.
References	fly_return_z



Ctrl Command	set_free_variable
Function	Sets a free variable to the desired value.
Call	set_free_variable(No, Value)
Parameters	<p>No Number of the free variable to be set as an unsigned 32-bit value. Allowed range: [0 ... 7]. Only the two least significant bits are evaluated.</p> <p>Value Desired variable value as an unsigned 32-bit value. Allowed range: [0 ... (2³²-1)].</p>
Comments	<ul style="list-style-type: none"> • See chapter 6.9.1 "Free Variables", page 102.
RTC4→RTC5	New command.
Version info	Available as of version DLL 531, OUT 532. Last change (version DLL 539, OUT 539): value range of parameter No increased to [0 ... 7].
References	set_free_variable_list , get_free_variable , set_trigger

Undelayed Short List Command	set_free_variable_list
Function	Same as set_free_variable , but a list command.
Call	set_free_variable_list(No, Value)
Parameters	<p>No See set_free_variable.</p> <p>Value See set_free_variable.</p>
Comments	<ul style="list-style-type: none"> • See set_free_variable.
RTC4→RTC5	New command.
Version info	Available as of version DLL 531, OUT 532.
References	set_free_variable



Ctrl Command	set_hi
Function	Defines gain and offset values for the galvanometer scanners of the scan system attached to the specified scan head connector.
Call	<code>set_hi(HeadNo, GalvoGainX, GalvoGainY, GalvoOffsetX, GalvoOffsetY)</code>
Parameters	<p>HeadNo Number of the scan head connector as an unsigned 32-bit value. Allowed values: = 1: Primary scan head connector. = 2: Secondary scan head connector (activation required).</p> <p>GalvoGainX, Gain values (as 64-bit IEEE floating point values). GalvoGainY Allowed range: [0.01...100].</p> <p>GalvoOffsetX, Offset values in bits (as 64-bit IEEE floating point values). GalvoOffsetY Allowed range: [-524288 ... 524287].</p>
Comments	<ul style="list-style-type: none"> For information on using the command, see section "Customer-Specific Calibration", page 227. The specified gain and offset values overwrite the values that were set by auto_cal and can themselves be overwritten by a subsequent call of auto_cal. Changed gain and offset values can trigger hard jumps. To avoid this, the transition is automatically performed at the predefined jump speed (see set_jump_speed). If a parameter value is invalid, then the command does not execute (get_last_error return code: RTC5_PARAM_ERROR). The command does likewise not execute (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list) or the board's INTERNAL-BUSY status is currently set. In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set). For RTC5_PARAM_ERROR, the BUSY status is not checked. Therefore the return codes RTC5_BUSY and RTC5_PARAM_ERROR do not arrive simultaneously. If the "second scan head control" option has not been enabled, values specified for the secondary scan head control has no effect.
RTC4→RTC5	New command.
References	auto_cal , get_hi_pos , write_hi_pos



Ctrl Command	set_input_pointer
Function	Opens the list buffer for writing with list commands and sets the input pointer to the specified (absolute) address in list memory ("List 1" or "List 2"). The next list command is stored at this address and all further list commands at the subsequent addresses in the selected list.
Call	<code>set_input_pointer(Pos)</code>
Parameter	Pos Position (absolute memory address) of the input pointer [0 ... (2^{20} -1)] as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The set_input_pointer command performs basically like the command set_start_list_pos (see the comments there). But set_input_pointer sets the input pointer based on a specified <i>absolute</i> memory address, whereas set_start_list_pos uses a specified list number and a <i>relative</i> memory address. For $\text{Pos} \geq \text{Mem1} + \text{Mem2}$ (see config_list), Pos is set to 0.
RTC4→ RTC5	Unchanged functionality.
References	set_start_list_pos, get_input_pointer

Undelayed Short List Command	set_io_cond_list
Function	Sets the bits of the 16-bit digital output port on the EXTENSION 1 socket connector that are set in the parameter MaskSet , if the current IOvalue at the 16-bit <i>input</i> port on the EXTENSION 1 socket connector meets the following condition: $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } ((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0}$ (i.e. if the bits specified in Mask1 are 1 and the bits specified in Mask0 are 0).
Call	<code>set_io_cond_list(Mask1, Mask0, MaskSet)</code>
Parameters	Mask1 , 16-bit masks as unsigned 32-bit values. Mask0 , Only the least significant 16 bits are evaluated. MaskSet
Comments	<ul style="list-style-type: none"> The command sets only those bits of the digital output port that are set in the parameter MaskSet and leaves the other bits unchanged. See also section "16-Bit Digital Input and Output", page 51 and chapter 9.3.2 "Conditional Command Execution", page 244.
Examples (Pascal)	<ul style="list-style-type: none"> Set bit #4 of the output port (DIGITAL OUT4), if bit #0 of the input port (DIGITAL IN0) is set and bits #1 to #3 (DIGITAL IN1...3) of the input port are not set: <code>set_io_cond_list(\$0001, \$000E, \$0010)</code> Always set bit #15 of the output port (and leave the other bits unchanged): <code>set_io_cond_list(0, 0, \$8000)</code>
RTC4→ RTC5	Unchanged functionality.
References	clear_io_cond_list, write_io_port, write_io_port_mask, get_io_status, read_io_port



Control Command	set_jump_mode
Function	Enables and activates or disables and deactivates jump mode for 2D jumps and sets the related parameters (before activation, a check might be performed, see also page 177).
Requirements	Enabling is only possible if a jump-tuning-equipped intelliSCAN, intellcube, intelliWELD or intelliDRILL scan system (with software version 2078 or higher) is attached to at least one of the two scan head connectors. Otherwise, the command has no effect.
Call	ErrorCode = set_jump_mode(Flag, Length, VA1, VA2, VB1, VB2, JA1, JA2, JB1, JB2)
Parameters	<p>Flag Switching flag as a signed 32-bit number. Allowed values: = -1: Jump mode is disabled. Afterward, switching the Flag by set_jump_mode_list is no longer possible. = 0: Jump mode is enabled (possibly only after successfully passing a check, see below), but deactivated. Afterward, switching the Flag by set_jump_mode_list is still possible. = 1: Jump mode is enabled and activated (possibly only after successfully passing a check, see below). Afterward, switching the Flag by set_jump_mode_list is still possible.</p> <p>Length Jump length limit (per axis) under which 2D jumps – even with activated jump mode – are performed in vector mode.</p> <p>VA1, VA2, Numbers (as unsigned 32-bit values) of the tunings that should be used for VB1, VB2, jump execution in jump mode: JA1, JA2, V: Vector tuning that is set at the end of a 2D jump. JB1, JB2 J: Jump tuning that is set at the beginning of a 2D jump. A: Primary scan head connector. B: Secondary scan head connector. 1: X axis (STATUS channel, galvanometer scanner 2). 2: Y axis (STATUS1 channel, galvanometer scanner 1). Allowed values: = -1: Tuning is neither checked nor used. = 0...3: After passing a check, tuning is used for jump mode (the allowed value range also depends on the number of tunings with which the attached scan system is equipped).</p>



Control Command	<code>set_jump_mode</code>
Result	<p>Error code as unsigned 32-bit value:</p> <ul style="list-style-type: none"> 0 No error: Flag successfully switched to 0 (jump mode deactivated, vector mode activated). 1 No error: Flag successfully switched to 1 (jump mode activated, vector mode deactivated). -1 Flag successfully (as requested) switched to -1 (jump mode deactivated and disabled). -2 Busy error: board BUSY or INTERNAL BUSY (<code>get_last_error</code> return code: RTC5_BUSY). -3 Board not responding: possibly no program loaded or PCI error (<code>get_last_error</code> return code: RTC5_TIMEOUT). -4 Access error: board reserved for another user program (<code>get_last_error</code> return code: RTC5_ACCESS_DENIED). > 1 Did not pass the check (see also notes). Flag is set to -1 (jump mode deactivated and disabled). The following is returned: Byte #0 = 255. Byte #1 = Error code for primary scan head connector. Byte #2 = Error code for secondary scan head connector. Byte #3 = 0. Whereby error code: =1: X axis (galvanometer scanner 2) not responding or no intelliSCAN, intelllicube, intelliWELD or intelliDRILL scan system (with software version 2078 or higher) attached. =2: Y axis (galvanometer scanner 1) not responding or no intelliSCAN, intelllicube, intelliWELD or intelliDRILL scan system (with software version 2078 or higher) attached. =4: no correction table assigned. =8: incorrect tuning number(s): incorrect type or unsuitable for rapid switching.
Notes	<ul style="list-style-type: none"> • For information on using this command, see chapter 8.1.5 "Jump Mode", page 176. • A check (see also "Requirements and Activation", Seite 177) is only performed if Flag = -1 (the initialization state) prior to the call and/or if the supplied tuning numbers do not match those stored on the board. Otherwise, only the flag is switched. For the check, the board must not be BUSY or INTERNAL BUSY, because meanwhile the to-be-returned data type changes and automatic laser control is deactivated (both get restored at the end of the command). Depending on results of the check, different error codes are returned (see above). In case of error, Flag gets set to -1 (jump mode deactivated and disabled). If the check is successful, then you can afterward (even by <code>set_jump_mode_list</code> during processing of a list) switch freely between the states Flag = 1 (jump mode activated, vector mode deactivated) and Flag = 0 (jump mode deactivated, vector mode activated) without another check having to be performed.



Control Command	<code>set_jump_mode</code>
Notes (cont'd)	<ul style="list-style-type: none"> Use <code>-1</code> as the tuning number if certain tunings should not be checked (e.g. because no intelliSCAN scan system is attached or specific tunings are not available – vector tuning, for example, is not needed in pure drilling applications) or if, after switching to jump tuning, it is not desirable to return to vector tuning. As a result, such tunings are neither checked nor switched on. If the “second scan head control” option has not been enabled, then the tuning numbers for the secondary scan head connector (B) is automatically set to <code>-1</code> (even if others were supplied). If all tuning numbers are <code>-1</code>, then <code>Flag</code> is set to <code>-1</code> (return value <code>-1</code>). Even after successful activation of jump mode (<code>Flag = 1</code>), the first servo switching only occurs after the first subsequent 2D jump (see page 176). If the currently set tuning then does not match the jump or vector tuning specified by <code>set_jump_mode</code>, then the first switching can take somewhat longer (approx. 250 ms), depending on the currently set tuning. You can determine ahead of time whether this is so by calling <code>set_jump_mode</code> using the currently set tuning as a parameter. If true (return value <code>> 1</code>, Error code = <code>2</code>), then you can achieve the desired operational sequence by calling <code>control_command</code> before the first 2D jump to set the tuning to one that was supplied by <code>set_jump_mode</code>.
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> Available as of version DLL 525. Changed with version DLL 530, OUT 531, RBF 522: changed error code for return values <code>> 1</code> (byte#1 and byte#2).
References	set_jump_mode_list , load_jump_table_offset , set_jump_table



Normal List Command	set_jump_mode_list
Function	Activates or deactivates and disables jump mode for 2D jumps.
Requirements	See set_jump_mode .
Call	<code>set_jump_mode_list(Flag)</code>
Parameters	Flag See set_jump_mode .
Notes	<ul style="list-style-type: none"> • For information on using the command, see chapter 8.1.5 "Jump Mode", page 176. • <code>set_jump_mode_list</code> functions like the control command <code>set_jump_mode</code> (see notes there) but, as a list command, has the following differences: <ul style="list-style-type: none"> – No tunings or jump length limits can be supplied by <code>set_jump_mode_list</code>. – <code>set_jump_mode_list</code> does not perform a check. Flag must have previously been successfully set to 0 or 1 by <code>set_jump_mode</code>. – Though jump mode can be deactivated and disabled by setting Flag to -1 by <code>set_jump_mode</code> or <code>set_jump_mode_list</code>, it can only be reactivated again by <code>set_jump_mode</code> (if the check is successful). If Flag was -1 prior to calling <code>set_jump_mode_list</code>, then the command has no effect.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 525.
Reference	set_jump_mode



Undelayed Short List Command	set_jump_speed
Function	Defines the jump speed for the following vector commands.
Call	<code>set_jump_speed(Speed)</code>
Parameter	Speed Jump speed in <i>bits per ms</i> (as a 64-bit IEEE floating point value). Allowed range: [1.6 ... 800000.0].
Comments	<ul style="list-style-type: none"> • By default a jump speed of 10000 <i>bits per ms</i> is preset. • The specified jump speed (or the preset jump speed, if no other value has been specified) is used for all subsequent jump commands until a new value is specified. • To obtain the actual jump speed v in the image plane (in <i>meters per second</i>), the specified speed value (in <i>bits per ms</i>) must be divided by the calibration factor K of the correction table (in <i>bits per mm</i>): $v_{jump} = \text{Speed} / K$ <p>The calibration factor K can be queried from the correction table by get_table_para or get_head_para.</p> <ul style="list-style-type: none"> • The set_jump_speed command is also available as the control command set_jump_speed_ctrl.
RTC4→RTC5	Unchanged functionality (except for the extended range of values). The image field coordinates for the X and Y axes (but not the Z axis) and all related parameters (e.g. jump speed or wobble amplitude) are specified as 20-bit values; in RTC4 compatibility mode they are specified as 16-bit values (as with the RTC4) and the RTC5 multiplies the specified value for Speed by 16 (the allowed range of values is correspondingly reduced to [0.1 ... 50000.0]).
References	jump_abs , jump_rel , set_mark_speed , set_jump_speed_ctrl

Ctrl Command	set_jump_speed_ctrl
Function	Same as set_jump_speed , but a control command.
Call	<code>set_jump_speed_ctrl(Speed)</code>
Parameter	Speed Jump speed in <i>bits per ms</i> (as a 64-bit IEEE floating point value). Allowed range: [1.6 ... 800000.0].
Comments	<ul style="list-style-type: none"> • The command is ignored (get_last_error return code: <code>RTC5_BUSY</code>) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list). In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set) or the board's INTERNAL-BUSY status is currently set.
RTC4→RTC5	New command. In RTC4 compatibility mode: as set_jump_speed .
References	set_jump_speed , set_mark_speed , set_mark_speed_ctrl



Control Command	set_jump_table
Function	Reads the jump delay table with 1024 unsigned 16-bit values stored at the supplied PC address and loads them onto the board as the jump delay table (see notes for get_jump_table).
Call	ErrorCode = set_jump_table(Addr)
Parameters	Addr PC Address of a 2048-byte area of PC main memory.
Result	Error code as an unsigned 32-bit value: 0 No error. 1 Busy error: board BUSY or INTERNAL BUSY (get_last_error return code: RTC5_BUSY). 4 Verify error: DSP memory error. 11 Driver error.
Notes	<ul style="list-style-type: none"> The command set_jump_table does <i>not</i> execute (get_last_error return code: RTC5_BUSY) if the board's BUSY status is currently set (list is being processed or was paused by pause_list) or if the board's INTERNAL BUSY status is set. In contrast, the command executes if a list was paused by set_wait (PAUSED status set).
RTC4→ RTC5	New command.
Version info	Available as of version DLL 525.
Reference	get_jump_table , load_jump_table_offset



Ctrl Command	set_laser_control																
Function	Defines and enables or disables the laser control signals.																
Call	<code>set_laser_control(Ctrl)</code>																
Parameter	<p>Ctrl (as an unsigned 32-bit value):</p> <table> <tr> <td>Bit #</td> <td>Description</td> </tr> <tr> <td>Bit #0 (LSB)</td> <td> <p>Pulse Switch Setting (does not apply neither to laser mode 4 nor to laser mode 6):</p> <p>The setting only affects those laser control signals (more precisely: those LASER1 or LASER2 "laser active" modulation pulses in CO₂ mode or LASER1 Q-Switch pulses in the YAG modes) that are not yet fully processed at completion of the LASERON signal, see figure 50 and figure 51.</p> <ul style="list-style-type: none"> = 0: The signals are cut off at the end of the LASERON signal. = 1: The final pulse fully executes despite completion of the LASERON signal. </td> </tr> <tr> <td>Bit #1</td> <td> <p>Phase shift of the laser control signals (does not apply neither to laser mode 4 nor to laser mode 6).</p> <ul style="list-style-type: none"> = 0: No phase shift. = 1: CO₂ mode: The LASER1 signal is exchanged with the LASER2 signal. YAG modes: The LASER1 is shifted back 180° (half a signal period). </td> </tr> <tr> <td>Bit #2</td> <td> <p>Enabling or disabling of laser control signals for "Laser active" operation</p> <ul style="list-style-type: none"> = 0: The "Laser active" laser control signals are enabled. = 1: The "Laser active" laser control signals are disabled (then the laser output ports are in the high impedance tristate mode). </td> </tr> <tr> <td>Bit #3</td> <td> <p>LASERON signal level.</p> <ul style="list-style-type: none"> = 0: The signal at the LASERON port is set to active-high. = 1: The signal at the LASERON port is set to active-low. </td> </tr> <tr> <td>Bit #4</td> <td> <p>LASER1/LASER2 signal level.</p> <ul style="list-style-type: none"> = 0: The signals at the LASER1 and LASER2 output ports are set to active-high. = 1: The signals at the LASER1 and LASER2 output ports are set to active-low. </td> </tr> <tr> <td>Bit #5</td> <td> <p>Determines for laser_on_pulses_list whether external signal pulses (at the LASER connector's DIGITAL IN1 digital input) are to be counted at rising or falling edges:</p> <ul style="list-style-type: none"> = 0: At the falling edge. = 1: At the rising edge. </td> </tr> <tr> <td>Bit #6</td> <td> <ul style="list-style-type: none"> = 0: Output synchronization is switched off (default setting). = 1: Output synchronization is switched on (see page 169). </td> </tr> </table>	Bit #	Description	Bit #0 (LSB)	<p>Pulse Switch Setting (does not apply neither to laser mode 4 nor to laser mode 6):</p> <p>The setting only affects those laser control signals (more precisely: those LASER1 or LASER2 "laser active" modulation pulses in CO₂ mode or LASER1 Q-Switch pulses in the YAG modes) that are not yet fully processed at completion of the LASERON signal, see figure 50 and figure 51.</p> <ul style="list-style-type: none"> = 0: The signals are cut off at the end of the LASERON signal. = 1: The final pulse fully executes despite completion of the LASERON signal. 	Bit #1	<p>Phase shift of the laser control signals (does not apply neither to laser mode 4 nor to laser mode 6).</p> <ul style="list-style-type: none"> = 0: No phase shift. = 1: CO₂ mode: The LASER1 signal is exchanged with the LASER2 signal. YAG modes: The LASER1 is shifted back 180° (half a signal period). 	Bit #2	<p>Enabling or disabling of laser control signals for "Laser active" operation</p> <ul style="list-style-type: none"> = 0: The "Laser active" laser control signals are enabled. = 1: The "Laser active" laser control signals are disabled (then the laser output ports are in the high impedance tristate mode). 	Bit #3	<p>LASERON signal level.</p> <ul style="list-style-type: none"> = 0: The signal at the LASERON port is set to active-high. = 1: The signal at the LASERON port is set to active-low. 	Bit #4	<p>LASER1/LASER2 signal level.</p> <ul style="list-style-type: none"> = 0: The signals at the LASER1 and LASER2 output ports are set to active-high. = 1: The signals at the LASER1 and LASER2 output ports are set to active-low. 	Bit #5	<p>Determines for laser_on_pulses_list whether external signal pulses (at the LASER connector's DIGITAL IN1 digital input) are to be counted at rising or falling edges:</p> <ul style="list-style-type: none"> = 0: At the falling edge. = 1: At the rising edge. 	Bit #6	<ul style="list-style-type: none"> = 0: Output synchronization is switched off (default setting). = 1: Output synchronization is switched on (see page 169).
Bit #	Description																
Bit #0 (LSB)	<p>Pulse Switch Setting (does not apply neither to laser mode 4 nor to laser mode 6):</p> <p>The setting only affects those laser control signals (more precisely: those LASER1 or LASER2 "laser active" modulation pulses in CO₂ mode or LASER1 Q-Switch pulses in the YAG modes) that are not yet fully processed at completion of the LASERON signal, see figure 50 and figure 51.</p> <ul style="list-style-type: none"> = 0: The signals are cut off at the end of the LASERON signal. = 1: The final pulse fully executes despite completion of the LASERON signal. 																
Bit #1	<p>Phase shift of the laser control signals (does not apply neither to laser mode 4 nor to laser mode 6).</p> <ul style="list-style-type: none"> = 0: No phase shift. = 1: CO₂ mode: The LASER1 signal is exchanged with the LASER2 signal. YAG modes: The LASER1 is shifted back 180° (half a signal period). 																
Bit #2	<p>Enabling or disabling of laser control signals for "Laser active" operation</p> <ul style="list-style-type: none"> = 0: The "Laser active" laser control signals are enabled. = 1: The "Laser active" laser control signals are disabled (then the laser output ports are in the high impedance tristate mode). 																
Bit #3	<p>LASERON signal level.</p> <ul style="list-style-type: none"> = 0: The signal at the LASERON port is set to active-high. = 1: The signal at the LASERON port is set to active-low. 																
Bit #4	<p>LASER1/LASER2 signal level.</p> <ul style="list-style-type: none"> = 0: The signals at the LASER1 and LASER2 output ports are set to active-high. = 1: The signals at the LASER1 and LASER2 output ports are set to active-low. 																
Bit #5	<p>Determines for laser_on_pulses_list whether external signal pulses (at the LASER connector's DIGITAL IN1 digital input) are to be counted at rising or falling edges:</p> <ul style="list-style-type: none"> = 0: At the falling edge. = 1: At the rising edge. 																
Bit #6	<ul style="list-style-type: none"> = 0: Output synchronization is switched off (default setting). = 1: Output synchronization is switched on (see page 169). 																



Ctrl Command	set_laser_control
Parameter (cont'd)	<p>Bit #7 = 0: The constant pulse length mode is switched off (default setting). = 1: The constant pulse length mode is switched on (see page 157 and set_pulse_picking_length).</p> <p>Bit #8... Reserved.</p> <p>Bit #15</p> <p>Bit #16 PowerOK of head A's X axis is used for laser-signal auto-suppression.</p> <p>Bit #17 TempOK of head A's X axis is used for laser-signal auto-suppression.</p> <p>Bit #18 PosAck of head A's X axis is used for laser-signal auto-suppression.</p> <p>Bit #19 PowerOK of head A's Y axis is used for laser-signal auto-suppression.</p> <p>Bit #20 TempOK of head A's Y axis is used for laser-signal auto-suppression.</p> <p>Bit #21 PosAck of head A's Y axis is used for laser-signal auto-suppression.</p> <p>Bit #22 PowerOK of head B's X axis is used for laser-signal auto-suppression.</p> <p>Bit #23 TempOK of head B's X axis is used for laser-signal auto-suppression.</p> <p>Bit #24 PosAck of head B's X axis is used for laser-signal auto-suppression.</p> <p>Bit #25 PowerOK of head B's Y axis is used for laser-signal auto-suppression.</p> <p>Bit #26 TempOK of head B's Y axis is used for laser-signal auto-suppression.</p> <p>Bit #27 PosAck of head B's Y axis is used for laser-signal auto-suppression.</p> <p>Bit #28 = 1: In case of error, automatic monitoring (laser-signal auto-suppression) automatically generates a /STOP signal (list stops, laser control signals get permanently switched off).</p> <p>Bit #29... Reserved.</p> <p>Bit #31</p>



Ctrl Command	set_laser_control
Comments	<ul style="list-style-type: none"> In the default setting (after a reset), all bits are set to 0. After a hardware reset, however, the settings only become effective following the first-time call of set_laser_control. Prior to this, all laser signal outputs (LASERON, LASER1 and LASER2) are in the (high-impedance) tristate mode. TTL states (LOW or HIGH) only become available when set_laser_control is called to define the desired TTL level – see also page 144. After load_program_file, which deactivates the laser control signals (but otherwise does not change the signal levels at the laser signal output ports), set_laser_control must be called for first-time activation. Therefore, during the RTC5 startup section at the beginning of each RTC5 user program, set_laser_control should be called after load_program_file. For the RTC5's predecessors, the laser signal levels were defined by (solderable) jumpers. The RTC5 lets users control them completely by software. The command get_startstop_info queries the current status of the laser control signals (Bit #9) and whether the signals are set to active-high or active-low (Bit #13). Enabling and disabling of laser control signals can also be achieved by the command enable_laser or disable_laser. Even if the laser control signals were enabled with set_laser_control or enable_laser, they are not outputted without further commands (see page 144). The phase shift of the laser control signals (Bit #1 = 1) can, for example, be set for the softstart mode (see page 152) or the pixel mode (see page 217). For usage of the bits #16 – 28 for laser-signal auto-suppression see page 146.
RTC4→RTC5	New command.
Version info	<p>Changed with version DLL 515, OUT 514, RBF 512: bit#5 added for laser_on_pulses_list.</p> <p>Changed with version DLL 530, OUT 531, RBF 522: bit#6 added for output synchronization.</p> <p>Changed with version DLL 533, OUT 534, RBF 524: bit#7 added for constant pulse length mode.</p> <p>Changed with version DLL 536, OUT 536: bit#28 added for /STOP in case of error.</p>
References	set_laser_mode



Undelayed Short List Command	set_laser_delays				
Function	Sets the LaserOn delay and the LaserOff delay.				
Call	<code>set_laser_delays(LaserOnDelay, LaserOffDelay)</code>				
Parameters	<table> <tr> <td>LaserOnDelay</td> <td>LaserOn Delay as a signed 32-bit value. <i>1 bit equals 0.5 µs.</i> Allowed range: $[-2^{31} \dots + (2^{21}-1)]$.</td> </tr> <tr> <td>LaserOffDelay</td> <td>LaserOff Delay as an unsigned 32-bit value. <i>1 bit equals 0.5 µs.</i> Allowed range: $[0 \dots + (2^{21}-1)]$.</td> </tr> </table>	LaserOnDelay	LaserOn Delay as a signed 32-bit value. <i>1 bit equals 0.5 µs.</i> Allowed range: $[-2^{31} \dots + (2^{21}-1)]$.	LaserOffDelay	LaserOff Delay as an unsigned 32-bit value. <i>1 bit equals 0.5 µs.</i> Allowed range: $[0 \dots + (2^{21}-1)]$.
LaserOnDelay	LaserOn Delay as a signed 32-bit value. <i>1 bit equals 0.5 µs.</i> Allowed range: $[-2^{31} \dots + (2^{21}-1)]$.				
LaserOffDelay	LaserOff Delay as an unsigned 32-bit value. <i>1 bit equals 0.5 µs.</i> Allowed range: $[0 \dots + (2^{21}-1)]$.				
Comments	<ul style="list-style-type: none"> Values over $(2^{21}-1)$ are clipped. The delays can be freely chosen within the allowed ranges. If <code>LaserOffDelay < LaserOnDelay</code>, overlaps of LaserOn and LaserOff are automatically prevented during processing of short vectors (see "Automatic Delay Adjustments" on page 121). The <code>LaserOffDelay > LaserOnDelay</code> condition required by the RTC5's predecessor boards is therefore unnecessary. Observe the notes in chapter 7.2.1 "Laser Delays", page 111. If the LaserOn Delay is negative, the total marking time is extended. The optional XY2-100 converter (see page 43) introduces a 10 µs runtime latency to scan-system control. This runtime latency can be compensated by increasing the LaserOn Delay and LaserOff Delay by 10 µs each. The default setting after <code>load_program_file</code> corresponds to <code>set_laser_delays(20, 20)</code>. 				
RTC4→RTC5	Essentially identical functionality (except for the extended range of values), however: <code>LaserOnDelay</code> and <code>LaserOffDelay</code> must be specified in units of 0.5 µs with the RTC5; in RTC4 compatibility mode they are specified in units of 1 µs (as with the RTC4). In RTC4 compatibility mode, the RTC5 multiplies the specified delays by 2 (the allowed range of values is correspondingly reduced).				
References	set_scanner_delays				



Ctrl Command	set_laser_mode
Function	Selects the laser control mode of the RTC5.
Call	<code>set_laser_mode(Mode)</code>
Parameter	<p>Mode As an unsigned 32-bit value.</p> <ul style="list-style-type: none"> = 0: CO₂ mode. = 1: YAG mode 1. = 2: YAG mode 2. = 3: YAG mode 3. = 4: Laser mode 4. = 5: YAG mode 5. = 6: Laser mode 6.
Comments	<ul style="list-style-type: none"> • The available laser signals depend on the selected laser control mode, see also chapter 7.4 "Laser Control", page 144. • The command should be used only once at the program start, see also "Start of Operation", page 69.
RTC4→RTC5	YAG mode 5 and laser mode 6 were introduced for the RTC5. Otherwise unchanged functionality.
References	set_laser_control , set_laser_pulses_ctrl , set_laser_pulses , set_laser_timing , set_firstpulse_killer , set_firstpulse_killer_list , set_standby , set_standby_list , set_qswitch_delay , set_qswitch_delay_list

Ctrl Command	set_laser_off_default
Function	Defines the default output value for the ANALOG OUT1 and ANALOG OUT2 ports, as well as the RTC5's 8-bit digital output port.
Call	<code>set_laser_off_default(AnalogOut1, AnalogOut2, DigitalOut)</code>
Parameters	<p>AnalogOut1, 12-bit values for analog output ports ANALOG OUT1 and ANALOG OUT2 AnalogOut2 (see also page 48) as unsigned 32-bit values. Higher bits are ignored (exception: AnalogOut1/2 = “-1”, see comments for set_port_default).</p> <p>DigitalOut 8-bit value for the 8-bit digital output port (see also page 52) as an unsigned 32-bit value. Higher bits are ignored (exception: DigitalOut = “-1”, see comments for set_port_default).</p>
Comments	<ul style="list-style-type: none"> • The default values can also be defined by the command set_port_default (see also all comments there).
RTC4→RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified values for AnalogOut1 and AnalogOut2 by 4 (the RTC4's analog outputs only handled 10-bit values).
References	set_port_default



Ctrl Command	set_laser_pin_out
Function	Sends a value to the two digital outputs of the laser connector.
Call	<code>set_laser_pin_out(Pins)</code>
Parameter	<p>Pins Output value (DIGITAL OUT1 and DIGITAL OUT2) as unsigned 32-bit value.</p> <p>Bit # 0: DIGITAL OUT1.</p> <p>Bit # 1: DIGITAL OUT2.</p> <p>Bits # 2...31:Reserved.</p>
Comments	<ul style="list-style-type: none"> • See also chapter 9.1.3 "2 Bit Digital Output Port", page 233.
RTC4→ RTC5	New command.
References	set_laser_pin_out_list , get_laser_pin_in

Undelayed Short List Command	set_laser_pin_out_list
Function	Same as set_laser_pin_out , but a list command.
Call	<code>set_laser_pin_out_list(Pins)</code>
Parameter	<p>Pins Output value (DIGITAL OUT1 and DIGITAL OUT2) as unsigned 32-bit value.</p> <p>Bit # 0: DIGITAL OUT1.</p> <p>Bit # 1: DIGITAL OUT2.</p> <p>Bits # 2...31:Reserved.</p>
Comments	<ul style="list-style-type: none"> • See set_laser_pin_out.
RTC4→ RTC5	New command.
References	set_laser_pin_out



Delayed Short List Command	set_laser_pulses
Function	Defines the output period and the pulse lengths for the laser signals LASER1 and LASER2 for "laser active" operation.
Call	<code>set_laser_pulses(HalfPeriod, PulseLength)</code>
Parameters	<p>HalfPeriod <i>Half of the output period in bits as an unsigned 32-bit value. 1 bit equals 1/64 µs. Allowed range: [0 ... (2³²-1)].</i></p> <p>PulseLength Pulse length of the laser signals LASER1 and LASER2 in <i>bits</i> as an unsigned 32-bit value. <i>1 bit equals 1/64 µs. Allowed range: [0 ... (2³²-1)].</i></p>
	<ul style="list-style-type: none"> • Note that <i>half</i> the period length must be specified for <code>HalfPeriod</code> (see figure 50 and figure 51). • If <code>HalfPeriod</code> = 0 and/or <code>PulseLength</code> = 0, no laser signals are outputted. • If the softstart mode is used (see page 152), <code>HalfPeriod</code> should not be smaller than 104 (this is not automatically checked). This value corresponds to a laser pulse frequency of approx. 308 kHz. • If <code>PulseLength</code> is larger than the output period ($2 \times \text{HalfPeriod}$), the laser is permanently on. • The signal level is defined by <code>set_laser_control</code>. • The <code>set_laser_pulses</code> command is also available as the control command <code>set_laser_pulses_ctrl</code>. • The <code>set_laser_pulses</code> command is largely identical to the RTC4's <code>set_laser_timing</code> command, but has less parameters.
RTC4 → RTC5	New command. In RTC4 compatibility mode, parameter values must be specified in units of 1/8 µs (the 1 µs time base – see <code>set_laser_timing</code> – is no longer supported). They are then internally converted to 1/64 µs units (i.e. multiplied by 8). The allowed range of values is correspondingly smaller (the smallest reasonable value for <code>HalfPeriod</code> with softstart mode is then 13).
References	<code>set_laser_pulses_ctrl</code> , <code>set_laser_timing</code>



Ctrl Command	set_laser_pulses_ctrl
Function	Same as set_laser_pulses , but a control command.
Call	<code>set_laser_pulses_ctrl(HalfPeriod, PulseLength)</code>
Parameters	<p>HalfPeriod <i>Half of the output period in bits as an unsigned 32-bit value. 1 bit equals 1/64 µs. Allowed range: [0 ... (2³²-1)].</i></p> <p>PulseLength Pulse length of the laser signals LASER1 and LASER2 in bits as an unsigned 32-bit value. <i>1 bit equals 1/64 µs. Allowed range: [0 ... (2³²-1)].</i></p>
Comments	<ul style="list-style-type: none"> • see set_laser_pulses
RTC4→ RTC5	New command. In RTC4 compatibility mode: as set_laser_pulses .
References	set_laser_pulses, set_laser_timing



Delayed Short List Command	set_laser_timing
Function	Defines the output period and the pulse lengths for the laser signals LASER1 and LASER2 for "laser active" operation.
Call	<code>set_laser_timing(HalfPeriod, PulseLength1, PulseLength2, TimeBase)</code>
Parameters	<p>HalfPeriod <i>Half of the output period in bits as an unsigned 32-bit value.</i></p> <ul style="list-style-type: none"> • In RTC5 mode: <i>1 bit equals 1/64 µs.</i> Allowed range: [0 ... (2³²-1)]. • In RTC4 compatibility mode: <i>1 bit equals 1/8 µs or 1 µs, depending on the selected clock frequency.</i> With respect to the specified <code>TimeBase</code>, the value is converted to an integer-multiple of 1/64 µs. The allowed range is correspondingly smaller. <p>PulseLength1 <i>Pulse lengths of the laser signals LASER1 and LASER2 in bits as an unsigned 32-bit value.</i></p> <ul style="list-style-type: none"> • In RTC5 mode: <i>1 bit equals 1/64 µs.</i> Allowed range: [0 ... (2³²-1)]. • In RTC4 compatibility mode: <i>1 bit equals 1/8 µs or 1 µs, depending on the selected clock frequency.</i> With respect to the specified <code>TimeBase</code>, the value is converted to an integer-multiple of 1/64 µs. The allowed range is correspondingly smaller. <p>(PulseLength2) <i>(As an unsigned 32-bit value.)</i> The value of <code>PulseLength2</code> is ignored and is no longer used. With the RTC5, the pulse lengths of laser signals LASER1 and LASER2 are always identical and are definable by <code>PulseLength1</code>.</p> <p>TimeBase <i>As an unsigned 32-bit value.</i></p> <ul style="list-style-type: none"> • In RTC5 mode, the value is ignored and the clock frequency is fixed at 64 MHz. <i>1 bit equals 1/64 µs.</i> • In RTC4 compatibility mode, the <code>TimeBase</code> value is handled as follows: =0: sets the time base to 1 MHz. <i>1 bit equals 1 µs.</i> ≠0: sets the time base to 8 MHz. <i>1 bit equals 1/8 µs.</i>

Delayed Short List Command	set_laser_timing
Comments	<ul style="list-style-type: none"> Note that <i>half</i> the period length must be specified for <code>HalfPeriod</code> (see figure 50 and figure 51). If <code>HalfPeriod</code> = 0 and/or <code>PulseLength1</code> = 0, no laser signals are outputted. If the softstart mode is used (see page 152), <code>HalfPeriod</code> should not be smaller than 104 (this is not automatically checked). This value corresponds to a laser pulse frequency of approx. 308 kHz. If <code>PulseLength1</code> is larger than the output period ($2 \times \text{HalfPeriod}$), the laser is permanently on. The time base setting applies only for the parameters of this command. For RTC4 compatibility mode, SCANLAB generally recommends setting the time base to 8 MHz. In this mode, a time base of 1 MHz should only be chosen if necessary. Refer to chapter 7.4 "Laser Control", page 144, for details. In RTC5 mode, the command <code>set_laser_timing</code> is identical to <code>set_laser_pulses</code>. The signal level is defined by <code>set_laser_control</code>. In laser mode 4 and laser mode 6, the time base for signals LASER1 and LASER2 is independent of <code>TimeBase</code> and always 1/64 μs (in RTC5 mode) or 1/8 μs (in RTC4 compatibility mode).
RTC4 → RTC5	<ul style="list-style-type: none"> With the RTC5, the pulse lengths of laser signals LASER1 and LASER2 are always identical. Clock frequency: <ul style="list-style-type: none"> In RTC5 mode: fixed clock frequency of 64 MHz. For the RTC4 and in RTC4 compatibility mode: 1 MHz or 8 MHz.
References	set_laser_pulses_ctrl , set_laser_pulses , set_laser_mode , set_firstpulse_killer , set_firstpulse_killer_list , set_standby , set_standby_list

Undelayed Short List Command	set_list_jump
Function	Execution produces an unconditional jump to the specified address within the list buffer. The next command there is executed immediately without delay.
Call	<code>set_list_jump(Pos)</code>
Parameter	<code>Pos</code> absolute jump address [0 ... $(2^{20}-1)$] as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The <code>set_list_jump</code> command is synonymous with <code>list_jump_pos</code> – see the comments there.
RTC4 → RTC5	Unchanged functionality (except for the extended range of values).
References	list_jump_pos



Delayed Short List Command	set_mark_speed
Function	Defines the marking speed for the subsequent vector commands and arc commands.
Call	<code>set_mark_speed(Speed)</code>
Parameter	Speed Marking speed in <i>bits per ms</i> (as a 64-bit IEEE floating point value). Allowed range: [1.6 ... 800000.0].
Comments	<ul style="list-style-type: none"> • By default a marking speed of 1000 <i>bits per ms</i> is preset. • The specified marking speed (or the preset marking speed, if no other value has been specified) is used for all subsequent mark and arc commands until a new value is specified. • To obtain the actual marking speed v in the image plane (in <i>meters per second</i>), the specified speed value (in <i>bits per ms</i>) must be divided by the calibration factor K of the correction table (in <i>bits per mm</i>): $v_{mark} = \text{Speed} / K$ <p>The calibration factor K can be queried from the correction table by get_table_para or get_head_para.</p> <ul style="list-style-type: none"> • The set_mark_speed command is also available as the control command set_mark_speed_ctrl.
RTC4→RTC5	Unchanged functionality (except for the extended range of values). The image field coordinates for the X and Y axes (but not the Z axis) and all related parameters (e.g. marking speed or wobble amplitude) are specified as 20-bit values; in RTC4 compatibility mode they are specified as 16-bit values (as with the RTC4) and the RTC5 multiplies the specified value for <code>Speed</code> by 16 (the allowed range of values is correspondingly reduced to [0.1 ... 50000.0]).
References	mark_abs , mark_rel , set_jump_speed , set_mark_speed_ctrl

Ctrl Command	set_mark_speed_ctrl
Function	Same as set_mark_speed , but a control command.
Call	<code>set_mark_speed_ctrl(Speed)</code>
Parameter	Speed marking speed in <i>bits per ms</i> (as a 64-bit IEEE floating point value) Allowed range: [1.6 ... 800000.0]
Comments	<ul style="list-style-type: none"> • The command is ignored (get_last_error return code: <code>RTC5_BUSY</code>) if the board's BUSY status is currently set (list is being processed or has been halted by pause_list). In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set) or the board's INTERNAL-BUSY status is currently set.
RTC4→RTC5	New command. In RTC4 compatibility mode: as set_mark_speed
References	set_mark_speed , set_jump_speed , set_jump_speed_ctrl



Ctrl Command	set_matrix
Function	Sets the coefficients of the general transformation matrix M_T for all subsequent coordinate transformations, see chapter 8.2 "Coordinate Transformations", page 183 .
Call	<code>set_matrix(HeadNo, M11, M12, M21, M22, at_once)</code>
Parameters	<p>HeadNo Number of the scan head connector as an unsigned 32-bit value: = 1: The definition only affects the <i>primary</i> scan head connector. = 2: The definition only affects the <i>secondary</i> scan head connector (activation required). = 0, 3: The definition affects <i>both</i> scan head connectors. = 4: The definition affects the virtual image field (see also comments). Only the three least significant bits are evaluated.</p> <p>M11, M12, M21, M22 Matrix coefficients of the general transformation matrix M_T as 64-bit IEEE floating point values. Allowed range: [-50 ... +50]; If the parameter is set to an invalid value, the command is ignored.</p> <p>at_once This parameter (unsigned 32-bit value) determines when the defined transformation becomes effective: = 0: The new total transformation (total matrix and offset) is only calculated and applied to the current position when the next list command is executed. = 1: The new total transformation is calculated immediately (or before the next list command if a list is currently BUSY or the board is INTERNAL- BUSY) and applied to the current position. The "laser active" laser control signals are first switched off if necessary. = 2: The new total transformation (total matrix and offset) is only calculated and applied to the current position when the next jump_abs, jump_rel, goto_xy or goto_xyz command is executed. = 3: As with at_once = 1, but the laser control signals remain unaffected. > 3: As with at_once = 2.</p>
Comments	<ul style="list-style-type: none"> See chapter 8.2 "Coordinate Transformations", page 183. The coordinate transformation defined by HeadNo = 4 is only in effect during a Processing-on-the-fly application initiated by set_fly_2d, see the section "Coordinate Transformations in the Virtual Image Field", page 208. Here, the at_once parameter is ignored.
RTC4 → RTC5	Unchanged primary functionality (matrix definition), however: <ul style="list-style-type: none"> The parameters HeadNo and at_once are new. Reduced value range for coefficients. See also "Notes for RTC4 Users", page 186.
Version info	Last changes: <ul style="list-style-type: none"> With version DLL 525, OUT 527: behavior for at_once = 3. With Version DLL 536, OUT 536: HeadNo = 4.
References	set_matrix_list , set_angle , set_offset , set_scale



Variable List Command	set_matrix_list
Function	Sets <i>one</i> of the four coefficients of the general transformation matrix M_T during execution of a list, see chapter 8.2 "Coordinate Transformations", page 183 .
Call	<code>set_matrix_list(HeadNo, Ind1, Ind2, Mij, at_once)</code>
Parameters	<p>HeadNo See set_matrix.</p> <p>Ind1, Ind2 Row index and column index of the matrix coefficient to be changed as an unsigned 32-bit value. Allowed values: [Index uneven: 1, Index even: 2].</p> <p>Mij Matrix coefficient as a 64-bit IEEE floating point value. Allowed range: [-50 ... +50]. If the parameter is set to an invalid value, the command is replaced by a list_nop.</p> <p>at_once This parameter (unsigned 32-bit value) determines when the defined transformation becomes effective: = 0: The transformation settings are only collected and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (e.g. by a list command with <code>at_once = 1</code> or a corresponding control command). = 1: The transformation is immediately calculated (including all transformation settings that were collected until then) and processed prior to the next list command. The "laser active" laser control signals are first switched off if necessary. = 2: The transformation settings are only collected and intermediately stored (as with <code>at_once = 0</code>). However, The transformation is immediately calculated (including all transformation settings that were collected and intermediately stored until then) and applied to the current position when the next jump_abs, jump_rel, goto_xy or goto_xyz command is executed. = 3: As with <code>at_once = 1</code>, but the laser control signals remain unaffected. > 3: See <code>at_once = 2</code>.</p>
Comments	<ul style="list-style-type: none"> The command <code>set_matrix_list</code> only allows changing <i>one</i> of the four coefficients at a time. To change several coefficients during execution of a list, the command has to be called repeatedly. Here, we recommend making the first calls with <code>at_once = 0</code> and only the last call with <code>at_once = 1</code>. See chapter 8.2 "Coordinate Transformations", page 183.
RTC4→ RTC5	See set_matrix .
Version info	Last change with version DLL 525, OUT 527: behavior for <code>at_once = 3</code> .
References	set_matrix



Ctrl Command	set_max_counts
Function	Defines the maximum number of external list starts.
Call	<code>set_max_counts(Counts)</code>
Parameter	Counts Maximum number of external list starts as an unsigned 32-bit value. Allowed range: [0 ... (2 ³² -1)].
Comments	<ul style="list-style-type: none"> When the specified number of external list starts has been reached, the external start input is disabled (see set_control_mode, bit#0 = 0). If Counts = 0, the number of external list starts is unlimited. When the RTC5 is initialized (by load_program_file), Counts is set to 0. The current number of successful external list starts can be read from the corresponding internal counter with the command get_counts. The counter can be reset by set_control_mode.
RTC4→RTC5	Unchanged functionality (except for the extended range of values).
References	get_counts , set_control_mode

Ctrl Command	set_mcbsp_freq
Function	Sets the transmission frequency of the McBSP/SPI interface.
Call	<code>mcbsp_freq = set_mcbsp_freq(Freq)</code>
Parameter	Freq Desired transmission frequency of the McBSP/SPI interface in Hz as an unsigned 32-bit value. Allowed range: [4000000 ... 16000000] (4 ... 16 MHz). Out-of-range values are edge-clipped. Out-of-range values cause 0 to be returned and the get_last_error return code RTC5_PARAM_ERROR to be generated.
Result	The actually set frequency in Hz as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The default transmission frequency (after initialization) is 8 MHz (Freq = 8,000,000). Because not every arbitrary frequency can be implemented, the command returns the actually set frequency. Example: <code>mcbsp_freq = set_mcbsp_freq(7,000,000)</code>; returns <code>mcbsp_freq = 7,200,000</code>. The new transmission frequency only becomes effective when you re-initialize the McBSP/SPI interface by the command mcbsp_init. The signals and operating conditions of the McBSP/SPI interface are presented in the chapter 4.4.6 "SPI / I2C Socket Connector", page 54. Note: The receiving frequency is exclusively determined by the incoming clock pulses and has a maximum limit of 16 MHz.
RTC4→RTC5	New command.
Version info	Available as of version DLL 531, OUT 532.
References	mcbsp_init , set_mcbsp_out , set_mcbsp_out_ptr

Ctrl Command	set_mcbsp_in
Function	Activates Processing-on-the-fly correction for compensation of a workpiece or scan system movement (based on position values transferred to the RTC5 by the McBSP/SPI interface). The McBSP/SPI interface can also be used for inputting other desired signals.
Restriction	If the Processing-on-the-fly option is not enabled, then the command terminates the Processing-on-the-fly process (even though it could not have been activated).
Call	<code>set_mcbsp_in(Mode, Scale)</code>
Parameters	<p>Mode an unsigned 32-bit value, allowed values:</p> <ul style="list-style-type: none"> = 0: Processing-on-the-fly correction is switched off. = 1: Compensation of linear movement in the X direction. = 2: Compensation of linear movement in the Y direction. = 3: Compensation of linear movement in the X and Y directions. = 4: Compensation of rotary movement. = 5: Processing-on-the-fly correction is switched off. <ul style="list-style-type: none"> • Mode = 0 ... 5: All McBSP/SPI input values are alternately copied to internal memory locations 1 and 2. • Mode = 1 ... 5 (but not for Mode = 0): McBSP/SPI input values coded with "Bit#31 = 0" is additionally copied to internal memory location 0 and those with "Bit#31 = 1" to internal memory location 3. • Mode = 1 ... 4: Values copied to internal memory location 0 are applied for Processing-on-the-fly correction. <p>Scale Scaling factor or rotation resolution as a 64-bit IEEE floating point value:</p> <ul style="list-style-type: none"> • Mode = 1 ... 3: scaling factor in $(RTC5)bits/(McBSP)bit$. Allowed range: $1/256 \leq Scale \leq 16000.0$ (if Mode = 3, Scale applies to both axes). • Mode = 4: number of steps (counts) per revolution. Allowed range: $Scale > 100.0$.
Comments	<ul style="list-style-type: none"> • You can query the internal memory locations at any time by read_mcbsp. • For Processing-on-the-fly correction and determination of the scaling factor, see chapter 8.7 "Processing-on-the-fly (Optional)", page 199. • The various Processing-on-the-fly corrections cannot be arbitrarily combined (see the page 200). • For deactivating Processing-on-the-fly correction, see page 209. • 15 bits per axis (with sign) are effectively available for two-dimensional correction (Mode = 3), whereby the X value is in the lower 16 bits of the "bit#31 = 0"-coded McBSP/SPI input value and the Y value in the upper 16 bits. • The McBSP/SPI interface cannot be simultaneously used for both Processing-on-the-fly applications and online positioning (see page 188). • The McBSP/SPI interface always ignores the first FrameSync signal after a load_program_file or mcbsp_init, so available data is not transmitted (see page 56). • If Mode > 5, <code>set_mcbsp_in</code> is not executed (<code>get_last_error</code> return code: <code>RTC5_PARAM_ERROR</code>). • If an unallowed Scale parameter value is supplied (e.g. Scale = 0), <code>set_mcbsp_in</code> behaves as with Mode = 0.



Ctrl Command	set_mcbsp_in
RTC4→ RTC5	New command.
Version info	Available as of version DLL 531, OUT 532.
References	set_mcbsp_in_list

Undelayed Short List Command	set_mcbsp_in_list				
Function	Identical with set_mcbsp_in , but a list command.				
Restriction	If the Processing-on-the-fly option is not enabled, then the command terminates the Processing-on-the-fly process (even though it could not have been activated).				
Call	<code>set_mcbsp_in_list(Mode, Scale)</code>				
Parameters	<table> <tr> <td>Mode</td> <td>See set_mcbsp_in.</td> </tr> <tr> <td>Scale</td> <td>See set_mcbsp_in.</td> </tr> </table>	Mode	See set_mcbsp_in .	Scale	See set_mcbsp_in .
Mode	See set_mcbsp_in .				
Scale	See set_mcbsp_in .				
Comments	<ul style="list-style-type: none"> • If Mode > 5, then set_mcbsp_in_list is replaced by a list_nop (get_last_error return code RTC5_PARAM_ERROR). • See set_mcbsp_in. 				
RTC4→ RTC5	New command.				
Version info	Available as of version DLL 531, OUT 532.				
References	set_mcbsp_in				



Ctrl Command	set_mcbsp_matrix
Function	Activates matrix correction for online positioning by the McBSP/SPI interface.
Call	<code>set_mcbsp_matrix()</code>
Comments	<ul style="list-style-type: none"> For online positioning, see chapter 8.3 "Online Positioning", page 187. Matrix corrections cannot be used in conjunction with offset and/or rotation corrections. Any such already-activated options gets deactivated by set_mcbsp_matrix. Subsequent activation of other options (by set_mcbsp_x, set_mcbsp_y or set_mcbsp_rot) deactivates the matrix option. The following restrictions apply to the matrix coefficients transferred over the McBSP/SPI interface (as with set_matrix): <p>The allowed value range for matrix coefficients is [-50 ... +50]. Transferred coefficients exceeding this range are ignored.</p> You must individually supply as input value M_{in} to the McBSP/SPI interface each matrix coefficient M_{ij} of the transformation matrix M_T as a normalized integer with associated indices i and j as follows: $M_{in} = (\text{integer}(M_{ij} * 2^{24}) << 2) + (i << 1) + j$ <p>with $M_T = \{ M_{00}, M_{01}, M_{10}, M_{11} \} = \{ m_{11}, m_{12}, m_{21}, m_{22} \}$.</p> <p>Conversely, the RTC5 determines a coefficient from the input value as follows:</p> $M_T[M_{in} \& 0x3] = (M_{in} >> 2) / 2^{24}.$ You must separately fetch each transferred matrix coefficient by apply_mcbsp or apply_mcbsp_list. We recommend fetching the first coefficient with <code>at_once = 0</code> and only the last one with <code>at_once > 0</code>. The coefficients get transferred to internal memory location 1 and can be checked there by querying with <code>read_mcbsp(1)</code>. The McBSP/SPI interface cannot be simultaneously used for both online positioning and Processing-on-the-fly applications (see page 188).
RTC4→RTC5	New command.
Version info	Available as of version DLL 533, OUT 534, RBF 524.
References	set_mcbsp_matrix_list



Undelayed Short List Command	set_mcbsp_matrix_list
Function	Same as set_mcbsp_matrix , but a list command.
Call	<code>set_mcbsp_matrix_list()</code>
Comments	<ul style="list-style-type: none"> • See set_mcbsp_matrix.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 533, OUT 534, RBF 524.
References	set_mcbsp_matrix

Undelayed Short List Command	set_mcbsp_out
Function	Defines two signal types for output at the McBSP/SPI interface (Multi channel Buffered Serial Port, see also chapter 4.4.6 "SPI / I2C Socket Connector", page 54).
Call	<code>set_mcbsp_out(Signal1, Signal2)</code>
Parameter	Signal1, Desired signal type as unsigned 32-bit values. Signal2
Comments	<ul style="list-style-type: none"> • The selectable signal types are identical to those of the set_trigger command (refer to the comments there for the allowed value range, signal types and other information). If the value for Signal1/Signal2 is unallowed, then set_mcbsp_out is replaced by list_nop (get_last_error return code RTC5_PARAM_ERROR). • Both selected data signals are continuously transmitted (once per 10 µs cycle). • Only 16-bit portions of the selected data signals are packed into a common 32-bit data word for output. Signal1 is the lower half and Signal2 the upper half of this 32-bit data word. <ul style="list-style-type: none"> – Only RTC4-compatible bits #4-19 of the sample values and status values returned by the scan system (Signal1, Signal2 = 1-23, 25-30) are outputted. The least significant 4 bits and any exceeding bits (in the virtual image field) are ignored. – For the other data types (Signal1, Signal2 = 0, 24, 31-42), the least significant 16 bits are outputted and the upper bits are ignored (only relevant for Signal1, Signal2 = 37-42 and possibly 24 and 31). • Transmitted values are those of the preceding clock cycle: data is processed at the end of a cycle and transmitted at the beginning of the next clock cycle at the set transmission frequency (see set_mcbsp_freq). • The signals and operating conditions of the chapter 4.4.6 "SPI / I2C Socket Connector", page 54 interface are presented in chapter 4.4.6 "SPI / I2C Socket Connector", page 54.
RTC4→ RTC5	New command.
Version info	Last change with version DLL 531, OUT 532: <ul style="list-style-type: none"> • Further selectable data signals (Signal1, Signal2 = 25...42, see set_trigger). • For some of the new data signals, only the lower 16 bits are outputted (see above).
References	read_mcbsp , mcbsp_init , set_mcbsp_freq , set_mcbsp_out_ptr , set_trigger



Ctrl Command	set_mcbsp_out_ptr
Function	Defines a list of up to 8 signal types for output at the McBSP/SPI interface (Multi channel Buffered Serial Port, see also page 54).
Call	set_mcbsp_out_ptr(Number, SignalPtr)
Parameters	<p>Number Unsigned 32-bit value. Allowed range: [0 ... 8]. = 1...8: Number of signal types to be outputted. = 0 Output at the McBSP/SPI interface occurs in accordance with the pre-defined settings or as specified by a prior set_mcbsp_out command.</p> <p>SignalPtr Pointer (in C and C++ data type ULONG_PTR, i.e. an unsigned 32-bit or 64-bit value) to an array of Number unsigned 32-bit values, where the desired Number signal type numbers are specified.</p>
Comments	<ul style="list-style-type: none"> The memory area for the SignalPtr array must be provided by the user program. If Number > 8 and/or SignalPtr = NULL, set_mcbsp_out_ptr is not executed (get_last_error return code: RTC5_PARAM_ERROR). The selectable signal types are identical to those of the set_trigger command (refer to the comments there for the allowed value range, signal types and other information). The up to 8 selected data types are outputted sequentially (one data type per 10 µs clock cycle). Each individual signal belongs to a different clock cycle. Transmitted values are always from the previous clock cycle. When outputting, each signal value is supplemented by the corresponding signal type number: the signal type number gets inserted into the lowest byte of the 32-bit data word. The actual signal value therefore gets shifted 8 bits to the left, whereby all bits above the 24th get truncated (overflow, no clipping, relevant only for signal types 37-42 and possibly 24 and 31): 32-bit output value = (signal value << 8) (signal type number & 0xFF). The signals and operating conditions of the McBSP/SPI interface are presented in the section "Use as McBSP Interface", page 54.
RTC4→RTC5	New command.
Version info	Available as of version DLL 531, OUT 532.
References	set_mcbsp_out , set_trigger



Ctrl Command	set_mcbsp_rot
Function	Activates or deactivates rotation correction for online positioning by the McBSP/SPI interface.
Call	set_mcbsp_rot(Resolution)
Parameter	Resolution Value as a 64-bit IEEE floating point number: $2^{-26} < \text{Resolution} < 2^{26}$: scaling factor (correction is activated), otherwise: correction is deactivated. Resolution = McBSP/SPI bits per full circle.
Comments	<ul style="list-style-type: none"> For online positioning, see page 187. For an McBSP/SPI rotation correction input value of Rot_{in}, the command apply_mcbsp functions like set_angle(... , Angle \times 360° , ...), whereby (FC = full circle) Angle (in FC) = $\text{Rot}_{in} / \text{Resolution}$ Only Angle values in the range [0.0...+20.0 FC] are allowed. The McBSP/SPI interface cannot be simultaneously used for both online positioning and Processing-on-the-fly applications (see page 188).
RTC4→ RTC5	New command.
Version info	Available as of version DLL 524, OUT 526.
References	set_mcbsp_rot_list , set_mcbsp_x , set_mcbsp_y , apply_mcbsp

Undelayed Short List Command	set_mcbsp_rot_list
Function	Same as set_mcbsp_rot , but a list command.
Call	set_mcbsp_rot_list(Resolution)
Parameter	Resolution See set_mcbsp_rot .
Comments	<ul style="list-style-type: none"> See set_mcbsp_rot.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 524, OUT 526.
References	set_mcbsp_rot



Ctrl Command	set_mcbsp_x
Function	Activates or deactivates X offset correction for online positioning by the McBSP/SPI interface.
Call	set_mcbsp_x(Scale)
Parameter	Scale Value as a 64-bit IEEE floating point number: $2^{-26} < \text{Scale} < 2^{26}$: scaling factor (correction is activated), otherwise: correction is deactivated.
Comments	<ul style="list-style-type: none"> For online positioning, see page 187. With an McBSP/SPI input value of X_{in} for the X offset correction, the command apply_mcbsp functions like <code>set_offset(..., XOffset, ...)</code>, whereby $XOffset$ (in bits) = Scale $\times X_{in}$ $XOffset$ values outside of $[-524288 \dots +524287]$ are clipped to the boundaries as long as Scale $\times X_{in}$ does not exceed the value range $\pm 2^{31}$. The McBSP/SPI interface cannot be simultaneously used for both online positioning and Processing-on-the-fly applications (see page 188).
RTC4→ RTC5	New command.
Version info	Available as of version DLL 524, OUT 526.
References	set_mcbsp_x_list , set_mcbsp_y , set_mcbsp_rot , apply_mcbsp

Undelayed Short List Command	set_mcbsp_x_list
Function	Same as set_mcbsp_x , but a list command.
Call	set_mcbsp_x_list(Scale)
Parameter	Scale See set_mcbsp_x .
Comments	<ul style="list-style-type: none"> See set_mcbsp_x.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 524, OUT 526.
References	set_mcbsp_x



Ctrl Command	set_mcbsp_y
Function	Activates or deactivates Y offset correction for online positioning by the McBSP/SPI interface.
Call	set_mcbsp_y(Scale)
Parameter	Scale Value as a 64-bit IEEE floating point number: $2^{-26} < \text{Scale} < 2^{26}$: scaling factor (correction is activated), otherwise: correction is deactivated.
Comments	<ul style="list-style-type: none"> For online positioning, see chapter 8.3 "Online Positioning", page 187. With an McBSP/SPI input value of Y_{in} for the Y offset correction, the command apply_mcbsp functions like set_offset(..., YOffset, ...), whereby Y_{Offset} (in bits) = Scale $\times Y_{in}$ Y_{Offset} values outside of [-524288 ... +524287] are clipped to the boundaries as long as Scale $\times Y_{in}$ does not exceed the value range $\pm 2^{31}$. The McBSP/SPI interface cannot be simultaneously used for both online positioning and Processing-on-the-fly applications (see notes page 188).
RTC4→ RTC5	New command.
Version info	Available as of version DLL 524, OUT 526.
References	set_mcbsp_y_list , set_mcbsp_x , set_mcbsp_rot , apply_mcbsp

Undelayed Short List Command	set_mcbsp_y_list
Function	Same as set_mcbsp_y , but a list command.
Call	set_mcbsp_y_list(Scale)
Parameter	Scale See set_mcbsp_y .
Comments	<ul style="list-style-type: none"> See set_mcbsp_y.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 524, OUT 526.
References	set_mcbsp_y

Ctrl Command	set_multi_mcbsp_in
Function	Activates a Processing-on-the-fly application and McBSP/SPI interface multi transmission with up to eight different data types.
Restriction	If the Processing-on-the-fly option is not enabled, then the command switches off the Processing-on-the-fly process (even if it was never switched on).
Call	<code>set_multi_mcbsp_in(Ctrl, P, Mode)</code>
Parameter	<p>Ctrl control parameter for initializing or deactivating laser power variation as an unsigned 32-bit value: = 1...6: defines which signal parameter to vary: for a description, see set_auto_laser_control. = 0 or > 6: deactivates laser power variation (for Ctrl > 6: get_last_error return code RTC5_PARAM_ERROR).</p> <p>P Initialization value for laser power as an unsigned 32-bit value. Allowed range: see set_auto_laser_control.</p> <p>Mode Unsigned 32-bit value. = 0: The transmitted value is used directly. = 1: The transmitted value is multiplied by P/16384 and then put out.</p>
	Ctrl, P and Mode are only relevant for Type 3 (= laser power) of the transmitted data word, see read_multi_mcbsp .
Comments	<ul style="list-style-type: none"> Any other previously activated McBSP/SPI transmission for online positioning and any other previously activated Processing-on-the-fly application with encoder signals or positional values is terminated first. The command enables inputting by the McBSP/SPI interface of up to eight different types for asynchronous transmission every 10 µs. Each 10 µs, the data is copied in accordance with its type code to separate memory, from where it can also be queried by read_multi_mcbsp. To avoid faulty sorting, no active McBSP/SPI transmission should be occurring when you issue the command. The McBSP/SPI interface always ignores the first FrameSync signal after a load_program_file or mcbsp_init, so available data is not transmitted (see page page 56). You must code the transmission's type assignment into the data word's least significant three bits in accordance with McBSPValue = (Value << 3) Type. The RTC5 stores the data word in accordance with $\text{Memory}[\text{McBSPValue} \& 0x7] = (\text{long}) \text{McBSPValue} >> 3.$ For more on type assignments, see read_multi_mcbsp. The remaining (most significant) 29 bits are available for the data word itself. There are no further restrictions other than the type-dependent value ranges themselves. The transmitted values are not checked with respect to their ranges. Clipping or data overflow may occur. The four extra parameters are not the same as the (previously four, now eight) free variables (see chapter 6.9.1 "Free Variables", page 102, although they can be used for similar purposes. Upon program start, they are initialized with 0 and this command does not further modify them. The transmitted values merely get copied into type-sorted memory.



Ctrl Command	set_multi_mcbsp_in
Comments (cont'd)	<ul style="list-style-type: none"> If more than four McBSP/SPI transfers complete within a $10 \mu\text{s}$ clock cycle, then prior values might get overwritten. If the Processing-on-the-fly option is enabled, then the command activates a Processing-on-the-fly application with positional values for the three coordinate directions X, Y and Z. The memory values of their respective types are initialized with 0. Z, too, has 20-bit resolution. Additionally, laser power can be varied by outputting the transmitted type 3 value at the port assigned by the <code>Ctrl</code> parameter. The initialization value <code>P</code> gets put out immediately. For <code>Mode = 0</code>, subsequent type-3 transfers are outputted directly at the port assigned by <code>Ctrl</code>. For <code>Mode = 1</code>, the transferred value is handled as a multiplication factor in accordance with Normalization $1.0 = 16384$ (14 bits). Thus, the following is put out: $(P \times \text{the transmitted value} / 16384)$. This mode is an alternative to laser power variation by "freely definable wobble shapes", but without the Softstart suppression. These laser power variation method can be combined with the vector-defined laser control (with parameterized commands). It cannot be combined with other automatic laser control methods. It overwrites other variations sharing the same <code>Ctrl</code> parameter. Though unidentical <code>Ctrl</code> parameters are allowed, they serve no practical purpose. If <code>Ctrl = 0</code>, then laser power variation is switched off. The values transmitted by McBSP/SPI continue to be copied into type-sorted memory, but are not put out. Special case: if a "freely definable wobble shape" is active, then <code>P</code> is always regarded (irrespective of the <code>Mode</code> parameter) as relative laser power and multiplicatively coupled with the wobble-shape's laser power (see set_wobble_vector). Because this wobble shape defines its own laser power (see set_wobble_control), the command's <code>Ctrl</code> parameter has no relevance. It should be set to an invalid value (e.g. with <code>Ctrl = 0</code>) to avoid doubled or meaningless output.
RTC4 → RTC5	New command.
Version info	Available as of version DLL 537, OUT 537.
Comments	set_multi_mcbsp_in_list , read_multi_mcbsp , set_wobble_control , set_wobble_vector



Normal List Command	set_multi_mcbsp_in_list
Function	Same as set_multi_mcbsp_in , but a list command.
Restriction	If the Processing-on-the-fly option is not enabled, then the command switches off the Processing-on-the-fly process (even if it was never switched on).
Call	set_multi_mcbsp_in_list(Ctrl, P, Mode)
Parameter	Ctrl, P See set_multi_mcbsp_in .
Comments	• See set_multi_mcbsp_in .
RTC4→ RTC5	New command.
Version info	Available as of version DLL 537, OUT 537.
References	set_multi_mcbsp_in , read_multi_mcbsp , set_wobbel_control , set_wobbel_vector



Variable List Command	set_n_pixel						
Function	In pixel output mode, this command defines the laser control parameters (pulse length and analog voltage level) for multiple directly successive identical pixels in an image line.						
Call	<code>set_n_pixel(PulseLength, AnalogOut, Number)</code>						
Parameters	<table> <tr> <td>PulseLength</td> <td>Pixel pulse length (as an unsigned 32-bit value). <i>1 bit equals 1/64 µs.</i> Allowed range: [0 ... (2³²-1)]</td> </tr> <tr> <td>AnalogOut</td> <td>12-bit output value (analog voltage level at the analog output port selected with <code>set_pixel_line</code>) for the pixel as an unsigned 32-bit value. Higher bits are ignored (see also <code>write_da_x</code>).</td> </tr> <tr> <td>Number</td> <td>number of pixels (as an unsigned 32-bit value) Allowed range: [1 ... (2³²-1)]. 0 is automatically clipped to 1.</td> </tr> </table>	PulseLength	Pixel pulse length (as an unsigned 32-bit value). <i>1 bit equals 1/64 µs.</i> Allowed range: [0 ... (2 ³² -1)]	AnalogOut	12-bit output value (analog voltage level at the analog output port selected with <code>set_pixel_line</code>) for the pixel as an unsigned 32-bit value. Higher bits are ignored (see also <code>write_da_x</code>).	Number	number of pixels (as an unsigned 32-bit value) Allowed range: [1 ... (2 ³² -1)]. 0 is automatically clipped to 1.
PulseLength	Pixel pulse length (as an unsigned 32-bit value). <i>1 bit equals 1/64 µs.</i> Allowed range: [0 ... (2 ³² -1)]						
AnalogOut	12-bit output value (analog voltage level at the analog output port selected with <code>set_pixel_line</code>) for the pixel as an unsigned 32-bit value. Higher bits are ignored (see also <code>write_da_x</code>).						
Number	number of pixels (as an unsigned 32-bit value) Allowed range: [1 ... (2 ³² -1)]. 0 is automatically clipped to 1.						
Comments	<ul style="list-style-type: none"> For usage of this command, see chapter 8.8 "Pixel Output Mode – Scanning Raster Images (Bitmaps)", page 217. Before the first <code>set_n_pixel</code> command of a line, a <code>set_pixel_line</code> command must be issued. The <code>set_n_pixel</code> command defines parameters for the subsequent <code>Number</code> of (identical) pixels in an image line. Each <code>set_n_pixel</code> command must follow immediately after the command <code>set_pixel_line</code> or after another <code>set_n_pixel</code> command. No other commands must be written into the list until the image line is completed. The first command in the list following a <code>set_pixel_line</code> command which is <i>not</i> a <code>set_n_pixel</code> command turns off the pixel output mode. If only an individual pixel needs to be defined, then <code>set_pixel</code> can be used as an alternative to <code>set_n_pixel</code>. This command is synonymous with <code>set_n_pixel</code> with parameter <code>Number = 1</code>. If not in pixel output mode (if the command is not directly preceded by <code>set_pixel_line</code> or <code>set_n_pixel</code>), <code>set_n_pixel</code> is a short list command and otherwise ignored. Under some circumstances, a <code>list_nop</code> might be inserted (see page 250). At the end of the image line (at the beginning of the default pixel), the pixel pulse length <code>PulseLength</code> and the analog output value <code>AnalogOut</code> are set to a default value, each. The default pixel pulse length value is 0 unless another value was defined by <code>set_default_pixel</code> or <code>set_default_pixel_list</code>. The default analog output value is 0xFF (=4095) unless another value was defined by <code>set_port_default</code> or <code>set_laser_off_default</code>. 						
RTC4→ RTC5	New command. For RTC4 compatibility mode: see <code>set_pixel</code> .						
References	set_pixel , set_pixel_line , set_pixel_line_3d						



Ctrl Command	set_offset
Function	Defines an offset for all subsequent coordinate transformations (see "Coordinate Transformations", page 183).
Call	set_offset(HeadNo, XOffset, YOffset, at_once)
Parameters	HeadNo See set_offset_xyz .
	XOffset, See set_offset_xyz .
	YOffset
Comments	at_once See set_offset_xyz .
	<ul style="list-style-type: none"> See chapter 8.2 "Coordinate Transformations", page 183. The command functions similarly to set_offset_xyz, but leaves ZOffset unchanged.
	<p>RTC4→ RTC5 Unchanged primary functionality (offset definition), however:</p> <ul style="list-style-type: none"> The parameters HeadNo and at_once are new. Extended offset value range. See also "Notes for RTC4 Users", page 186. <p>The image field coordinates for the X and Y axes are specified as 20-bit values; in RTC4 compatibility mode they are specified as 16-bit values (as with the RTC4) and the RTC5 multiplies the specified offset values by 16 (the allowed range of values is correspondingly reduced to [-32768 ... 32767]).</p>
Version info	Last changes: <ul style="list-style-type: none"> With version DLL 525, OUT 527: behavior for at_once = 3. With Version DLL 536, OUT 536: HeadNo = 4.
References	set_offset_list , set_offset_xyz , set_angle , set_matrix , set_scale

Variable List Command	set_offset_list
Function	Same as set_offset_xyz , but a list command.
Call	set_offset_list(HeadNo, XOffset, YOffset, at_once)
Parameters	HeadNo See set_offset_xyz .
	XOffset, See set_offset_xyz .
	YOffset
RTC4→ RTC5	at_once See set_offset_xyz_list .
	See set_offset_xyz .
	Last change with version DLL 525, OUT 527: behavior for at_once = 3.
References	set_offset_xyz



Ctrl Command	set_offset_xyz
Function	Defines an offset for all subsequent coordinate transformations, see chapter 8.2 "Coordinate Transformations", page 183.
Call	<code>set_offset_xyz(HeadNo, XOffset, YOffset, ZOffset, at_once)</code>
Parameters	<p>HeadNo Number of the scan head connector as an unsigned 32-bit value: = 1: The definition only affects the <i>primary</i> scan head connector. = 2: The definition only affects the <i>secondary</i> scan head connector (activation required). = 0, 3: The definition affects <i>both</i> scan head connectors. = 4: The definition affects the virtual image field (see also comments). Only the three least significant bits are evaluated.</p> <p>XOffset, YOffset Offsets for the X and Y directions (coordinate translation relative to the origin of the cartesian coordinate system) in <i>bits</i> (as a signed 32-bit value). Allowed range: [-524288 ... +524287]. Out-of-range values are edge-clipped.</p> <p>ZOffset Offset for the Z directions (coordinate translation relative to the origin of the cartesian coordinate system) in <i>bits</i> (as a signed 32-bit value). Allowed range: [-32768 ... +32767]. Out-of-range values are edge-clipped.</p> <p>at_once This parameter (unsigned 32-bit value) determines when the defined transformation becomes effective: = 0: The new total transformation (total matrix and offset) is only calculated and applied to the current position when the next list command is executed. = 1: The new total transformation is calculated immediately (or before the next list command if a list is currently BUSY or the board is INTERNAL-BUSY) and applied to the current position. The "laser active" laser control signals are first switched off if necessary. = 2: The new total transformation (total matrix and offset) is only calculated and applied to the current position when the next jump_abs, jump_rel, goto_xy or goto_xyz command is executed. = 3: As with at_once = 1, but the laser control signals remain unaffected. > 3: See at_once = 2.</p>
Comments	<ul style="list-style-type: none"> See chapter 8.2 "Coordinate Transformations", page 183. <code>zOffset</code> is stored only once, applying jointly for both scan head connectors (<code>HeadNo</code> is therefore ignored). <code>zOffset <> 0</code> shifts the working plane (in the direction opposite to the laser beam); for a hypothetical control value of <code>(0 0 0)</code>, this would be by $d = \text{Shift} / K_z$ to the plane $z = d$ in the direction of the Z coordinate axis (and in the direction opposite to the focus shift specified by <code>set_defocus</code>). Here, K_z is the calibration factor for the Z direction (see also chapter 8.6.3 "3D Marking Commands", page 194). The coordinate transformation defined by <code>HeadNo = 4</code> is only in effect during a Processing-on-the-fly application initiated by <code>set_fly_2d</code> (see the section "Coordinate Transformations in the Virtual Image Field", page 208). Here, the <code>at_once</code> parameter is ignored. If <code>HeadNo = 4</code>, then <code>zOffset</code> is ignored.



Ctrl Command	set_offset_xyz
RTC4→ RTC5	<p>New command.</p> <p>With the RTC5, the image field coordinates for the X and Y axes are specified as 20-bit values. In RTC4 compatibility mode, they are specified as 16-bit values (as with the RTC4) and the RTC5 multiplies the specified offset values for the X and Y axes (not for the Z axis) by 16 (the allowed range of values is correspondingly reduced to [-32768 ... +32767]).</p>
Version info	<p>Last changes:</p> <ul style="list-style-type: none"> • With version DLL 525, OUT 527: behavior for <code>at_once = 3</code>. • With Version DLL 536, OUT 536: HeadNo = 4.
References	set_offset_xyz_list , set_offset , set_angle , set_matrix , set_scale , set_defocus

Variable List Command	set_offset_xyz_list
Function	Same as set_offset_xyz , but a list command.
Call	<code>set_offset_xyz_list(HeadNo, XOffset, YOffset, ZOffset, at_once)</code>
Parameters	<p>HeadNo See set_offset_xyz.</p> <p>XOffset, See set_offset_xyz.</p> <p>YOffset</p> <p>at_once This parameter (unsigned 32-bit value) determines when the defined transformation becomes effective:</p> <ul style="list-style-type: none"> = 0: The transformation settings are only collected and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (e.g. by a list command with <code>at_once = 1</code> or a corresponding control command). = 1: The transformation is immediately calculated (including all transformation settings that were collected until then) and processed prior to the next list command. The “laser active” laser control signals are first switched off if necessary. = 2: The transformation settings are only collected and intermediately stored (as with <code>at_once = 0</code>). However, The transformation is immediately calculated (including all transformation settings that were collected and intermediately stored until then) and applied to the current position when the next jump_abs, jump_rel, goto_xy or goto_xyz command is executed. = 3: As with <code>at_once = 1</code>, but the laser control signals remain unaffected. > 3: See <code>at_once = 2</code>
RTC4→ RTC5	See set_offset_xyz .
Version info	Last change with version DLL 525, OUT 527: behavior for <code>at_once = 3</code> .
References	set_offset_xyz , set_offset_list , set_defocus_list



Ctrl Command	set_pause_list_cond
Function	Defines the condition at the 16-bit digital input port of the EXTENSION 1 socket connector under which a pause_list automatically is executed.
Call	<code>set_pause_list_cond(Mask1, Mask0)</code>
Parameter	<p>Mask1 16-bit mask. As unsigned 32-bit value. Only the least 16 bits are evaluated.</p> <p>Mask0 As Mask1.</p>
Comments	<ul style="list-style-type: none"> If a list is currently executed and the following condition is met (identical to other conditional commands, see also chapter 9.3.2 "Conditional Command Execution", page 244): $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } ((\text{not IOvalue) AND Mask0}) = \text{Mask0})$ (that is, if the bits in IOValue specified in Mask1 are 1 and the bits specified in Mask0 are 0), then automatically a pause_list is executed. The condition is checked once per 10 μs clock cycle. The paused list can only be resumed by restart_list. Mask1 = Mask0 = 0 disables the condition. If the condition is disabled or no list is currently executed, nothing else happens. With set_pause_list_cond in the case of an error the currently executed list can be paused immediately by a hardware circuit. This avoids using a time critical call of a command via the operating system. A simultaneously present /STOP signal takes precedence over the conditional pause_list.
RTC4 → RTC5	New command.
Version info	Available as of version DLL 543, OUT 543.
References	pause_list , restart_list



Variable List Command	set_pixel
Function	In the pixel output mode, defines the laser control parameters (pulse length and analog voltage level) for one pixel in an image line.
Call	<code>set_pixel(PulseLength, AnalogOut)</code>
Parameters	<p>PulseLength Pixel pulse length (as an unsigned 32-bit value). <i>1 bit equals 1/64 µs.</i> Allowed range: [0 ... ($2^{32}-1$)].</p> <p>AnalogOut 12-bit output value (analog voltage level at the analog output port selected with set_pixel_line) for the pixel as an unsigned 32-bit value. Higher bits are ignored (see also write_da_x).</p>
Comments	<ul style="list-style-type: none"> The command is synonymous with set_n_pixel with Number = 1 (see comments there).
RTC4→ RTC5	<ul style="list-style-type: none"> The RTC5 does <i>not</i> support querying of analog voltage levels (see the RTC4's ADChannel parameter and read_pixel_ad command). The RTC5 controls the pixel pulses (at the LASER1 port) directly by the PulseLength parameter, whereas the RTC4 controls them by the LASERON signal. PulseLength must be specified in units of 1/64 µs with the RTC5; in RTC4 compatibility mode it is specified in units of 1/8 µs (as with the RTC4) and the RTC5 multiplies the specified value by 8. In RTC4 compatibility mode, the RTC5 also multiplies AnalogOut by 4 (the RTC4's analog outputs only handled 10-bit values). The allowed range of values is correspondingly reduced.
References	set_n_pixel , set_pixel_line , set_pixel_line_3d



Normal List Command	set_pixel_line
Function	Activates the pixel output mode and defines various pixel output parameters.
Call	<code>set_pixel_line(Channel, HalfPeriod, dX, dY)</code>
Parameters	<p>Channel Number of the analog output port (as an unsigned 32-bit value) that should output the analog voltage levels defined by subsequent set_pixel/set_n_pixel commands. Allowed range: [1, 2] (1: ANALOG OUT1, 2: ANALOG OUT2).</p> <p>HalfPeriod <i>Half</i> pixel output period in <i>bits</i> as an unsigned 32-bit value. <i>1 bit equals 1/64 µs.</i> Allowed range: [104 ... (2³²-1)].</p> <p>dX, dY Distance in the X and Y directions between adjacent pixels in <i>bits</i> (64-bit IEEE floating point values).</p>
Comments	<ul style="list-style-type: none"> Each image line must start with the command set_pixel_line. This command should be preceded by a jump or mark command to the start point of the image line. The set_pixel_line command is directly followed by the desired number of set_pixel/set_n_pixel commands, which transmit the <code>PulseLength</code> and <code>AnalogOut</code> parameters. The first list command after set_pixel_line that is <i>not</i> a set_pixel/set_n_pixel command turns off the pixel output mode (the set_pixel_line command, too, ends the pixel output mode before starting it again). In the process, a default pixel is inserted. If <code>Channel < 1</code> or <code>Channel > 2</code>, then set_pixel_line is replaced by a list_nop (get_last_error return code <code>RTC5_PARAM_ERROR</code>) and the pixel output mode is <i>not</i> activated. Note that <i>half</i> the period length must be specified for <code>HalfPeriod</code>. $2 \times \text{HalfPeriod}$ is thus the chronological distance between individual pixels (see page 217). <code>HalfPeriod</code> must not be smaller than 104 (otherwise it is clipped). This value corresponds to a pixel frequency of approx. 308 kHz. For pixel frequencies above around 100 kHz (i.e. for a <code>HalfPeriod < approx. 320</code>) digital-to-analog conversion cannot always be fully completed. For such pixel frequencies, users must carefully verify that sufficient capability is available. The pixel output mode is incompatible with the softstart mode (see page 152). The pixel output mode should <i>not</i> be used in conjunction with automatic laser control (see page 159) if automatic laser control readjusts the 12-bit output values at the ANALOG OUT1 or ANALOG OUT2 output ports or pulse lengths (<code>PulseLength</code>) or output periods (<code>HalfPeriod</code>) of the laser signals LASER1 and LASER2. The pixel output mode can be combined with Processing-on-the-fly (see page 199). The pixel output mode <i>cannot</i> be combined with Sky Writing (see page 127) or Wobbel (see page 189). See also chapter 8.8 "Pixel Output Mode – Scanning Raster Images (Bitmaps)", page 217.



Normal List Command	set_pixel_line
RTC4→ RTC5	<ul style="list-style-type: none"> On the RTC5, only one pixel mode is available (equivalent to the RTC4's PixelMode = 1). The RTC5 allows selection of the output channel (Channel) for analog signals (in pixel output mode, the RTC4 can only output analog voltage levels at its ANALOG OUT2 port). For the RTC5, the pixel output period is specified as the <i>half</i> output period HalfPeriod (in contrast, for the RTC4 it is the <i>full</i> output period PixelPeriod). For the RTC5, the command set_pixel_line requires only one list entry, as with other normal list commands (whereas the RTC4 requires two list entries). In RTC4 compatibility mode, HalfPeriod must be specified in units of 1/8 µs. The parameter values are then internally converted to 1/64 µs units (i.e. multiplied by 8). The allowed range of values is correspondingly smaller (the smallest allowed value is then 13). In RTC4 compatibility mode, the RTC5 also multiplies the specified coordinate values dx, dy by 16 (the allowed range of values is correspondingly reduced).
Version info	Last change with version OUT 515.
References	set_pixel , set_n_pixel , set_pixel_line_3d

List Multi-Command	set_pixel_line_3d								
Function	Activates the pixel output mode and defines various pixel output parameters.								
Call	<code>set_pixel_line_3d(Channel, HalfPeriod, dx, dy, dz)</code>								
Parameters	<table> <tr> <td>Channel</td> <td>See set_pixel_line (number of the analog output port).</td> </tr> <tr> <td>HalfPeriod</td> <td>See set_pixel_line (<i>half</i> pixel output period).</td> </tr> <tr> <td>dx, dy</td> <td>See set_pixel_line: Distance in the X and Y directions between adjacent pixels in <i>bits</i> (64-bit IEEE floating point values)</td> </tr> <tr> <td>dz</td> <td>Distance in the Z direction between adjacent pixels in <i>bits</i> (64-bit IEEE floating point values) As with all Z-coordinates, dz too is scaled 16x smaller than dx and dy.</td> </tr> </table>	Channel	See set_pixel_line (number of the analog output port).	HalfPeriod	See set_pixel_line (<i>half</i> pixel output period).	dx, dy	See set_pixel_line : Distance in the X and Y directions between adjacent pixels in <i>bits</i> (64-bit IEEE floating point values)	dz	Distance in the Z direction between adjacent pixels in <i>bits</i> (64-bit IEEE floating point values) As with all Z-coordinates, dz too is scaled 16x smaller than dx and dy.
Channel	See set_pixel_line (number of the analog output port).								
HalfPeriod	See set_pixel_line (<i>half</i> pixel output period).								
dx, dy	See set_pixel_line : Distance in the X and Y directions between adjacent pixels in <i>bits</i> (64-bit IEEE floating point values)								
dz	Distance in the Z direction between adjacent pixels in <i>bits</i> (64-bit IEEE floating point values) As with all Z-coordinates, dz too is scaled 16x smaller than dx and dy.								
Comments	<ul style="list-style-type: none"> Unlike set_pixel_line, set_pixel_line_3d lets you define the pixel spacing (between two adjacent image points on a line) by a 3D vector. Unlike set_pixel_line, set_pixel_line_3d additionally requires two list-memory locations if parameter dz is non-zero. The initial component executes as a short list command before the primary component (a normal list command). Any still-pending delayed short list command gets executed first. In other respects, set_pixel_line_3d behaves similarly to set_pixel_line (see the comments there). 								
RTC4→ RTC5	New command. RTC4 compatibility mode: see set_pixel_line .								
Version info	Available as of version DLL 536, OUT 536.								
References	set_pixel_line , set_pixel , set_n_pixel								



Ctrl Command	set_port_default
Function	Defines the default output value for the selected output port.
Call	<code>set_port_default(Port, Value)</code>
Parameters	<p>Port Selection of the output port for which a default value shall be defined as an unsigned 32-bit value. Allowed values: = 0: ANALOG OUT1 output port (see also page 48) = 1: ANALOG OUT2 output port (see also page 48) = 2: 8-bit digital output port (see also page 52) = 3: 16-bit digital output port (see also page 51) = 4: 2-bit digital output port (see also page 48)</p> <p>Value Desired default value as an unsigned 32-bit value. Allowed values: For Port = 0/1: 12-bit values [0 ... 4095], higher bits are ignored. For Port = 2: 8-bit values [0 ... 255], higher bits are ignored. For Port = 3: 16-bit values [0 ... (2¹⁶-1)], higher bits are ignored. For Port = 4: 2-bit values [0 ... 3], higher bits are ignored. For Value = "-1" (= 2³²-1 = 0xFFFFFFFF), the corresponding default output functionality is switched off (see comment below).</p>
Comments	<ul style="list-style-type: none"> During initialization of the RTC5 (by load_program_file), the default values of all output ports are set to "-1" (= 2³²-1). If a default value ≠ "-1" has been specified for an output port, the port is set to this default value as soon as processing of a list has ended with stop_execution or by an external stop signal. For a default value of "-1", the corresponding output port is <i>not</i> addressed (any existing high-impedance state of the digital output ports is retained). Default values (≠ "-1") at the output ports 0/1 (ANALOG OUT1 or ANALOG OUT2) are also set at the end of pixel output mode (see chapter 8.8 "Pixel Output Mode – Scanning Raster Images (Bitmaps)", page 217). After initialization of position- and/or speed-dependent laser control by set_auto_laser_control, the corresponding default value (for output port 0, 1, 2 or 3, depending on the selected laser signal parameter Ctrl), is also outputted each time the laser is switched off after marking or when position- and/or speed-dependent laser control is set to another Ctrl parameter by set_auto_laser_control or deactivated by set_auto_laser_control (Ctrl = 0) (see page 161). If the default value is then defined as "-1", then the maximum allowed value (4095, 4095, 255 or 65535) is outputted. If the values for Port is invalid, then set_port_default is not executed (get_last_error return code: RTC5_PARAM_ERROR). The default values for the output ports 0-2 can also be defined by the command set_laser_off_default.
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified value for Port = 0/1 by 4 (the RTC4's analog outputs only handled 10-bit values).
References	set_laser_off_default , set_default_pixel



Ctrl Command	set_pulse_picking
Function	Switches on pulse picking laser mode.
Call	<code>set_pulse_picking(No)</code>
Parameter	No an unsigned 32-bit value. Allowed range: [0 ... 63]. = 0: LASER2 puts out the LASERON signal. = 1...63: LASER2 puts out each No th LASER1 pulse. If No > 63, No is clipped to 63 (get_last_error return code RTC5_PARAM_ERROR).
Comments	<ul style="list-style-type: none"> For pulse picking laser mode, see chapter 7.4.8 "Pulse Picking Laser Mode", page 157. The pulse picking signals are outputted until a different laser mode gets set by set_laser_mode (the pulse picking laser mode gets switched off if any other laser mode switches on by set_laser_mode). You can <i>not</i> switch on pulse picking laser mode by set_laser_mode.
RTC4→RTC5	New command.
Version info	Available as of version DLL 529, OUT 530, RBF 521.
References	set_laser_mode , set_pulse_picking_list

Ctrl Command	set_pulse_picking_length
Function	Defines a constant pulse length for the "laser active" LASER2 signal in pulse picking laser mode.
Call	<code>set_pulse_picking_length(Length)</code>
Parameter	Length Pulse length as an unsigned 32-bit value. Allowed range: [0 ... 65535]. Higher bits are ignored. 1 bit equals 1/64 µs. The default value after load_program_file is 0.
Comments	<ul style="list-style-type: none"> For the value to take effect, you must activate pulse picking laser mode by set_pulse_picking and constant pulse length mode by set_laser_control(bit #7 = 1). The value then takes immediate effect even during an active marking procedure. If pulse picking laser mode has been activated, but not constant pulse length mode (set_laser_control(bit #7 = 0)), then the pulse-picking signal uses the pulse length of the LASER1 signal (see pulse picking laser mode on page 157). If neither pulse picking laser mode nor constant pulse length mode has been activated, then the value Length is irrelevant (set_pulse_picking, set_laser_control and set_pulse_picking_length can be activated in any desired order). After set_pulse_picking(0), LASER2 is continuously output the LASERON signal, but no constant pulse length signal.
RTC4→RTC5	New command.
Version info	Available as of version DLL 533, OUT 534, RBF 524.



Undelayed Short List Command	set_pulse_picking_list
Function	Same as set_pulse_picking , but a list command.
Call	set_pulse_picking_list(No)
Parameter	No See set_pulse_picking .
RTC4→RTC5	New command.
Version info	Available as of version DLL 529, OUT 530, RBF 521.
References	set_pulse_picking

Ctrl Command	set_qswitch_delay
Function	In the YAG modes, defines the delay length of the first Q-Switch pulse with reference to the FirstPulseKiller signal (see also figure 51).
Call	set_qswitch_delay(Delay)
Parameter	Delay Q-Switch Delay as an unsigned 32-bit value. <i>1 bit equals 1/64 μs.</i> Allowed range: [0...(2 ²⁶ -1)].
Comments	<ul style="list-style-type: none"> Values over (2²⁶-1) are clipped. The YAG modes are selectable by set_laser_mode ([1, 2, 3 or 5]). Also for YAG modes defined with set_laser_mode([1-3]), the length of the Q-Switch delay can be subsequently changed by this command; and for YAG mode 2 also with set_firstpulse_killer or set_firstpulse_killer_list.
RTC4→RTC5	New command. In RTC4 compatibility mode, the delay value must be specified in units of 1/8 μs. It is then internally converted to 1/64 μs units (i.e. multiplied by 8). The allowed range of values is correspondingly smaller.
References	set_qswitch_delay_list , set_laser_control , set_laser_pulses_ctrl , set_laser_pulses , set_laser_timing , set_firstpulse_killer , set_firstpulse_killer_list

Undelayed Short List Command	set_qswitch_delay_list
Function	Same as set_qswitch_delay , but a list command.
Call	set_qswitch_delay_list(Delay)
Parameter	Delay Q-Switch Delay as an unsigned 32-bit value. <i>1 bit equals 1/64 μs.</i> Allowed range: [0...(2 ²⁶ -1)].
RTC4→RTC5	New command.
References	set_qswitch_delay



Ctrl Command	set_rot_center
Function	Sets the rotation center of a Processing-on-the-fly rotation correction (see set_fly_rot and set_fly_rot_pos).
Call	set_rot_center(X, Y)
Parameter	X, Y Position of the rotation center referenced to the zero point (0 0) of the image field as signed 32-bit values. Allowed range: [-2 ²⁴ ... (2 ²⁴ -1)]
Comments	<ul style="list-style-type: none"> For Processing-on-the-fly correction, see chapter 8.7.3 "Compensation of Rotary Movements", page 204. The position of the rotation center should be defined by set_rot_center or set_rot_center_list before the Processing-on-the-fly correction is activated by set_fly_rot or set_fly_rot_pos. The rotation center can also lie outside the image field (the allowed area is roughly equivalent to 30x the image field). Usage of a second scan head is only practical if it is set up for exactly the same rotational center.
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).
References	set_fly_rot , set_fly_rot_pos

Delayed Short List Command	set_rot_center_list
Function	Same as set_rot_center , but a list command.
Call	set_rot_center_list(X, Y)
Parameter	X, Y See set_rot_center .
RTC4→ RTC5	New command.
References	set_rot_center



Ctrl Command	set_RTC4_mode
Function	Sets RTC4 compatibility mode as the current DLL operation mode.
Call	<code>set_RTC4_mode()</code>
Comments	<ul style="list-style-type: none"> RTC4 compatibility mode is an optional operation mode of the DLL. It has been made available so that user programs written for the RTC4 can also be processed by the RTC5 (to a large extent) without needing to modify the programming code. However, a prerequisite here is that the program can only contain RTC4 commands that also exist with unchanged functionality as RTC5 commands. This user manual's list of commands, when applicable, notes such changes in the "RTC4→RTC5" section. RTC5 mode is pre-defined as the default DLL operation mode and can also be specified subsequently by set_RTC5_mode. The current DLL operation mode can be queried by get_RTC_mode. The <code>set_RTC4_mode</code> is also available without explicit access rights to a particular board. <code>set_RTC4_mode</code> is not available as a multi-board command. This command's scope is not board-specific, but rather global to the DLL and all RTC5 Boards to which the user program has access rights. The board-specific error variables <code>LastError</code> and <code>AccError</code> (see chapter 6.8 "Error Handling", page 98) are neither generated nor altered by <code>set_RTC4_mode</code>.
RTC4→RTC5	New command.
References	get_RTC_mode , set_RTC5_mode

Ctrl Command	set_RTC5_mode
Function	Sets RTC5 mode as the current DLL operation mode (default setting).
Call	<code>set_RTC5_mode()</code>
Comments	<ul style="list-style-type: none"> set_RTC4_mode sets the DLL operation mode to RTC4 compatibility mode. The current DLL operation mode can be queried by get_RTC_mode. The <code>set_RTC5_mode</code> command is also available without explicit access rights to a particular RTC5 Board. <code>set_RTC5_mode</code> is not available as a multi-board command. This command's scope is not board-specific, but rather global to the DLL and all RTC5 Boards to which the user program has access rights. The board-specific error variables <code>LastError</code> and <code>AccError</code> (see chapter 6.8 "Error Handling", page 98) are neither generated nor altered by <code>set_RTC5_mode</code>.
RTC4→RTC5	New command.
References	get_RTC_mode , set_RTC4_mode



Ctrl Command	set_scale
Function	Uses a specified scaling factor common to the X and Y axes to define the scaling matrix M_S for all subsequent coordinate transformations, see chapter 8.2 "Coordinate Transformations", page 183 .
Call	<code>set_scale(HeadNo, Scale, at_once)</code>
Parameters	<p>HeadNo Number of the scan head connector as an unsigned 32-bit value: = 1: The definition only affects the <i>primary</i> scan head connector. = 2: The definition only affects the <i>secondary</i> scan head connector (activation required). = 0, 3: The definition affects <i>both</i> scan head connectors. Only the two least significant bits are evaluated.</p> <p>Scale Scaling factor (as a 64-bit IEEE floating point value). Allowed range: [-16 ... +16]. If the parameter is set to an invalid value, it is set to 1. Negative values additionally produce a mirroring around both axes (corresponding to a 180° rotation).</p> <p>at_once This parameter (unsigned 32-bit value) determines when the defined transformation becomes effective: = 0: The new total transformation (total matrix and offset) is only calculated and applied to the current position when the next list command is executed. = 1: The new total transformation is calculated immediately (or before the next list command if a list is currently BUSY or the board is INTERNAL-BUSY) and applied to the current position. The "laser active" laser control signals are first switched off if necessary. = 2: The new total transformation (total matrix and offset) is only calculated and applied to the current position when the next jump_abs, jump_rel, goto_xy or goto_xyz command is executed. = 3: As with at_once = 1, but the laser control signals remain unaffected. > 3: See at_once = 2.</p>
Comments	• See chapter 8.2 "Coordinate Transformations", page 183 .
RTC4 → RTC5	New command.
Version info	Last change with version DLL 525, OUT 527: behavior for at_once = 3.
References	set_scale_list , set_angle , set_matrix , set_offset



Variable List Command	set_scale_list
Function	Same as set_scale , but a list command.
Call	set_scale_list(HeadNo, Scale, at_once)
Parameters	<p>HeadNo See set_scale.</p> <p>Scale See set_scale.</p> <p>at_once This parameter (unsigned 32-bit value) determines when the defined transformation becomes effective:</p> <ul style="list-style-type: none"> = 0: The transformation settings are only collected and intermediately stored, but the transformation is not processed as long as this is not activated by another coordinate transformation (e.g. by a list command with at_once = 1 or a corresponding control command). = 1: The transformation is immediately calculated (including all transformation settings that were collected until then) and processed prior to the next list command. The “laser active” laser control signals are first switched off if necessary. = 2: The transformation settings are only collected and intermediately stored (as with at_once = 0). However, The transformation is immediately calculated (including all transformation settings that were collected and intermediately stored until then) and applied to the current position when the next jump_abs, jump_rel, goto_xy or goto_xyz command is executed. = 3: As with at_once = 1, but the laser control signals remain unaffected. > 3: See at_once = 2
RTC4→ RTC5	New command.
Version info	Last change with version DLL 525, OUT 527: behavior for at_once = 3.
References	set_scale



Delayed Short List Command	set_scanner_delays
Function	Sets the scanner delays.
Call	<code>set_scanner_delays(Jump, Mark, Polygon)</code>
Parameters	Jump, Mark, Scanner delays as unsigned 32-bit values. Polygon <i>1 bit equals 10 µs.</i> Allowed range: [0...(2 ³² -1)].
Comments	<ul style="list-style-type: none"> Scanner delays are described in chapter 7.2.2 "Scanner Delays", page 113. Variable Delays for jumps and polylines can be set by set_delay_mode. If necessary, the specified delays are automatically adjusted by the RTC5 to avoid laser control errors (see "Automatic Delay Adjustments" on page 121). The default setting after load_program_file corresponds to <code>set_scanner_delays(9, 6, 3)</code>.
RTC4→ RTC5	Unchanged functionality (except for the extended range of values).
References	set_delay_mode , set_laser_delays

Ctrl Command	set_serial
Function	Sets the starting serial number of the serial-number-set most recently selected by select_serial_set (or of serial-number-set 0 after load_program_file) and sets the increment size for this serial-number-set to 1.
Call	<code>set_serial(No)</code>
Parameters	No Serial number as an unsigned 32-bit value. Allowed range: [0 ... (2 ³² -1)].
Comments	<ul style="list-style-type: none"> The command set_serial is synonymous with set_serial_step with Step = 1 (see comments there).
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). The command was previously only available for the RTC SCANalone Board, i.e. the standalone version of the RTC4 Board.
References	set_serial_step , select_serial_set



Ctrl Command	set_serial_step				
Function	Sets the starting serial number and the increment size for the serial-number-set most recently selected by select_serial_set (or for serial-number-set 0 after load_program_file).				
Call	<code>set_serial_step(No, Step)</code>				
Parameters	<table> <tr> <td>No</td><td>Serial number as an unsigned 32-bit value. Allowed range: [0 ... ($2^{32}-1$)].</td></tr> <tr> <td>Step</td><td>Increment size as an unsigned 32-bit value. Allowed range: [0 ... 9999]; only the last 4 decimal digits are used.</td></tr> </table>	No	Serial number as an unsigned 32-bit value. Allowed range: [0 ... ($2^{32}-1$)].	Step	Increment size as an unsigned 32-bit value. Allowed range: [0 ... 9999]; only the last 4 decimal digits are used.
No	Serial number as an unsigned 32-bit value. Allowed range: [0 ... ($2^{32}-1$)].				
Step	Increment size as an unsigned 32-bit value. Allowed range: [0 ... 9999]; only the last 4 decimal digits are used.				
Comments	<ul style="list-style-type: none"> During initialization, the starting serial number is set to 0 and the increment size set to 1. If mark_serial or mark_serial_abs was called with Mode $M_2 = 1$ (i.e. automatic serial-number incrementing was deactivated), then the increment size setting has no effect. If Step = 0, then incrementing does not occur, except in the case of markless marking (see mark_serial or mark_serial_abs: digits = 0), which always increments serial numbers by 1. For command usage, see chapter 7.5.2 "Marking Serial Numbers", page 170. 				
RTC4→ RTC5	New command.				
References	mark_serial , mark_serial_abs , set_serial , select_serial_set , select_serial_set_list				



Normal List Command	set_serial_step_list
Function	Sets the starting serial number and the increment size for the serial-number-set most recently selected by select_serial_set_list (or for serial-number-set 0 after load_program_file).
Call	<code>set_serial_step_list(No, Step)</code>
Parameters	No Serial number as an unsigned 32-bit value. Allowed range: [0 ... ($2^{32}-1$)]
	Step Increment size as an unsigned 32-bit value. Allowed range: [0 ... 9999]; only the last 4 decimal digits are used.
Comments	<ul style="list-style-type: none"> During initialization, the starting serial number (for mark_serial and mark_serial_abs) is set to 0 and the increment size set to 1. If mark_serial or mark_serial_abs was called with Mode $M_2 = 1$ (i.e. automatic serial-number incrementing was deactivated), then the increment size setting has no effect. If Step = 0, then incrementing does not occur, except in the case of markless marking (see mark_serial or mark_serial_abs: digits = 0), which always increments serial numbers by 1. For command usage, see chapter 7.5.2 "Marking Serial Numbers", page 170.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 537, OUT 537.
References	set_serial_step , select_serial_set_list , get_list_serial , mark_serial , mark_serial_abs

Ctrl Command	set_sky_writing
Function	Activates or deactivates Sky Writing mode and sets the corresponding parameters.
Call	<code>set_sky_writing(Timelag, LaserOnShift)</code>
Parameters	Timelag See set_sky_writing_para .
	LaserOnShift See set_sky_writing_para .
Comments	<ul style="list-style-type: none"> The command is identical to <code>set_sky_writing_para(Timelag, LaserOnShift, Nprev, Npost)</code> with <code>Nprev = approx. (0.15 × Timelag)</code> and <code>Npost = approx. (0.1 × Timelag)</code>. See also information about set_sky_writing_para and chapter 7.2.4 "Sky Writing", page 127.
RTC4→ RTC5	New command.
References	set_sky_writing_para , set_sky_writing_list



Ctrl Command	set_sky_writing_limit
Function	Defines the limit for Sky Writing switching in mode 3.
Call	set_sky_writing_limit(Limit)
Parameters	<p>Limit Desired limit as a 64-bit IEEE floating point value. Allowed range: [-1.0 ... +1.0]. Out-of-range values are edge-clipped.</p>
Comments	<ul style="list-style-type: none"> For command usage, see chapter 7.2.4 "Sky Writing", page 127. Limit is the cosine of the angular limit (for angular change between consecutive vectors or arcs within a polyline) for which a Sky Writing motion should be performed. The initialized value (after program start) is Limit = 0 (angular limit = 90°).
RTC4→ RTC5	New command.
Version info	Available as of version DLL 531, OUT 532.
References	set_sky_writing_limit_list

Undelayed Short List Command	set_sky_writing_limit_list
Function	Identical with set_sky_writing_limit , but a list command.
Call	set_sky_writing_limit_list(Limit)
Parameters	Limit See set_sky_writing_limit .
Comments	<ul style="list-style-type: none"> See set_sky_writing_limit.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 531, OUT 532.
References	set_sky_writing_limit

Normal List Command	set_sky_writing_list
Function	Identical with set_sky_writing , but a list command.
Call	set_sky_writing_list(Timelag, LaserOnShift)
Parameters	<p>Timelag See set_sky_writing_para. LaserOnShift See set_sky_writing_para.</p>
Comments	<ul style="list-style-type: none"> See set_sky_writing, set_sky_writing_para and set_sky_writing_para_list.
RTC4→ RTC5	New command.
References	set_sky_writing , set_sky_writing_para , set_sky_writing_para_list .



Ctrl Command	set_sky_writing_mode
Function	Switches Sky Writing mode on or off.
Call	<code>set_sky_writing_mode(Mode)</code>
Parameters	<p>Mode Desired Sky Writing mode as an unsigned 32-bit value. Allowed range: [0 ... ($2^{32}-1$)].</p> <ul style="list-style-type: none"> = 0: Sky Writing (mode 1 or possibly mode 2 or 3) is deactivated. = 1: Sky Writing mode 1 is activated. = 2: Sky Writing mode 2 is activated. > 2: Sky Writing mode 3 is activated.
Comments	<ul style="list-style-type: none"> • You can only activate Sky Writing mode 2 and 3 (by <code>Mode > 1</code>) if you had previously (recently) called set_sky_writing_para with <code>Timelag ≥ 1/64</code> at least once (thus activating Sky Writing mode 1). • After set_sky_writing_mode(<code>Mode = 0</code>), subsequent Sky Writing reactivation is possible by set_sky_writing_mode(<code>Mode > 0</code>), but not after set_sky_writing_para with <code>Timelag < 1/64</code> (reactivation here would require a renewed call of set_sky_writing_para with <code>Timelag ≥ 1/64</code>). • Each mode switch by set_sky_writing_mode becomes effective as of the next list command. • If you reactivate Sky Writing mode 1 (switching from <code>Mode = 0</code> to <code>1</code>) during execution of a list, then an already-begun mark or arc command still executes to completion without Sky Writing. • If you deactivate Sky Writing mode 1 (switching from <code>Mode = 1</code> to <code>0</code>) while a list is executing, then a mark or arc command already begun in Sky Writing mode 1 still executes to completion in Sky Writing mode 1 and then is appended with a mark delay. • Activation or deactivation of Sky Writing mode 2 or 3 (switching to or from <code>Mode = 2</code> or <code>3</code>) is not possible if the board's BUSY status is currently set (list is being processed or has been halted by pause_list). Then the command is ignored (get_last_error return code: <code>RTC5_BUSY</code>). In contrast, the command is always executed when a list has been paused by set_wait.
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> • Available as of version DLL 527, OUT 529, RBF 519. • Last change with version DLL 531, OUT 532: mode 3.
References	set_sky_writing_mode_list , set_sky_writing_para



Normal List Command	set_sky_writing_mode_list
Function	Identical with set_sky_writing_mode , but a list command.
Call	set_sky_writing_mode_list(Mode)
Parameters	Mode See set_sky_writing_mode .
Comments	<ul style="list-style-type: none"> Unlike set_sky_writing_mode, the command set_sky_writing_mode_list lets you activate or deactivate Sky Writing mode 2 or 3 (switching to or from Mode = 2 or 3) even within a list. Here, a temporary switch to Sky Writing mode 1 occurs. Deactivation of Sky Writing mode 1 by set_sky_writing_mode_list (switching from Mode = 1 to 0) results in the addition of a mark delay defined prior to activation of Sky Writing – provided that no other delay is in effect (e.g. a jump delay).
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> Available as of version DLL 527, OUT 529, RBF 519. Last change with version DLL 531, OUT 532: mode 3.
References	set_sky_writing_mode



Ctrl Command	set_sky_writing_para
Function	Activates or deactivates Sky Writing mode and sets the corresponding parameters.
Call	<code>set_sky_writing_para(Timelag, LaserOnShift, Nprev, Npost)</code>
Parameters	<p>Timelag Sky Writing parameter as a 64-bit IEEE floating point value. <i>1.0 equals 1 µs.</i> Up to DLL 536, the following applies: ≥ 1/4: Sky Writing mode 1 is activated. < 1/8: Sky-Writing (mode 1 or mode 2 or 3 when applicable) is deactivated. As of DLL 537, the following applies: ≥ 1/4: Sky-Writing mode 1 is activated. From 1/8 to 1/4, Sky-Writing is activated, but Timelag = 0 is internally applied. < 1/8: Sky-Writing (mode 1 or mode 2 or 3 when applicable) is deactivated.</p> <p>LaserOnShift Shift (for positive values: delay) of the point of time, where the laser control signals are switched on as a signed 32-bit value (see figure 45). <i>1 bit equals 0.5 µs.</i> Negative values smaller than the complete run-in phase are clipped, therefore the following applies automatically: – LaserOnShift ≥ approx. $(-40) \times N_{prev}$ (mode 1). – LaserOnShift ≥ approx. $(-20) \times N_{prev}$ (mode 2 or 3).</p> <p>Nprev This parameter (an unsigned 32-bit value) defines the duration of the run-in phase (see figure 45). <i>1 bit equals 10 µs.</i> – Run-in duration / µs = $20 \times N_{prev}$ (mode 1). – Run-in duration / µs = $10 \times N_{prev}$ (mode 2 or 3). – Allowed range: $0 \leq N_{prev} \leq (2^{32}-1)$.</p> <p>Npost This parameter (an unsigned 32-bit value) defines the duration of the run-out phase (see figure 45). <i>1 bit equals 10 µs.</i> – Run-out duration / µs = $20 \times N_{post}$ (mode 1). – Run-out duration / µs = $10 \times N_{post}$ (mode 2 or 3). – Allowed range: $0 \leq N_{post} \leq (2^{32}-1)$.</p>



Ctrl Command	set_sky_writing_para
Comments	<ul style="list-style-type: none"> For information on Sky Writing mode and a description of the parameters, see chapter 7.2.4 "Sky Writing", page 127. If $N_{prev} \geq 65535$, then set_sky_writing_para behaves similarly to set_sky_writing: N_{prev} is set to a value of approx. $(0.15 \times \text{Timelag})$, i.e. <ul style="list-style-type: none"> run-in duration / $\mu\text{s} = \text{approx. } 3 \times \text{Timelag}$ (mode 1) run-in duration / $\mu\text{s} = \text{approx. } 1.5 \times \text{Timelag}$ (mode 2 or 3) Here, N_{prev} gets clipped to the maximum value of 65535 if necessary. If $N_{post} \geq 65535$, then set_sky_writing_para behaves similarly to set_sky_writing: N_{post} is set to a value of approx. $(0.1 \times \text{Timelag})$, i.e. <ul style="list-style-type: none"> run-out duration / $\mu\text{s} = \text{approx. } 2 \times \text{Timelag}$ (mode 1) run-out duration / $\mu\text{s} = \text{approx. } 1 \times \text{Timelag}$ (mode 2 or 3) Here, N_{post} gets clipped to the maximum value of 65535 if necessary. set_sky_writing_para cannot be used to switch back and forth between Sky Writing modes 1, 2 and 3 (the proper command for this is set_sky_writing_mode). If you call set_sky_writing_para ($\text{Timelag} \geq 1/64$) while Sky Writing is deactivated, then Sky Writing mode 1 gets activated. If the command call succeeds during execution of a list, then mode 1 only become active as of the next list command (an already-begun mark or arc command still executes to completion <i>without</i> Sky Writing). If you call set_sky_writing_para while Sky Writing mode 1 is activated, then the supplied parameters become active as of the next list command. If the command call succeeds during execution of a list, then: <ul style="list-style-type: none"> If ($\text{Timelag} \geq 1/64$), then Sky Writing for an already begun mark or arc command executes to completion still using the existing parameters. If ($\text{Timelag} < 1/64$), then a mark or arc command already begun in Sky Writing mode 1 still completes in Sky Writing mode 1 and then is appended with a mark delay. When you call set_sky_writing_para while both Sky Writing mode 2 or 3 is activated and the board's BUSY status is currently set (list is being processed or has been halted by pause_list), then the command is ignored (get_last_error return code: RTC5_BUSY). In contrast, the command executes when no list is currently running or when a list has been paused by set_wait. <ul style="list-style-type: none"> If $\text{Timelag} < 1/64$, then Sky Writing mode 2 or 3 is deactivated. If $\text{Timelag} \geq 1/64$, then the supplied parameters become effective as of the next list command. For Sky Writing mode 2 or 3, however, $N_{prev} = 0$ and/or $N_{post} = 0$ automatically get(s) internally corrected to 1 (this is not treated as an error). If you subsequently activate Sky Writing mode 1 (by set_sky_writing_mode), then $N_{prev} = 0$ and/or $N_{post} = 0$ is/are reused.
RTC4 → RTC5	New command.
Version info	Last change with version DLL 537, OUT 537: Timelag.
References	set_sky_writing , set_sky_writing_para_list



Normal List Command	set_sky_writing_para_list
Function	Identical with set_sky_writing_para , but a list command.
Call	set_sky_writing(Timelag, LaserOnShift, Nprev, Npost)
Parameters	Timelag See set_sky_writing_para .
	LaserOnShift See set_sky_writing_para .
	Nprev See set_sky_writing_para .
	Npost See set_sky_writing_para .
Comments	<ul style="list-style-type: none"> • Unlike set_sky_writing_para, the command set_sky_writing_para_list is also executable within a list if Sky Writing mode 2 or 3 is active. Here, Sky Writing temporarily switches from mode 2 or 3 to mode 1: if necessary, the immediately preceding marking command completes in mode 1 and the next marking command, too, begins in mode 1. Afterward, mode 2 or 3 gets reactivated if $\text{Timelag} \geq 1/64$. The following also applies: <ul style="list-style-type: none"> – If $\text{Timelag} \geq 1/64$, then the supplied parameters become effective as of the next list command ($\text{Nprev} = 0$ and/or $\text{Npost} = 0$: see set_sky_writing_para). – If $\text{Timelag} < 1/64$, then Sky Writing is permanently deactivated. • As with set_sky_writing_para, you cannot use set_sky_writing_para_list to permanently switch from Sky Writing mode 2 or 3 to Sky Writing mode 1. • Deactivation of Sky Writing mode 1, 2 or 3 by set_sky_writing_para_list results in the addition of a mark delay defined prior to activation of Sky Writing – provided that no other delay is in effect (e.g. a jump delay).
RTC4→ RTC5	New command.
Version info	Last change with version DLL 531, OUT 532: mode 3.
References	set_sky_writing_para



Ctrl Command	set_softstart_level		
Function	Sets the softstart values.		
Call	set_softstart_level(Index, Level)		
Parameters	All parameters are unsigned 32-bit values.		
	Parameter	Allowed Values	Description
	Index	0 ... 1023	Index number of the softstart table value.
	Level	0 ... $2^{12}-1$	For softstart mode = 1, 2, 11 and 12 (see set_softstart_mode). Value 0 corresponds to an analog voltage value of 0 V. Value $2^{12}-1$ corresponds to 10 V.
		0 ... $(2^{32}-1)$	For softstart mode = 3 and 13 (see set_softstart_mode). 1 bit corresponds to a pulse length of 1/64 μ s (in RTC5 mode) or 1/8 μ s (in RTC4 compatibility mode).
Comments	<ul style="list-style-type: none"> The set_softstart_level command defines the <code>Level</code> value for the pulse number <code>Index</code>. The command must be called separately for each individual <code>Level</code> value. <code>Index</code> must be in the range [0...1023], otherwise the set_softstart_level command is not executed (get_last_error return code <code>RTC5_PARAM_ERROR</code>). If <code>Index</code> ≥ <code>Number</code> (<code>Number</code> is defined by set_softstart_mode) then set_softstart_level is executed but the defined value is not used during softstart. Be sure to set as many softstart values as you defined by set_softstart_mode (unspecified softstart values have undefined contents). The softstart mode is enabled by the command set_softstart_mode. For softstart modes 1, 2, 11 or 12, <code>Level</code> is restricted to 12 bits; higher bits are ignored (see also write_da_x or write_da_x_list). If Softstart mode = 3 or 13 and the specified softstart pulse length is larger than the laser signal period duration specified by set_laser_pulses or set_laser_timing ($2 \times \text{HalfPeriod}$), then the laser remains continuously on. Particularly if softstart values are to be outputted as analog voltage values (softstart mode = 1, 2, 11 or 12), the set_softstart_mode command must be called <i>before</i> first-time use of the set_softstart_level command; otherwise the specified value is not correctly converted to an analog value. Also observe the note in section "Softstart Mode" on page 152. If the "freely definable wobbel shape" wobbel mode is activated, then this command has no effect, see chapter 8.4 "Wobbel Mode", page 189. 		
RTC4→RTC5	Essentially unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, analog voltage levels are internally multiplied by 4 (here $2^{10}-1$ corresponds to 10 V; the values are converted from 10 to 12 bits) and pulse length values are multiplied by 8 (here 1 bit corresponds to 1/8 μ s; the values are converted from units of 1/8 μ s to units of 1/64 μ s; the 1 μ s timebase – see set_laser_timing – is no longer supported). The allowed range of values for <code>Level</code> is correspondingly smaller. Due to this difference in the handling of analog voltage and pulse length values, set_softstart_mode must always be called before set_softstart_level .		
References	set_softstart_mode , set_softstart_mode_list , set_softstart_level_list		



Normal List Command	set_softstart_level_list
Function	Similar to set_softstart_level , but a list command.
Call	set_softstart_level_list(Index, Level1, Level2, Level3)
Parameters	See set_softstart_level .
Comments	<ul style="list-style-type: none"> The set_softstart_level command defines three Level values for the pulses number Index through Index+2. The command must be called separately for each individual Level1/2/3 value-triple. Index must be in the range [0...1023], otherwise the set_softstart_level_list command is, already during loading, replaced by a list_nop (get_last_error return code RTC5_PARAM_ERROR). Softstart does not use level values where the index count exceeds Number (Number is specified by set_softstart_mode). Otherwise this command is identical to the control command set_softstart_level (see also the comments there). If the “freely defined wobbel shape” wobbel mode is activated, then this command has no effect, see chapter 8.4 “Wobbel Mode”, page 189.
RTC4→ RTC5	New command. In RTC4 compatibility mode: see set_softstart_level .
Version info	Available as of version DLL 536, OUT 536.
References	set_softstart_level , set_softstart_mode , set_softstart_mode_list



Ctrl Command	set_softstart_mode		
Function	Switches softstart mode on or off.		
Call	set_softstart_mode(Mode, Number, Delay)		
Parameters	All parameters are unsigned 32-bit values.		
	Parameter	Allowed Values	Description
	Mode	= 0 = 1, 11 = 2, 12 = 3, 13	Disables the softstart mode, but does not remove previously loaded softstart values. The softstart values defined by set_softstart_level are outputted as analog voltage values at the ANALOG OUT1 port. The softstart values defined by set_softstart_level are outputted as analog voltage values at the ANALOG OUT2 port. The softstart values defined by set_softstart_level specify the pulselenghts of the first Number laser signal pulses at the LASER1 port (neither in laser mode 4 nor in laser mode 6).
	Number	1 ... 1024	Number of softstart values to be transmitted.
	Delay	0 ... (2^{32} –1). <i>1 bit equals</i> 10 µs.	Defines the time period for which the laser must have been switched off before softstart is activated at the next switch-on.
Comments	<ul style="list-style-type: none"> The command set_softstart_mode must be called <i>before</i> first-time use of the set_softstart_level command, particularly if softstart values are to be outputted as analog voltage values (default setting for softstart value handling: Mode = 3). The individual softstart values Level(0) – Level(Number – 1) must be provided individually by separate calls of set_softstart_level. Indices for which no Level value was supplied are undefined. If the values for Mode or Number are invalid, then softstart mode is switched off. When softstart mode is switched off, the already transmitted values are not deleted and can be further used after a renewed switch-on. Here, you should not unintentionally switch between analog voltage values (Mode = 1, 2, 11 or 12) and pulselenghts (Mode = 3 or 13). In contrast, you can switch at any time between ANALOG OUT1 (Mode = 1 or 11) and ANALOG OUT2 (Mode = 2 or 12). The softstart mode is incompatible with the pixel mode (see page 217). The command set_softstart_mode is executed, but softstart is not executed while pixel mode is operational. The softstart mode should <i>not</i> be used in conjunction with automatic laser control (see page 159) if automatic laser control readjusts the 12-bit output values at the ANALOG OUT1 or ANALOG OUT2 output ports or pulse lengths (PulseLength) or output periods (HalfPeriod) of the laser signals LASER1 and LASER2. Softstart mode cannot be combined with "freely definable wobble shapes". If one is defined and activated, the command has no effect. Observe the notes in chapter 7.4.5 "Softstart Mode", page 152. 		



Ctrl Command	set_softstart_mode
RTC4→ RTC5	<p>Essentially unchanged functionality (except for the extended range of values). Instead of acquiring analog softstart values with the trailing edge of the laser signal pulse (as achievable in the RTC4 with <code>Mode = 1, 2 or 3</code>), a 180° phase-shifted laser signal can be used as a trigger signal for realizing phase-shifted data acquisition with the RTC5 (see notes in chapter 7.4.5 "Softstart Mode", page 152). An index shift no longer exists under any circumstances.</p>
Version info	Last change with version OUT 515.
References	set_softstart_level , set_softstart_level_list , set_softstart_mode_list

Variable List Command	set_softstart_mode_list
Function	Same as set_softstart_mode , but a list command.
Call	<code>set_softstart_mode_list(Mode, Number, Delay)</code>
Parameters	See set_softstart_mode .
Comments	<ul style="list-style-type: none"> • <code>set_softstart_mode_list</code> switches off the “laser active” laser control signals by executing <code>list_nop</code> and waits until the LaserOff delay has expired. • Otherwise this command is identical to the control command <code>set_softstart_mode</code> (see also the comments there).
RTC4→ RTC5	New command.
Version info	Available as of version DLL 536, OUT 536.
References	set_softstart_mode , set_softstart_level , set_softstart_level_list



Ctrl Command	set_standby
Function	Defines the output period and the pulse length of the standby pulses for "laser standby" operation or – in laser mode 4 and laser mode 6 – the continuously-running laser signals for "laser active" and "laser standby" operation.
Call	<code>set_standby(HalfPeriod, PulseLength)</code>
Parameters	<p>HalfPeriod <i>half of the standby output period as an unsigned 32-bit value. 1 bit equals 1/64 µs. Allowed range: [0 ... +(2³²-1)].</i></p> <p>PulseLength <i>Pulse length of the standby pulses as an unsigned 32-bit value. 1 bit equals 1/64 µs. Allowed range: [0 ... (2³²-1)].</i></p>
Comments	<ul style="list-style-type: none"> After a hardware reset, (and after load_program_file), the LASER1, LASER2 and LASERON ports must be first-time-activated with set_laser_control before standby pulses can be activated by set_standby or set_standby_list (see page 144). At the same time, the signal level of the standby pulses can be defined with set_laser_control. The standby pulses are available in all laser modes (YAG 1/2/3/5, CO₂, laser mode 4 and 6). They can be deactivated (turned off) by setting the standby pulse length and/or the standby output period to zero (default). Note that <i>half</i> the standby output period must be specified for HalfPeriod (see figure 50 and figure 52). If the softstart mode is used (see page 152), HalfPeriod should not be smaller than 104 (this is not automatically checked). This value corresponds to a laser pulse frequency of approx. 308 kHz. If PulseLength is larger than the output period ($2 \times$ HalfPeriod), the laser is permanently on. The time base for the stand-by pulses is always 64 MHz (i.e. 1 bit equals 1/64 µs). The laser control mode has to be set with the command set_laser_mode (see also chapter 7.4 "Laser Control", page 144). To set the active output period and pulse length for the "laser active" laser control signals (beside for laser mode 4 and 6), use the command set_laser_pulses_ctrl, set_laser_pulses or set_laser_timing.
RTC4→ RTC5	<p>Essentially unchanged functionality (except for the extended range of values), however:</p> <ul style="list-style-type: none"> HalfPeriod and PulseLength must be specified in units of 1/64 µs with the RTC5; in RTC4 compatibility mode they are specified in units of 1/8 µs (as with the RTC4). In RTC4 compatibility mode, the RTC5 multiplies the specified values by 8 (the allowed range of values is correspondingly reduced; the smallest reasonable value for HalfPeriod with softstart mode is then 13). Under some circumstances, set_laser_control must be called before set_standby (see above).
References	set_standby_list , get_standby



Delayed Short List Command	set_standby_list
Function	Same as set_standby , but a list command.
Call	<code>set_standby_list(HalfPeriod, PulseLength)</code>
Parameters	HalfPeriod <i>half of the stand-by output period as an unsigned 32-bit value. 1 bit equals 1/64 µs. Allowed range: [0 ... +(2³²-1)].</i>
	PulseLength <i>Pulse length of the stand-by pulses as an unsigned 32-bit value. 1 bit equals 1/64 µs. Allowed range: [0 ... (2³²-1)].</i>
RTC4→ RTC5	as set_standby

Ctrl Command	set_start_list
Function	Opens the list buffer for writing of list commands and sets the input pointer to the start of the desired list ("List 1" or "List 2"). The next list command is stored at this address and all further list commands at the subsequent addresses in the selected list.
Call	<code>set_start_list(ListNo)</code>
Parameter	ListNo Number of the list in which the input pointer should be set, as an unsigned 32-bit value. Allowed values: [uneven: "List 1", even: "List 2"].
Comments	<ul style="list-style-type: none"> The <code>set_start_list</code> command is synonymous with <code>set_start_list_pos</code> for Pos = 0. Alternatively, the commands <code>set_start_list_1</code> and <code>set_start_list_2</code> (with no parameter) can be used.
RTC4→ RTC5	Unchanged functionality.
References	execute_list , read_status , set_start_list_pos

Ctrl Command	set_start_list_1
Function	See set_start_list .
Call	<code>set_start_list_1()</code>
RTC4→ RTC5	Unchanged functionality.
References	set_start_list

Ctrl Command	set_start_list_2
Function	See set_start_list .
Call	<code>set_start_list_2()</code>
RTC4→ RTC5	Unchanged functionality.
References	set_start_list



Ctrl Command	set_start_list_pos
Function	Opens the list buffer for writing of list commands and sets the input pointer to the specified (relative) position in the desired list ("List 1" or "List 2"). The next list command is stored at this address and all further list commands at the subsequent addresses in the selected list.
Call	<code>set_start_list_pos(ListNo, Pos)</code>
Parameters	ListNo Number of the list for which the input pointer should be set, as an unsigned 32-bit value. Allowed values: [uneven: "List 1", even: "List 2"].
	Pos Position of the input pointer (offset relative to the start of the respective list) as an unsigned 32-bit value. Allowed value range: [0 ... (2^{20} -1)].
Comments	<ul style="list-style-type: none"> The selected list is unconditionally opened for loading. There is no checking of whether the list is currently being processed (see also chapter 6.4.1 "Loading Lists", page 75 and load_list). The input pointer <i>cannot</i> be set to the protected area ("List 3"). If the protected area is to be written to (at the full risk of users) with <code>set_start_list_pos</code>, then this area can be temporarily allocated to "List 2" by <code>ConfigList(Mem1, -1)</code>. After writing, configuration of the area's protection should be restored: <code>ConfigList(Mem1, Mem2)</code>. For uneven <code>ListNo</code> values, "List 1" is opened, otherwise "List 2". This facilitates continuous automatic list changing through incrementing counts. If "List 2" has not been assigned memory (<code>Mem2 = 0</code>, see config_list) then "List 1" is opened. If <code>Pos</code> is specified as being larger than the memory area of the respective list (<code>Pos > Mem1</code> or <code>Pos > Mem2</code>), then <code>Pos</code> is set to 0. The status values of the selected list (see also read_status) are set as follows: LOAD set, READY reset, USED reset. The LOAD status of the other corresponding list is reset. The <code>set_start_list_pos</code> command triggers a flush of the list input buffer (see page 75). <code>set_start_list_pos</code> also covers the specialized variants set_start_list_1, set_start_list_2, set_start_list and set_input_pointer. <i>CAUTION: If the end of the respective list buffer area is reached, the list input pointer is automatically reset to the start of the same list buffer area.</i> <i>Make sure not to overwrite any commands still needed by your user program.</i>
RTC4→ RTC5	New command.
References	execute_list_pos , read_status



Ctrl Command	set_sub_pointer
Function	Stores the absolute start address of a command list in the internal management table for indexed subroutines.
Call	<code>set_sub_pointer(Index, Pos)</code>
Parameters	Index Index of the indexed subroutine whose starting address <code>Pos</code> should be entered in the management table (as an unsigned 32-bit value). Allowed range: [0 ... 1023].
	Pos Absolute start address as an unsigned 32-bit value. Allowed range: [0 ... $(2^{20}-1)$].
Comments	<ul style="list-style-type: none"> If <code>Index > 1023</code> and/or <code>Pos > (2²⁰-1)</code>, then the command is <i>not</i> executed (<code>get_last_error</code> return code <code>RTC5_PARAM_ERROR</code>). The <code>set_sub_pointer</code> command can be used for referencing a nonindexed subroutine, which thereby becomes an indexed subroutine that is protectable by <code>save_disk/load_disk</code> and/or callable by the index. <code>set_sub_pointer</code> can also be used to reference anew an indexed subroutine, character or text string so that it can also be called by a second index. Here, it is preferable to use the <code>copy_dst_src</code> command for index management. The start addresses of command lists that are to be referenced with <code>set_sub_pointer</code> can be queried by <code>get_input_pointer</code> before loading the command lists. <code>set_sub_pointer</code> only stores <i>starting addresses</i> in the internal management table. An indexed subroutine only gains protection by a subsequent <code>save_disk/load_disk</code> command. <code>Pos</code> should not be an arbitrary address within a list. Instead, it should be the starting address of an actually existing subroutine that was finalized by <code>list_return</code> and does not contain <code>set_end_of_list</code>.
RTC4→ RTC5	New command.
References	load_sub



Ctrl Command	set_text_table_pointer
Function	Stores the absolute start address of a command list in the internal management table for indexed text strings.
Call	<code>set_text_table_pointer(Index, Pos)</code>
Parameters	<p>Index Index of the indexed text string whose starting address <code>Pos</code> should be entered in the management table (as an unsigned 32-bit value, allowed range: [0 ... 41]). The same ordering applies as for the load_text_table command:</p> <ul style="list-style-type: none"> = 0...9: Digits for marking the time and date [0 ... 9]. = 10...21: Months [January ... December]. = 22...28: Days-of-the-week [Sunday ... Saturday]. = 29: Blank character for marking serial numbers. = 30...39: Digits for marking serial numbers [0 ... 9]. = 40: Text for "a.m." = 41: Text for "p.m." <p>Pos Absolute start address as an unsigned 32-bit value. Allowed range: [0 ... $(2^{20}-1)$].</p>
Comments	<ul style="list-style-type: none"> • Indexed text strings can be used for marking time, date or serial numbers (see "Calling Indexed Text Strings", page 89). • If <code>Index > 41</code> and/or <code>Pos > (2²⁰-1)</code>, then the command is <i>not</i> executed (get_last_error return code <code>RTC5_PARAM_ERROR</code>). • The set_text_table_pointer command can be used for referencing a nonindexed subroutine, which thereby becomes an indexed text string that is protectable by save_disk/load_disk and/or callable by the index. • set_text_table_pointer can also be used to reference anew an indexed subroutine, character or text string so that it can also be called by a second index. Here, it is preferable to use the copy_dst_src command for index management. • The start addresses of command lists that are to be referenced with set_text_table_pointer can be queried by get_input_pointer before loading the command lists. • set_text_table_pointer only stores <i>starting addresses</i> in the internal management table. An indexed text string only gains protection by a subsequent save_disk/load_disk command. • <code>Pos</code> should not be an arbitrary address within a list. Instead, it should be the starting address of an actually existing subroutine that was finalized by list_return and does not contain set_end_of_list. • The command is synonymous with set_char_table, which was introduced for the RTC SCANalone Board (standalone version of the RTC4 Board).
RTC4→ RTC5	New command.
References	load_text_table , mark_date , mark_serial , mark_time

Delayed Short List Command	set_trigger
Function	Starts measurement and storage of the specified signals.
Call	<code>set_trigger(Period, Signal1, Signal2)</code>
Parameters	<p>Period Measurement period as an unsigned 32-bit value. <i>1 bit equals 10 µs.</i> <i>Allowed range: [0 ... (2³²-1)].</i></p> <p>Signal1, desired signal type for measurement channels 1 and 2 as an unsigned 32-bit Signal2 value. Allowed range: as listed below.</p> <ul style="list-style-type: none"> = 0: LASERON signal (1 = laser signal on, 0 = laser signal off). = 1: StatusAX (X-axis status signal of the primary scan head connector). = 2: StatusAY (Y-axis status signal of the primary scan head connector). = 3: Reserved. = 4: StatusBX (X-axis status signal of the secondary scan head connector). = 5: StatusBY (Y-axis status signal of the secondary scan head connector). = 6: Reserved. = 7: SampleX (X-axis cartesian control value). = 8: SampleY (Y-axis cartesian control value). = 9: SampleZ (Z-axis cartesian control value). = 10: SampleAX_Corr (corrected X-axis control value for the primary scan head connector). = 11: SampleAY_Corr (corrected Y-axis control value for the primary scan head connector). = 12: SampleAZ_Corr (corrected Z-axis control value, if XY are connected to the primary scan head connector; identical to the effective output value for the Z axis). = 13: SampleBX_Corr (corrected X-axis control value for the secondary scan head connector). = 14: SampleBY_Corr (corrected Y-axis control value for the secondary scan head connector). = 15: SampleBZ_Corr (corrected Z-axis control value, if XY are connected to the secondary scan head connector; identical to the effective output value for the Z axis). = 16: StatusAX+LASERON (StatusAX for laser signal on, -524288 for laser signal off). = 17: StatusAY+LASERON (StatusAY for laser signal on, -524288 for laser signal off). = 18: StatusBX+LASERON (StatusBX for laser signal on, -524288 for laser signal off). = 19: StatusBY+LASERON (StatusBY for laser signal on, -524288 for laser signal off).



Delayed Short List Command	set_trigger
	<p>Signal1, = 20: SampleAX_Out (effective X-axis output value for the primary scan head connector; if applicable incl. any scanner offset and gain compensation, see comments; not usable for measuring Z-axis output values).</p> <p>= 21 SampleAY_Out (effective Y-axis output value for the primary scan head connector; not usable for measuring Z-axis output values).</p> <p>= 22: SampleBX_Out (effective X-axis output value for the secondary scan head connector; not usable for measuring Z-axis output values).</p> <p>= 23 SampleBY_Out (effective Y-axis output value for the secondary scan head connector; not usable for measuring Z-axis output values).</p> <p>= 24: Laser control parameter of automatic laser control (see set_auto_laser_control).</p> <p>= 25: SampleAX_Trans (transformed X-axis control value for the primary scan head connector).</p> <p>= 26: SampleAY_Trans (transformed Y-axis control value for the primary scan head connector).</p> <p>= 27: SampleAZ_Trans (transformed Z-axis control value, if XY are connected to the primary scan head connector).</p> <p>= 28: SampleBX_Trans (transformed X-axis control value for the secondary scan head connector).</p> <p>= 29: SampleBY_Trans (transformed Y-axis control value for the secondary scan head connector).</p> <p>= 30: SampleBZ_Trans (transformed Z-axis control value, if XY are connected to the secondary scan head connector).</p> <p>= 31: Laser control parameter of vector-defined laser control (see set_vector_control).</p> <p>= 32: Focus shift (see set_vector_control, set_defocus, set_defocus_list).</p> <p>= 33: 12-bit output value at the ANALOG OUT1 output port (see set_auto_laser_control, set_vector_control and chapter 9.1.4 "12-Bit Analog Output Ports").</p> <p>= 34: 12-bit output value at the ANALOG OUT2 output port (see set_auto_laser_control, set_vector_control and chapter 9.1.4 "12-Bit Analog Output Ports").</p> <p>= 35: Output value at the 16-bit digital output port (see set_auto_laser_control, set_vector_control and chapter 9.1.1 "16-Bit Digital Output Port")</p>



Delayed Short List Command	set_trigger
	<p>Signal1, = 36: Output value at the 8-bit digital output port (see set_auto_laser_control, set_vector_control and chapter 9.1.2 "8-Bit Digital Output Port").</p> <p>Signal2 (cont'd)</p> <p>= 37: Pulse length (PulseLength) of the LASER1 and LASER2 laser signals (see set_auto_laser_control, set_vector_control).</p> <p>= 38: Output period (HalfPeriod) of the LASER1 and LASER2 laser signals (see set_auto_laser_control, set_vector_control).</p> <p>= 39: FreeVariable0.</p> <p>= 40: FreeVariable1.</p> <p>= 41: FreeVariable2.</p> <p>= 42: FreeVariable3.</p> <p>= 43: Counter value of encoder counter Encoder0.</p> <p>= 44: Counter value of encoder counter Encoder1.</p> <p>= 45: Marking speed (from set_mark_speed, set_mark_speed_ctrl).</p> <p>= 46: 16-bit digital input (EXTENSION 1).</p> <p>= 47: Zoom value (only for intelliWELD II).</p> <p>= 48: FreeVariable4.</p> <p>= 49: FreeVariable5.</p> <p>= 50: FreeVariable6.</p> <p>= 51: FreeVariable7.</p> <p>= 52: Time stamp counter (see chapter 8.13 "Time Measurements", page 231).</p> <p>= 53: Wobbel amplitude (see set_wobbel, set_wobbel_mode and chapter 8.4 "Wobbel Mode", page 189).</p> <p>= 54: ReadAnalogIn (see read_analog_in).</p>
Comments	<ul style="list-style-type: none"> If Signal1 and Signal2 are not from the above list, then the command is replaced with a list_nop, even if Period = 0 (get_last_error return code RTC5_PARAM_ERROR). After a set_trigger command (with Period > 0), a data pair is stored immediately and subsequently at intervals defined by the specified measurement period. Data storage continues until the command is called again (with Period = 0), or until set_trigger4(Period = 0) is called or until the full data storage capacity is reached (automatic termination at the limit of 2²⁰ data pairs). If set_trigger was preceded by a load_correction_file which loaded a file as correction table №=3 or №=4, then the memory area's upper half reserved for data recording by set_trigger is already occupied. Then, only 2¹⁹ data pairs are recordable with set_trigger and measurement recording automatically terminates at this value. If loading of correction table №=3 or №=4 occurs after more than 2¹⁹ data pairs had been recorded by set_trigger, then the values beyond 2¹⁹ are no longer available. load_program_file reestablishes the original state of up to 2²⁰ possible measurement pairs.

Delayed Short List Command	set_trigger
Comments (cont'd)	<ul style="list-style-type: none"> Measurement sessions started by set_trigger run only during list execution. This applies even for sessions not terminated by set_trigger (<code>Period = 0</code>) or set_trigger4(<code>Period = 0</code>). Here, the measurement session automatically resumes when a new list starts (set_end_of_list does not reset the measurement session's status). During list execution, the status can be reset either by set_trigger (<code>Period = 0</code>) or stop_execution. If no list is active (here stop_execution is not executed), the status can be reset by stop_trigger. Sample values and status values returned by the scan system and stored on the RTC5 by set_trigger (<code>Signal1, Signal2 = 1-23, 25-30</code>) are always in the RTC5's 20-bit range, even if the DLL is set to RTC4 compatibility mode. The measured and stored values can be subsequently transferred to the PC by the get_waveform command for evaluation. We recommend explicitly ending the measurement session before reading the data. The measured values are transferred to the PC by get_waveform as 32-bit data and must be evaluated accordingly by users (see comments for get_value). The current status of a measurement session can be queried by the measurement_status command. For aborting a measurement session by set_trigger(<code>Period = 0</code>) or set_trigger4(<code>Period = 0</code>), the values of <code>Signal1</code> and <code>Signal2</code> are irrelevant as long as they are in the permitted value range. If you abort a measurement session with set_trigger(<code>Period = 0</code>) or set_trigger4(<code>Period = 0</code>), then previously recorded measurement values are <i>not</i> lost and the measurement pointer halts at its most recent value. This allows subsequent querying by measurement_status of the number of successful data entries. In contrast, if a measurement session is newly started with set_trigger and <code>Period > 0</code> or set_trigger4(<code>Period > 0</code>), the measurement pointer is reset and the measurements obtained thus far are overwritten. It is not possible to resume an explicitly or automatically halted measurement session. The type of scan system being used determines which status signals are generated and returned by the status channels. Specific information can be found in your scan system's operating manual. The control_command command can be used with iDRIVE scan systems (intelliSCAN, intelliSCAN_{de}, intelliDRILL, intellicube, intelliWELD, varioSCAN_{de}) to specify which information should be returned by the status channels. If the scan system has only one status channel, then only the X measurement signal contains meaningful data. With 3D scan systems, the output and status values of the Z axis are transmitted over the scan head connector's channel to which the Z axis is attached (if correspondingly configured by select_cor_table). Example: If the Z axis is attached to the secondary scan head connector's X channel (select_cor_table([1 or 2], 0)), then its status signal can be queried by StatusBX (<code>Signal1/2 = 4</code>).

Delayed Short List Command	set_trigger
Comments (cont'd)	<ul style="list-style-type: none"> The cartesian control values <code>Sample<X..Z></code> (<code>Signal1/2= 7...9</code>) take into account any wobble and Processing-on-the-fly corrections, but not image field corrections or coordinate transformations. The RTC5 calculates the "transformed" control values <code>Sample<AX..BZ>_Trans</code> (<code>Signal1/2= 25...30</code>) from the cartesian control values <code>Sample<X..Z></code> while taking into account any coordinate transformations defined by set_matrix, set_offset, set_scale or set_angle (or by the corresponding list commands) and any Z coordinate offset defined by set_offset_xyz, set_offset_xyz_list or any focal length offset defined by set_defocus or set_defocus_list. The RTC5 calculates the "corrected" control values <code>Sample<AX..BZ>_Corr</code> (<code>Signal1/2= 10...15</code>) from the transformed control values <code>Sample<X..Z>_Trans</code> while taking into account any loaded correction table. The effective output values <code>Sample<AX..BY>_Out</code> (<code>Signal1/2= 20...23</code>) transmitted by the RTC5 to the scan system additionally take into account any scanner offset and gain compensations for automatic self-calibration of the scan system (if activated by auto_cal or set_hi, see also page 224). The signals 12 and 15 as well as 27 and 30 are identical, each: <code>SampleAZ_Corr = SampleBZ_Corr</code> and <code>SampleAZ_Trans = SampleBZ_Trans</code>. The status signals <code>Status<...></code> lag the control signals <code>Sample<...></code> by a few clock cycles. Signal1, Signal2 = 0, 24 and 31...42 enables logging of values that were outputted during the previous clock cycle. During the current clock cycle, the corresponding signals may change after logging, either automatically or by control and list commands. Signal1, Signal2 = 24 enables logging of the signal parameter that is automatically adjusted during position- and/or speed-dependent or encoder-speed-dependent laser control (see set_auto_laser_control). Logging only occurs when automatic laser control has been activated and the laser is on. For switched-off lasers, 0 is logged. Signal1, Signal2 = 31 enables logging of the signal parameter specified for vector-defined laser control (see set_vector_control). Logging is also possible without executing para commands, but then the signal values remain unchanged. Pulse length and output period (Signal1, Signal2 = 37, 38) are only logged for "Laser active" laser control signals (not standby signals). In softstart mode (see page 152) and in pixel-output mode (see page 217), the pulse length and output period might possibly change faster than the 10 µs clock period (time resolution of logging by set_trigger). For Signal1, Signal2 = 39...42 see chapter 6.9.1 "Free Variables", page 102.



Delayed Short List Command	set_trigger
RTC4→ RTC5	<p>Unchanged basic functionality (measurement storage), however:</p> <ul style="list-style-type: none"> The measurement session can be aborted (unlike with the RTC4) without losing the previously acquired measurement values (see <i>Period = 0</i> above). The RTC5 has no StatusAZ and StatusBZ (RTC4: Signal1/2 = 3, 6) signals. Only either SampleAZ_Corr or SampleBZ_Corr (Signal1/2 = 12, 15) simultaneously supplies a meaningful signal (see also the above information on 3D systems). The Sample<X...Z> and Sample<AX...BZ>_Corr signals have a different meaning on the RTC5 than on the RTC4 (see information above). On the RTC5, <i>all</i> coordinate values are relative to the image field centerpoint 0 and lie within the range [-2¹⁹ ... (2¹⁹-1)]. For the RTC4, in contrast, the effective output values transmitted to the scan system are in the shifted range of [0 ... (2¹⁶-1)]. The latter also applies if the scan system is controlled by a RTC5 in conjunction with the XY2-100 converter (see also page 141).
Version info	Last change with version DLL 543, OUT 543, RBF 524: Signal1, Signal2 = 52...54.
References	get_waveform , measurement_status , control_command , get_value , get_values , set_mcbsp_out , set_mcbsp_out_ptr , set_trigger4



Delayed Short List Command	set_trigger4
Function	Starts measurement and storage of the specified signals (functions similarly to set_trigger , but with 4 instead of 2 simultaneous measurement signals).
Call	<code>set_trigger4(Period, Signal1, Signal2, Signal3, Signal4)</code>
Parameters	<p>Period See set_trigger.</p> <p>Signal1, See set_trigger.</p> <p>Signal2,</p> <p>Signal3,</p> <p>Signal4</p>
Comments	<ul style="list-style-type: none"> See comments for set_trigger. Unlike set_trigger, only 2^{19} entries per measurement channel are available with set_trigger4 (automatic termination upon the maximum 2^{19} data quadruplet). set_trigger(Period = 0) and set_trigger4(Period = 0) both terminate active data recording regardless of which command started the data recording. If set_trigger(Period > 0) or set_trigger4(Period > 0) is used to start a new measurement, then the measurement value counters are reset and existing data overwritten with new measurement values. See also related comments for set_trigger. Data channels 3 and 4 occupy the measurement memory's upper half, as do correction files $\text{No}=3$ and $\text{No}=4$. Therefore, both options cannot be used simultaneously. If set_trigger4 was preceded by loading correction file $\text{No}=3$ or $\text{No}=4$ with load_correction_file, then measurement values overwrite this area and the correction file is no longer available. If loading of correction file $\text{No}=3$ or $\text{No}=4$ succeeded after any data quadruplets were recorded with set_trigger4 (or more than 2^{19} data pairs with set_trigger), then the values for channels 3 and 4 (or beyond 2^{19} data pairs) are no longer available. load_program_file reestablishes the initial state of measurement memory. As with set_trigger, the get_waveform command can be used to transfer recorded values to the PC.
RTC4→ RTC5	New command.
Version info	Last change with version DLL 543, OUT 543, RBF 524: Signal1, Signal2 = 52...54.
References	set_trigger, stop_trigger, get_waveform



Undelayed Short List Command	set_vector_control	
Function	Initializes or deactivates vector-defined laser control (see page 166).	
Call	set_vector_control(Ctrl, Value)	
Parameters	Ctrl	<p>Control parameter (unsigned 32-bit value) for initializing or deactivating vector-defined laser control (for Ctrl = 1...6: identical with set_auto_laser_control).</p> <p>= 1...7: Defines which signal parameter is to be varied by vector-defined laser control.</p> <ul style="list-style-type: none"> = 1: 12-bit output value at the ANALOG OUT1 output port. = 2: 12-bit output value at the ANALOG OUT2 output port. = 3: Output value at the 8-bit digital output port. = 4: Pulse length (<code>PulseLength</code>) of the laser signals LASER1 and LASER2. = 5: Output period (<code>HalfPeriod</code>) of the laser signals LASER1 and LASER2. = 6: Output value at the 16-bit digital output. = 7: Focus shift. <p>= 0 or > 7: deactivates vector-defined laser control (for Ctrl > 7: get_last_error return code <code>RTC5_PARAM_ERROR</code>).</p>
	Value	<p>Defines the starting value for the parameter selected by Ctrl (unsigned 32-bit value). Allowed values:</p> <ul style="list-style-type: none"> For Ctrl = 1/2: 12-bit values [0 ... 4095]. For Ctrl = 3: 8-bit values [0 ... 255]. For Ctrl = 4: [0 ... (2³²-1)]. 1 bit equals 1/64 µs. For Ctrl = 5: [0 ... (2³²-1)]. 1 bit equals 1/64 µs. For Ctrl = 6: 16-bit values [0 ... (2¹⁶-1)]. For Ctrl = 7: [0 ... 65535], Value = focus shift + 32768.
Comments	<ul style="list-style-type: none"> • If Ctrl is an invalid value, then vector-defined laser control is deactivated (Ctrl = 0, initialization state). Otherwise, Value is applied as the starting value of the next para-command. If Value is invalid, then it is clipped to the maximum allowed value. • The command only affects para-mark and para-jump commands. • If position- and/or speed-dependent or encoder-speed-dependent laser control has been activated for the same control parameter (Ctrl = 1...6), then set_vector_control sets the 100% value of the position- and/or speed-dependent or encoder-speed-dependent laser control to Value. 	



Undelayed Short List Command	set_vector_control
Comments (cont'd)	<ul style="list-style-type: none"> If Ctrl = 7, then the focus shift for subsequent parameterized 2D and 3D mark and jump commands is linearly varied (as with set_defocus and set_defocus_list, see comments there) up to the specified end value (this requires enabling the 3D option as well as loading and assigning a 3D correction file). If, for the first parameterized mark or jump command, the currently set focus shift does not match the initial value (Value) specified by set_vector_control, then the parameterized command begins with a hard (z coordinate) jump to Value. Any settings made by set_defocus or set_defocus_list are lost. Unlike signed values in the range [-32768 ... 32767] for set_defocus and set_defocus_list, the focus shift (Value) specified for set_vector_control must be an unsigned number shifted upward by 32768: Value = focus shift + 32768. Automatic laser control should not be combined with the variable laser power of set_multi_mcbsp_in or the "freely definable wobble shape".
RTC4→ RTC5	New command. This command has no RTC4 compatibility mode for Value.
References	set_auto_laser_control



Ctrl Command	set_verify
Function	Activates or deactivates a download verification, see chapter 6.8.1 "Download Verification", page 99 .
Call	<code>OldVerify = set_verify(Verify)</code>
Parameter	<p><code>Verify</code> Parameter as an unsigned 32-bit value:</p> <ul style="list-style-type: none"> = 0: Verification is deactivated. > 0: Verification is activated.
Result	The <code>Verify</code> parameter that was active before calling the command (as an unsigned 32-bit value)
Comments	<ul style="list-style-type: none"> • If verification is activated, the download times are extended. • Complete download verification (as described in "Download Verification", page 99) requires driver version 1.0.4.0 or higher. Though older DLLs also function with the new driver, verification would then either be limited to checking of list command downloads (DLL version 511), or not functional at all (DLL version < 511). If a newer DLL (version > 511) is used in conjunction with an older driver version, then it returns error code 9 (DriverCall error) during load_program_file. • Verification of correction file downloads only works if the correction file contains a checksum (otherwise the error code <code>RTC5_VERIFY_ERROR</code> is generated). To ensure that a checksum is present (which is not the case for older ct5 correction files), you should test your correction file prior to loading by using the control command verify_checksum. If the correction file didn't contain a checksum, then this command creates and inserts a checksum into the file. • The commands <code>set_verify</code> and <code>n_set_verify</code> are also available without explicit access rights to a specific RTC5 Board. These commands only change settings in the DLL of the specified (or default) board. They have no effect on the board itself. • The board-specific error variables <code>LastError</code> and <code>AccError</code> (see "Error Handling", page 98) are neither generated nor altered by <code>set_verify</code>. • As of DLL 535, activation of the verify option by <code>set_verify</code> can generate the get_last_error return code <code>RTC5_OUT_OF_MEMORY</code> if a DLL-internal Windows memory request fails. In this case, the verify option remains deactivated.
RTC4→RTC5	New command.
Version info	<ul style="list-style-type: none"> • Available as of version DLL 511, OUT 510, RBF 504. • Change with version DLL 512: enhanced download verification. • Last change with version DLL 535 (see comment).
References	get_error , get_last_error , verify_checksum



Normal List Command	set_wait
Function	Sets a numbered wait marker (break point) in the list.
Call	<code>set_wait(WaitWord)</code>
Parameter	WaitWord number of the wait marker as an unsigned 32-bit value. Allowed range: [1 ... ($2^{32}-1$)]. 0 is corrected to 1.
Comments	<ul style="list-style-type: none"> Processing of a list is interrupted at each wait marker and remain halted until continued by release_wait. This provides a way to implement synchronizing. The “laser active” laser control signals are switched off by set_wait and a home jump defined by home_position or home_position_xyz might be executed (the INTERNAL-BUSY status is set while the home jump is executed). Continuation by execute_list_pos, restart_list or an external start is consequently suppressed. However, stop_execution or an external list stop is possible. release_wait, stop_execution or an external list stop removes suppression of the list start. set_wait sets the PAUSED status and resets the BUSY status (both queriable with get_status). The opposite occurs with a subsequent release_wait command (see also “List Execution Status”, page 77). If processing has been stopped at a wait marker, the command get_wait_status returns the number of this marker.
RTC4→ RTC5	Essentially unchanged functionality, however: The RTC5 also provides a PAUSED status. It is set with set_wait .
References	get_wait_status , release_wait , pause_list , stop_list



Delayed Short List Command	set_wobbel
Function	Defines the parameters for an ellipse-shaped wobble motion. "Wobbel" is a motion of the output position which is added to the regular marking movement (see page 189).
Call	<code>set_wobbel(Transversal, Longitudinal, Freq)</code>
Parameters	Transversal Amplitude of the elliptical or figure-8 movement <i>perpendicular</i> to the momentary direction of motion or to the one defined by set_wobbel_direction in <i>bits</i> (unsigned 32-bit value). Allowed range: [0 ... 131.071 (= $2^{17}-1$)]. Larger values are clipped.
	Longitudinal Amplitude of the elliptical or figure-8 movement <i>parallel</i> to the momentary direction of motion or to the one defined by set_wobbel_direction in <i>bits</i> (unsigned 32-bit value). Allowed range: [0 ... 131.071 (= $2^{17}-1$)]. Larger values are clipped.
	Freq Frequency of the wobble movement in <i>Hz</i> (number of ellipses per second, 64-bit IEEE floating point value). Allowed range: [-6000 ... 6000]. Larger values are clipped.
Comments	<ul style="list-style-type: none"> The wobble mode can be used for marking lines with various line widths. An ellipse-shaped movement is added to the linear marking movement, resulting in a spiral movement of the laser focus in the image field. Alternatively, a figure-of-8 wobble shape (horizontal or vertical to the direction of motion) can be activated by set_wobbel_mode. The line width can be set by appropriate values for the amplitude and frequency (frequency and marking speed should be compatible with each other). For arcs, too, the wobble motion follows the current marking direction (exception: see below). Therefore, independently of the cartesian angle, the effective line width is always the same. As of Version DLL 537, any desired "direction of motion" can also be specified by set_wobbel_direction. "freely definable wobble shapes" are likewise possible (see set_wobbel_vector). The wobble mode is terminated by setting both amplitudes and/or the frequency to zero (more accurate for $-n \leq \text{Freq} \leq n$ with $n = 50000/65536 = 0.7629\dots$). The frequency is signed. The wobble vector rotates clockwise for positive values and counter-clockwise for negative values. Thus, the inner and outer point densities of arcs can be individually set independently of their actual direction of rotation. At the beginning of a marking, (after set_wobbel or a jump), the wobble start point is generally set for the same value relative to the vector/arc startpoint; it is repeatedly continued, however, for polylines (including arcs). For identical amplitudes (Longitudinal = Transversal), the wobble startpoint is permanently referenced to the coordinate system, i.e. independent of the current direction of motion (as with the RTC4).

Delayed Short List Command	set_wobbel
Comments (cont'd)	<ul style="list-style-type: none"> For an angle, ellipse-shaped wobbel movements can result in more or less small jumps after reaching the corner, depending on the current wobbel phase. This does not occur (= "rounded corners") if the wobbel motion is exactly circular (<code>Longitudinal = Transversal</code>). <code>Longitudinal = 0</code> produces a sine-shaped wobbel motion across the direction of movement. When defining the wobbel shape and its frequency take the dynamics of the scan head and laser into account. Otherwise, an overheating and even a permanent damage of the system may occur, see chapter 8.4.1 "Wobbel Shapes – Important Notes on Choosing Appropriate Parameter Values". As of version DLL 543, OUT 543, RBF 524 the present wobbel amplitude can be recorded with trigger signal 53 (see set_trigger or set_trigger4). The format of the data is ((transversal << 16) + longitudinal).
RTC4 → RTC5	<p>The RTC4 only executed (elliptical) wobbel movements permanently referenced to the coordinate system. For the RTC5, the parameters <code>Transversal</code> and <code>Longitudinal</code> can be used to define an ellipse-shaped movement (the frequency's sign determines the direction of rotation) that follows the current direction of movement.</p> <p>The image field coordinates for the X and Y axes are specified as 20-bit values; in RTC4 compatibility mode they are specified as 16-bit values (as with the RTC4) and the RTC5 multiplies the specified value for the amplitudes <code>Longitudinal</code> and <code>Transversal</code> by 16 (the allowed range of values is correspondingly reduced to [1 ... 8191]).</p>
References	set_mark_speed , set_wobbel_mode



Undelayed Short List Command	set_wobbel_control
Function	Specifies laser control parameters for laser power variation with “freely definable wobbel shapes”.
Call	<code>set_wobbel_control(Ctrl, Value, MinValue, MaxValue)</code>
Parameter	<p>Ctrl control parameter for initializing or deactivating laser power variation as an unsigned 32-bit value: = 1...6: specifies which signal parameter to vary: For more information, see set_auto_laser_control. = 0 or > 6: deactivates laser power variation (for Ctrl > 6: get_last_error return code RTC5_PARAM_ERROR).</p> <p>Value Nominal laser power P0 (100%) as an unsigned 32-bit value. For the allowed range, see set_auto_laser_control; the maximum is [0 ... 65535] or 0xFFFFFFFF. Excessive values are clipped.</p> <p>MinValue, Limits that cannot be exceeded, as unsigned 32-bit values (see set_auto_laser_control). MaxValue The allowed ranges depend on the selected Ctrl parameter and on Value.</p>
Comments	<ul style="list-style-type: none"> Any still-pending delayed short list command executes first. For Ctrl = 0 and Ctrl > 6, laser power variation is switched off (initialization state after load_program_file). The values for McBSP/SPI multi transfers are then not used. The limits for Value, minValue, maxValue are the same as for automatic laser control (see set_auto_laser_control), but with a maximum of [0 ... 65535]. Initialized values are minValue = 0 and maxValue = 0xFFFFFFFF, i.e. no restrictions. The laser power can be combined with the vector-defined laser control, if the corresponding Ctrl parameter has been chosen identically. If you want variable laser power along a wobbel shape, then the command must execute before you activate the “freely definable wobbel shape”. Otherwise, laser power is not varied, even if you make a change in the data sets. Special case: if Value = 0xFFFFFFFF, then the nominal laser power is derived from the port assigned by the signal parameter Ctrl, instead of from the command. This is then the current content at this timepoint set by other normal commands, e.g. write_da_1_list for Ctrl = 1. These – and (should the situation arise) other queued delayed short list commands – are executed before the command set_wobbel_control. But beware: with execution of a “freely definable wobbel shape”, the value at the port can change at any time. Subsequent calls of set_wobbel_control then return other values for the nominal laser power.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 537, OUT 537.
References	set_wobbel_vector , set_multi_mcbsp_in , set_multi_mcbsp_in_list



Undelayed Short List Command	set_wobbel_direction
Function	Defines a direction vector for wobbel motions.
Call	set_wobbel_direction(dX, dY)
Parameter	dX, dY Components of the direction vector in bits as signed 32-bit values.
Comments	<ul style="list-style-type: none"> For non-zero direction vectors (dX and/or dY non-zero), wobbel motion (longitudinal and transversal) is relative to <i>this</i> direction vector instead of to the momentary direction vector. The direction vector's length is inconsequential. The RTC5 normalizes the direction vector. If dX = dY = 0, then the function is deactivated. The direction vector setting remains in effect even after the wobbel mode is switched off by set_wobbel or set_wobbel_mode, and continues being used if you switch the wobbel mode back on.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 537, OUT 537.
References	set_wobbel, set_wobbel_direction, set_wobbel_mode



Delayed Short List Command	set_wobbel_mode
Function	Switches the wobbel mode on and off and defines the parameters for an ellipse-shaped or figure-of-8 wobbel motion (see page 189).
Call	<code>set_wobbel_mode(Transversal, Longitudinal, Freq, Mode)</code>
Parameters	<p>Transversal Amplitude of the elliptical or figure-8 movement <i>perpendicular</i> to the momentary direction of motion or to the one defined by set_wobbel_direction in <i>bits</i> (unsigned 32-bit value). Allowed range: [0 ... 131.071 (=$2^{17}-1$)]. Excessive values are clipped.</p> <p>Longitudinal Amplitude of the elliptical or figure-8 movement <i>perpendicular</i> to the momentary direction of motion or to the one defined by set_wobbel_direction in <i>bits</i> (unsigned 32-bit value). Allowed range: [0 ... 131.071 (=$2^{17}-1$)]. Excessive values are clipped.</p> <p>Freq frequency of the wobbel movement in <i>Hz</i> (number of ellipses or figure-of-8s per second, 64-bit IEEE floating point value) Allowed range: [-6000 ... 6000]. Larger values are clipped.</p> <p>Mode Selection of the desired wobbel shape (signed 32-bit value): = 0: Ellipse-shaped wobbel movement < 0: Figure-of-8 wobbel movement perpendicular to the direction of motion (vertical 8) = 1: Figure-8 wobbel movement parallel to the direction of motion (horizontal figure-8) > 1: "Freely definable wobbel shape"</p>
Comments	<ul style="list-style-type: none"> For Mode = 0, the set_wobbel_mode command functions identically to the set_wobbel command (see comments there). If Mode \neq 0, then a figure-of-8 motion is activated. With figure-of-8s, broader mid-line processing can be achieved by appropriate parameter values. If the specified transverse and longitudinal amplitudes are identical, then the wobbel shape remains stationary in space; otherwise the orientation of the wobbel shape follows the current direction of motion. If the command executes while a list contains a "freely definable wobbel shape" (see set_wobbel_vector and chapter 8.4 "Wobbel Mode", page 189), then Mode > 1 selects this shape, otherwise the horizontal figure-8 shape is selected similarly to Mode = 1. If you subsequently define a wobbel shape and the wobbel mode was activated with the parameter Mode > 1, then a switch from the horizontal figure-8 to the wobbel shape automatically occurs. The wobbel mode is terminated by setting both amplitudes and/or the frequency to zero (more accurate for $-n \leq Freq \leq n$ with $n = 50000/65536 = 0.7629\dots$).



Delayed Short List Command	set_wobbel_mode
Comments (cont'd)	<ul style="list-style-type: none"> The frequency is signed. During the figure-of-8's first loop, the wobbel vector rotates clockwise for positive values and counter-clockwise for negative values. Thus (especially for ellipse-shaped wobbel motions), the inner and outer point densities of arcs can be individually set independently of their actual direction of rotation. At the beginning of a marking, (after set_wobbel or a jump), the wobbel start point is generally set for the same value relative to the vector/arc startpoint; it is repeatedly continued, however, for polylines (including arcs). For identical amplitudes (Longitudinal = Transversal), the wobbel startpoint is permanently referenced to the coordinate system, i.e. independent of the current direction of motion (as with the RTC4). For an angle, ellipse-shaped wobbel movements can result in more or less small jumps after reaching the corner, depending on the current wobbel phase. This does not occur (= "rounded corners") if the wobbel motion is exactly circular (Longitudinal = Transversal). Longitudinal = 0 produces a sine-shaped wobbel motion across the direction of movement. For "freely definable wobbel shapes", the parameter Freq has no meaning. It must nevertheless lie within the valid range, because otherwise the wobbel mode gets deactivated. This also applies to the parameters Transversal and Longitudinal. If the wobbel mode gets deactivated, then any already-defined wobbel shape remains active. It is used upon the next switch-on of the wobbel mode with Mode > 1. When defining the wobbel shape and its frequency take the dynamics of the scan head and laser into account. Otherwise, an overheating and even a permanent damage of the system may occur, see chapter 8.4.1 "Wobbel Shapes – Important Notes on Choosing Appropriate Parameter Values". As of version DLL 543, OUT 543, RBF 524 the present wobbel amplitude can be recorded with trigger signal 53 (see set_trigger or set_trigger4). The format of the data is ((transversal << 16) + longitudinal). The wobbel mode cannot be combined with: <ul style="list-style-type: none"> – Sky Writing – Pixel output mode – Jumps – laser_on_list
RTC4 → RTC5	<p>New command.</p> <p>In RTC4 compatibility mode the RTC5 multiplies the specified values for the amplitudes Longitudinal and Transversal by 16 (the allowed range of values is correspondingly reduced to [1 ... 8191]).</p>
Version info	Last change with version DLL 537, OUT 537: Modes > 0.
References	set_wobbel , set_wobbel_control , set_wobbel_vector



Delayed Short List Command	set_wobbel_offset
Function	Defines a wobbel shape shift in the direction of motion or perpendicular to the direction of motion.
Call	set_wobbel_offset(OffsetTrans, OffsetLong)
Parameter	OffsetTrans Transversal and longitudinal offset as signed 32-bit values. , OffsetLong Allowed range: ± 32767 . Larger values are clipped. Initialization values after load_program_file : (0,0).
Comments	<ul style="list-style-type: none"> Offsets can be defined for "classic" wobbel shapes (circle, ellipse, sine, figure-8) as well as for "freely definable wobbel shapes" (see chapter 8.4 "Wobbel Mode", page 189). The summed up wobbel amplitude including offsets may never exceed $\pm(2^{17}-1)$. At the beginning of the wobbel marking offsets are set as hard jumps. This applies also in case of switching-off or non-using the wobbel mode, for example, when using a jump command (analogously to "classical" wobbel shapes longitudinal amplitudes).
RTC4→ RTC5	New command. In RTC4 compatibility mode the RTC5 multiplies the specified values for offsets by 16. The allowed range of values is correspondingly reduced to ± 2047 .
Version info	Available as of version DLL 537, OUT 537.
References	set_wobbel_mode , set_wobbel_vector , set_wobbel_direction



Undelayed Short List Command	set_wobbel_vector
Function	Defines a (linear) section of a wobbel shape.
Call	<code>set_wobbel_vector(dTrans, dLong, Period, dPower)</code>
Parameter	<p>dTrans, dLong Microstep of a linear wobbel shape section <i>in bits</i> in 64-bit IEEE floating point format. Allowed value range [-256.0 ... +255.0]. dTrans is the microvector component perpendicular to the direction of motion (which is either the laser trajectory or the direction of motion defined by set_wobbel_direction). dLong is the microvector component in the direction of motion.</p> <p>Period Unsigned 32-bit value. Allowed range: [0 ... 65535]. = 1 ... 65535: number of microsteps = 0: the wobbel shape is switched off</p> <p>dPower Microstep of the relative laser power in 64-bit IEEE floating point format. Allowed range [-1.0 ... +1.0].</p> <p>Unallowed values are clipped to the corresponding limits.</p>
Comments	<ul style="list-style-type: none"> • <code>set_wobbel_vector</code> cannot be used together with the Softstart function because both require a shared memory area on the RTC5. To avoid (unintentional) mutual overwriting, you absolutely must explicitly deactivate Softstart. Otherwise, the command has no effect. <code>set_wobbel_vector</code> itself blocks subsequent memory changes by set_softstart_level or set_softstart_level_list. These commands have no effect as long as the wobbel shape has not been explicitly switched off. • Period = 0 is not allowed as a shape section. Period = 0 means explicitly switch off the “freely definable wobbel shape” (not the wobbel mode itself, see set_wobbel_mode). Subsequent calls of <code>set_wobbel_vector</code> begin a new wobbel shape. • Each call of <code>set_wobbel_vector</code> adds a new section at the end of the previous section independently of when the call is performed. • Up to 512 wobbel shape sections can be defined. After 512 sections, storage automatically wraps around to the first section and overwrites it. Each further call does the same to the next section. • The wobbel shape automatically begins with wobbel vector (0,0), i.e. directly on the marking itself (the hard jump of “classic” wobbel shapes is eliminated if the longitudinal amplitude ≠ 0) and also ends there without requiring explicit declaration. • Each wobbel shape section consists of a vector defined by microsteps and their number. The parameters dTrans, dLong get internally rounded to 7 bit decimal places. At runtime each microvector is executed within 10 µs. For large Period values, you should therefore take rounding error into account. Positive dTrans values cause that in respect to the direction of movement the start is to the right (which applies to circular wobbel shapes as well). Positive dLong values cause that in respect to the direction of movement the start is forward. The last section’s endpoint is also the first section’s start point. Independently of its respective position, it always ends at (0,0) and might get executed as a hard jump. With the last step the laser power is reset to the initial nominal laser power.



Undelayed Short List Command	set_wobbel_vector
Comments (cont'd)	<ul style="list-style-type: none"> The summed up wobbel amplitude may never exceed $\pm(2^{17}-1)$. If activated by set_wobbel_control, the RTC5 varies the current laser power P within the wobbel shape section as per $P = P_0 \times (1 + dPower \times n)$, whereby $0 \leq n \leq \text{Period}$. n represents the current number of each microstep. You can define the nominal laser power P0 with the list command set_wobbel_control. If activated by set_multi_mcbsp_in or set_multi_mcbsp_in_list, the nominal laser power P0 is multiplicatively varied by the laser power parameter P across multiple McBSP/SPI transfers: $P_{\text{McBSP}} = P_0 \times P / 16384$. Beware here that for each both corrections a maximum laser power of only $P_0 \times 4.0$ is allowed, whereby the maximum range of the laser control parameter likewise must not be exceeded (see set_wobbel_control). For laser power variation with Ctrl = 5 (half period), dPower needs to have the opposite sign. Unlike with automatic laser control, the multiplication factor cannot be inverted. By using an "empty" wobbel shape set_wobbel_vector(0.0, 0.0, 1, 0.0), you can also multiplicatively vary the nominal laser power P0 without needing to explicitly wobbel (for another alternative, see set_multi_mcbsp_in). When you switch off the wobbel shape (not by deactivation by set_wobbel_mode), the nominal laser power P0 is emitted at the port assigned by Ctrl if set_wobbel_control was last called with Value = 0xFFFFFFFF. Otherwise, the laser power P0 multiplied by the external factor from set_multi_mcbsp_in is emitted. If the wobbel shape gets switched off while the wobbel mode remains active, then the shape automatically switches to Mode = 1 (horizontal figure-8). The wobbel mode with "freely definable wobbel shapes" also needs to be switched on by set_wobbel_mode. Variable laser power for wobbel shapes cannot be combined with variable laser power for automatic or vector-based laser control (parameterized mark commands). Wobbel variation overwrites other variations if the respective Ctrl parameters are identical. Though unidentical Ctrl parameters are allowed, they serve no practical purpose. When defining "freely definable wobbel shapes" make sure that the microvectors of each individual wobbel vector does not exceed the maximum positioning speed significantly. Otherwise, a galvanometer scanner overheating may occur, see also chapter 8.4.1 "Wobbel Shapes – Important Notes on Choosing Appropriate Parameter Values"
RTC4 → RTC5	New command.
Version info	Available as of version DLL 537, OUT 537.
References	set_multi_mcbsp_in, set_multi_mcbsp_in_list, set_wobbel_control



Ctrl Command	simulate_encoder
Function	Activates or deactivates encoder simulation for the specified encoder.
Call	<code>simulate_encoder(EncoderNo)</code>
Parameter	<p>EncoderNo Encoder number as an unsigned 32-bit value. Allowed values:</p> <ul style="list-style-type: none"> = 1: ENCODER X pulses are simulated and encoder counter Encoder0 thereby incremented. = 2: ENCODER Y pulses are simulated and encoder counter Encoder1 thereby incremented. = 3: Pulses for ENCODER X and ENCODER Y are simulated. = 0: The encoder simulation is deactivated.
Comments	<ul style="list-style-type: none"> • The encoder simulation is driven by an internal 1 MHz clock (see also "Encoder Simulation" on page 247). • simulate_encoder does not trigger a reset of the encoder counter. • If <code>EncoderNo > 3</code>, then simulate_encoder is ignored (get_last_error return code <code>RTC5_PARAM_ERROR</code>).
RTC4→ RTC5	Unchanged.
References	get_encoder , store_encoder , read_encoder , wait_for_encoder , set_fly_x , set_fly_y , set_fly_rot



Normal List Command	simulate_ext_start
Function	After the specified track delay, causes a simulated external list start.
Call	<code>simulate_ext_start(Delay, EncoderNo)</code>
Parameters	<p>Delay track delay (in counter steps of the selected <code>EncoderNo</code> encoder counter) as a signed 32-bit value. Allowed range: $[-2^{31} \dots + (2^{31}-1)]$.</p> <p>EncoderNo Number of the to-be-used encoder counter as an unsigned 32-bit value. Allowed values: = 0: Encoder counter Encoder0 = 1: Encoder counter Encoder1</p>
Comments	<ul style="list-style-type: none"> For external list starts, see page 240. The track delay is specified in (relative) counting units of the selected encoder counter (the RTC5's encoder counters are triggered by an external or simulated encoder signal, see page 246). The list start is only triggered after the internal encoder counter was counted up (or down) by the specified track delay. External list starts initiated by <code>simulate_ext_start</code> or by an external start signal, but whose execution was postponed according to the specified track delay, are placed in a queue that accommodates up to 8 starts. <code>simulate_ext_start</code> cancels a previous queue and starts a new one. A start trigger initiated by <code>simulate_ext_start</code> or an external start signal only triggers a list start if it does not occur when the BUSY status is set (e.g. when outputting a list), when the INTERNAL-BUSY status is set (e.g. during <code>goto_xy</code>) or/and when the PAUSED status is set (after <code>pause_list</code>, <code>stop_list</code> or <code>set_wait</code>). Otherwise, bit#11 of the <code>get_startstop_info</code> return value is set. Therefore, if an unsuitable track delay is specified (e.g. <code>Delay = 0</code>), no list start is triggered. If <code>simulate_ext_start</code> is the first command in a list, then <code>get_startstop_info</code> can be used for checking whether processing of this list can finish within the defined track delay. Ensure that the sign of the track delay (<code>Delay</code> parameter) is appropriate for the selected encoder's counting direction (for external triggering, this corresponds to the work-piece's direction of motion). Track delays can also be set with <code>set_ext_start_delay</code> or <code>set_ext_start_delay_list</code>. Track delays are deactivated by initialization (with <code>load_program_file</code>), by external list stops and by <code>stop_execution</code>. They can also be deactivated with <code>set_control_mode</code> (bit#2). The <code>simulate_ext_start</code> command alone <i>does not</i> cause an encoder reset. But if accordingly set with <code>set_control_mode</code> (bit#9), a start trigger initiated by <code>simulate_ext_start</code>, <code>simulate_ext_start_ctrl</code> or by an external start signal causes an encoder reset if the start trigger is preceded by one of the Processing-on-the-fly commands <code>set_fly_x</code>, <code>set_fly_y</code> or <code>set_fly_rot</code>. If <code>EncoderNo > 1</code>, then <code>simulate_ext_start</code> is replaced with a <code>list_nop</code> (<code>get_last_error</code> return code <code>RTC5_PARAM_ERROR</code>).



Normal List Command	simulate_ext_start
RTC4→ RTC5	Unchanged functionality (except for the extended range of values). In RTC4 compatibility mode, the RTC5 multiplies the specified value for <code>Delay</code> by 16 (the allowed range of values is correspondingly reduced).
References	simulate_ext_start_ctrl , set_ext_start_delay , set_ext_start_delay_list , set_extstartpos , set_extstartpos_list , set_control_mode , simulate_ext_stop

Ctrl Command	simulate_ext_start_ctrl
Function	Causes a simulated external list start.
Call	<code>simulate_ext_start_ctrl()</code>
Comments	<ul style="list-style-type: none"> For external list starts, see page 240. If applicable, the start trigger for the list start occurs only after a track delay previously set by set_ext_start_delay, set_ext_start_delay_list or simulate_ext_start (i.e. after the internal encoder counter which was selected by these commands was counted up (or down) by the specified track delay). Track delays are deactivated by initialization (with load_program_file), by external list stops and by stop_execution. They can also be deactivated with set_control_mode (bit#2). External list starts initiated by simulate_ext_start_ctrl or by an external start signal, but whose execution was postponed according to the specified track delay, are placed in a queue that accommodates up to 8 starts. In contrast to simulate_ext_start, simulate_ext_start_ctrl does <i>not</i> cancel the previous queue. A start trigger initiated by simulate_ext_start_ctrl or by an external start signal does only trigger a list start if it does not coincide with the output of a list (otherwise, bit#11 of the get_startstop_info return value gets set). The simulate_ext_start_ctrl command alone <i>does not</i> cause an encoder reset. But if accordingly set with set_control_mode (bit#9), a start trigger initiated by simulate_ext_start_ctrl, simulate_ext_start or by an external start signal causes an encoder reset if the start trigger is preceded by one of the Processing-on-the-fly commands set_fly_x, set_fly_y or set_fly_rot. As of version DLL 543, OUT 543, simulate_ext_start_ctrl can be disabled by set_control_mode bit #4=1 or set_control_mode_list bit #4=1.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 515, OUT 514, RBF 512.
References	simulate_ext_start

Ctrl Command	simulate_ext_stop
Function	Causes a simulated external list stop.
Call	<code>simulate_ext_stop()</code>
Comments	<ul style="list-style-type: none"> For external list stops, see page 239. simulate_ext_stop simultaneously halts the master board and all slave boards, but stop_execution only halts the master board.
RTC4→ RTC5	New command.
References	simulate_ext_start , simulate_ext_start_ctrl , stop_execution



Ctrl Command	start_loop
Function	Starts a repeating automatic list change.
Call	<code>start_loop()</code>
Comments	<ul style="list-style-type: none"> A list change or new list start activated with start_loop repeats until execution is ended by calling the command quit_loop. The command start_loop can be issued at any desired time point, but only has effect upon the next set_end_of_list command. If the automatic list change is activated during processing of a list, then upon reaching set_end_of_list execution continues without delay at the other list. If there is only one list (<code>Mem2 = 0</code>, see config_list), then upon reaching set_end_of_list execution continues at <code>Pos = 0</code> (i.e. at the list's beginning). During processing of a list, the other list (and also the current list) can be newly loaded, see chapter 6.4.6 "Automatic List Changing", page 79. So that the start_loop command can function at all, the already active list must absolutely be finalized by set_end_of_list; the new list should already be loaded and the input pointer should be sufficiently ahead of the output pointer (otherwise, "old" commands are executed). If, during list execution, the end of the list is reached without encountering a set_end_of_list, then execution automatically continues at the beginning of the current list, not at the beginning of the other list. If, during processing of a list, the start_loop and auto_change_pos(Pos > 0) commands are called, then upon the next set_end_of_list the command auto_change_pos(Pos > 0) is executed; and at the next one the start_loop command is executed. The current list and list execution statuses can be queried by the commands read_status and get_status. The start_loop command triggers a flush of the list input buffer (see page 75).
RTC4→ RTC5	Unchanged.
References	quit_loop



Ctrl Command	stepper_abs
Function	Triggers set-position movements to the specified absolute set positions by both stepper motor outputs.
Call	<code>stepper_abs(Pos1, Pos2, WaitTime)</code>
Parameters	<p>Pos1, Absolute set positions in CLOCK pulse units for stepper motor outputs 1 and Pos2 2 as signed 32-bit values. Allowed range: $[-2^{31} \dots + (2^{31}-1)]$.</p> <p>WaitTime This parameter (an unsigned 32-bit value) determines when, at the latest, the function returns. <i>1 bit equals 1 s.</i> Allowed range: $[0 \dots +(2^{32}-1)]$.</p>
Comments	<ul style="list-style-type: none"> For programming the stepper motor signals, see chapter 9.1.5 "Stepper Motor Control", page 234. stepper_abs sets the new set-position values even if a previously started set-position movement is still in progress: <ul style="list-style-type: none"> If Pos1/Pos2 in the current direction of movement lies in front of the internal position variable's value, then the movement continues and the Busy status remains set. If Pos1/Pos2 equals the current value of the internal position variable, then the movement stops. The Busy status gets reset. If Pos1/Pos2 in the current direction of movement already lies past the internal position variable's value, then the corresponding stepper motor's direction of movement reverses (see note on page 234). The Busy status remains set. If no set-position movement is in progress, then one starts and the Busy status gets set. During performance of a reference movement (Init status set, see stepper_init), the command stepper_abs does not execute (get_last_error return code: RTC5_PARAM_ERROR). If the CLOCK pulse period was set to 0 by stepper_init, stepper_control or stepper_control_list, then no stepper motor movement occurs at the corresponding stepper motor output. If WaitTime = 0, then the command returns immediately so that system control is restored to the user program.
RTC4 → RTC5	New command.
Version info	<ul style="list-style-type: none"> Available as of version DLL 527, OUT 529, RBF 519. For older RTC5 Boards with DSP version numbers < 2 (get_RTC_version bits #16-23), the command is not executed (get_last_error return code: RTC5_TYPE_REJECTED).
References	stepper_abs_no , stepper_rel , stepper_rel_no , stepper_abs_list



Undelayed Short List Command	stepper_abs_list
Function	Same as stepper_abs , but a list command and without <code>WaitTime</code> parameter.
Call	<code>stepper_abs_list(Pos1, Pos2)</code>
Parameters	Pos1, See stepper_abs . Pos2
Comments	<ul style="list-style-type: none"> See stepper_abs. During performance of a reference movement (see stepper_init), execution of stepper_abs_list is delayed until the reference movement completes.
RTC4→RTC5	New command.
Version info	<ul style="list-style-type: none"> Available as of version DLL 527, OUT 529, RBF 519. For older RTC5 Boards with DSP version numbers < 2 (<code>get_RTC_version</code> bits #16-23), the command is neither executed nor replaced by list_nop (<code>get_last_error</code> return code: <code>RTC5_TYPE_REJECTED</code>).
References	stepper_abs

Ctrl Command	stepper_abs_no
Function	Triggers a set-position movement to the specified absolute set position by the desired stepper motor output.
Call	<code>stepper_abs_no(No, Pos, WaitTime)</code>
Parameters	<p>No Number of the stepper motor output as an unsigned 32-bit value. Allowed values: = 1: Stepper motor output 1. = 2: Stepper motor output 2. If the value is invalid, then the command is not executed (<code>get_last_error</code> return code: <code>RTC5_PARAM_ERROR</code>).</p> <p>Pos Absolute set position in CLOCK pulse units as a signed 32-bit value. Allowed range: $[-2^{31} \dots + (2^{31}-1)]$.</p> <p>WaitTime This parameter (an unsigned 32-bit value) determines when, at the latest, the function returns. $1 \text{ bit equals } 1 \text{ s}$. Allowed range: $[0 \dots + (2^{32}-1)]$.</p>
Comments	<ul style="list-style-type: none"> A set-position movement is only performed at the stepper motor output specified by <code>No</code>. Otherwise the command is identical to stepper_abs (see comments there).
RTC4→RTC5	New command.
Version info	<ul style="list-style-type: none"> See stepper_abs.
References	stepper_abs, stepper_abs_no_list



Undelayed Short List Command	stepper_abs_no_list
Function	Same as stepper_abs_no , but a list command and without <code>WaitTime</code> parameter.
Call	<code>stepper_abs_no_list(No, Pos)</code>
Parameters	<p>No Number of the stepper motor output as an unsigned 32-bit value. Allowed values: = 1: Stepper motor output 1. = 2: Stepper motor output 2. If the value is invalid, then the command is, already during loading, replaced by a list_nop (get_last_error return code <code>RTC5_PARAM_ERROR</code>).</p> <p>Pos See stepper_abs_no.</p>
Comments	<ul style="list-style-type: none"> See stepper_abs_no. During performance of a reference movement (see stepper_init), execution of stepper_abs_no_list is delayed until the reference movement completes.
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> Available as of version DLL 527, OUT 529, RBF 519. For older RTC5 Boards with DSP version numbers < 2 (get_RTC_version bits #16-23), the command is neither executed nor replaced by list_nop (get_last_error return code: <code>RTC5_TYPE_REJECTED</code>).
References	stepper_abs_no

Ctrl Command	stepper_control
Function	Sets the CLOCK signal pulse periods for stepper motor control.
Call	<code>stepper_control(Period1, Period2)</code>
Parameters	<p>Period1, Pulse periods of the CLOCK signals for stepper motor outputs 1 and 2 as signed Period2 32-bit values. 1 bit equals 10 µs. Allowed range: [0 ... +(2²⁴-1)] or < 0. Larger values are clipped.</p> <p>> 0: The new period waits for an already-running CLOCK pulse period at the corresponding stepper motor output before starting.</p> <p>= 0: An already-running CLOCK pulse period aborts at each stepper motor output (the corresponding stepper motor movement gets stopped, the Init- and/or Busy statuses for the respective stepper motor outputs get reset).</p> <p>< 0: the corresponding stepper motor control remains unchanged.</p>
Comments	<ul style="list-style-type: none"> For programming the stepper motor signals, see page 234. Period = 0 can be used as an emergency stop (see also the section "Terminating Infinite Movements", page 236).
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> Available as of version DLL 527, OUT 529, RBF 519. For older RTC5 Boards with DSP version numbers < 2 (get_RTC_version bits #16-23), the command is not executed (get_last_error return code: <code>RTC5_TYPE_REJECTED</code>).
References	stepper_control_list



Undelayed Short List Command	stepper_control_list
Function	Same as stepper_control , but a list command.
Call	<code>stepper_control_list(Period1, Period2)</code>
Parameters	Period1, See stepper_control . Period2
Comments	<ul style="list-style-type: none"> • See stepper_control.
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> • Available as of version DLL 527, OUT 529, RBF 519. • For older RTC5 Boards with DSP version numbers < 2 (get_RTC_version bits #16-23), the command is neither executed nor replaced by list_nop (get_last_error return code: RTC5_TYPE_REJECTED).
References	stepper_control

Ctrl Command	stepper_disable_switch
Function	Controls the usage of the stepper motor control SWITCH signals.
Call	<code>stepper_disable_switch(Disable1, Disable2)</code>
Parameters	<p>Disable1, Instruction how the SWITCH signals from stepper motor input 1 and 2 are to be used. As signed 32-bit values. Allowed range: $[-2^{32} \dots + (2^{32}-1)]$.</p> <p>> 0: The SWITCH signal at the respective stepper motor input is not used.</p> <p>= 0: The SWITCH signal at the respective stepper motor input is used.</p> <p>< 0: The use of the respective stepper motor input signal remains unchanged.</p>
Comments	<ul style="list-style-type: none"> • For programming the stepper motor signals, see chapter 9.1.5 "Stepper Motor Control", page 234. • The limit switch can be ignored during normal motions. For example, this may make sense for continuously rotating axes. • The SWITCH signals are always used with motions initiated by stepper_init.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 542, OUT 542.
References	stepper_init



Ctrl Command	stepper_enable
Function	Changes the stepper motor control's ENABLE signals.
Call	<code>stepper_enable(Enable1, Enable2)</code>
Parameters	<p>Enable1, ENABLE signals for stepper motor outputs 1and 2 as signed 32-bit values. Enable2 Allowed range: $[-2^{32} \dots +2^{32}-1]$. > 0: The ENABLE signal at the respective stepper motor output gets set. = 0: The ENABLE signal at the respective stepper motor output gets reset. < 0: The respective stepper motor output signal remains unchanged.</p>
Comments	<ul style="list-style-type: none"> For programming the stepper motor signals, see page 234.
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> Available as of version DLL 527, OUT 529, RBF 519. For older RTC5 Boards with DSP version numbers < 2 (get_RTC_version bits #16-23), the command is not executed (get_last_error return code: RTC5_TYPE_REJECTED).
References	stepper_enable_list

Undelayed Short List Command	stepper_enable_list
Function	Same as stepper_enable , but a list command and without a return value.
Call	<code>stepper_enable_list(Enable1, Enable2)</code>
Parameters	<p>Enable1, See stepper_enable. Enable2</p>
Comments	<ul style="list-style-type: none"> See stepper_enable.
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> Available as of version DLL 527, OUT 529, RBF 519. For older RTC5 Boards with DSP version numbers < 2 (get_RTC_version bits #16-23), the command is neither executed nor replaced by list_nop (get_last_error return code: RTC5_TYPE_REJECTED).
References	stepper_enable



Ctrl Command	stepper_init
Function	Initializes a stepper motor.
Call	<code>stepper_init(No, Period, Dir, Pos, Tol, Enable, WaitTime)</code>
Parameters	<p>No Number of the stepper motor output as an unsigned 32-bit value. Allowed values: = 1: Stepper motor output 1. = 2: Stepper motor output 2. If the value is invalid, then the command is not executed (get_last_error return code: RTC5_PARAM_ERROR).</p> <p>Period Pulse period of the CLOCK signal as an unsigned 32-bit value. <i>1 bit equals 10 µs.</i> Allowed range: [0 ... +(2²⁴-1)]. Larger values are clipped. If Period = 0, then no reference movement is performed and the function immediately returns (see comments).</p> <p>Dir Direction of stepper motor motion during the reference movement as a signed 32-bit value. Allowed range: [-2³¹ ... +(2³¹-1)]. > 0: The DIRECTION signal gets set; during the reference movement the internal position variable increments. = 0: The DIRECTION signal gets reset; during the reference movement the internal position variable decrements. < 0: The DIRECTION signal remains unchanged, no reference movement is performed and the function immediately returns.</p> <p>Pos New values for position variable and set position in CLOCK pulse units as a signed 32-bit value (see comments). Allowed range: [-2³¹ ... +(2³¹-1)]</p> <p>Tol Tolerance for the reference movement (see comments) as an unsigned 32-bit value. Allowed range: [1 ... +(2³²-1)]. If Dir < 0 and/or Period = 0, then the value Tol = 0 is irrelevant, otherwise Tol = 0 causes the command not to execute (get_last_error return code: RTC5_PARAM_ERROR).</p> <p>Enable ENABLE signal as an unsigned 32-bit value. Allowed range: [0 ... +(2³²-1)]. = 0: The ENABLE signal is reset. > 0: The ENABLE signal is set.</p> <p>WaitTime This parameter (an unsigned 32-bit value) defines when the command, at the latest, returns (see comments). <i>1 bit equals 1 s.</i> Allowed range: [0 ... +(2³²-1)]</p>

Ctrl Command	stepper_init
Comments	<ul style="list-style-type: none"> For programming the stepper motor signals, see chapter 9.1.5 "Stepper Motor Control", page 234. stepper_init immediately stops all previously started movements of the stepper motor specified by the <code>No</code> parameter. The <code>ENABLE</code> signal is merely forwarded and always correspondingly set, but has no effect on internal operations. If <code>Period > 0</code> and <code>Dir >= 0</code>, then stepper motor <code>No</code> starts a reference movement with the supplied <code>CLOCK</code> pulse period in the defined direction and the <code>Init</code> status gets set. The first Clock pulse is only generated after a full <code>CLOCK</code> pulse period. <ul style="list-style-type: none"> If a limit switch is activated right from the beginning, then the controller attempts to seek a position within the $\pm \text{Tol}$ range of the current position, initially opposite to the defined direction, with the limit switch deactivated. If this does not bring success, then the attempt terminates. In this case, the <code>SWITCH</code> status bit remains set (see get_steerer_status). If a limit switch gets activated during a movement, then the reference movement stops there. Afterward, the limit switch position is crossed multiple times to arrive at an averaged value for this position. Finally, the stepper motor is driven in the opposite direction by a normal set-position movement (<code>Init</code> status reset, <code>Busy</code> status set) within the tolerance value <code>Tol</code>. Here, the <code>DIRECTION</code> status signal changes, whereas it remains constant during seeking movements with multiple direction changes. Both the internal position variable (for the current position) and the internal set-position value get set to the value defined by the <code>Pos</code> parameter. Thus, this value represents a positional offset by <code>Tol</code> with respect to the defined position. If no limit switch is found (e.g. because no limit switch exists in the defined direction), then the stepper motor performs an infinite movement. The command then returns after <code>WaitTime</code> seconds to restore system control to the user program. But the stepper motor's infinite movement continues until it is aborted by a new stepper_init command or stepper_control (<code>Period = 0</code>). For more on this, see the section "Terminating Infinite Movements", page 236. If <code>Period = 0</code>, <code>Dir < 0</code> and/or <code>WaitTime = 0</code>, then the command returns straight away to immediately restore system control to the user program. You can then call get_steerer_status to check whether the reference movement has completed. <code>Period = 0</code> and/or <code>Dir < 0</code> can be used as an emergency stop. When necessary, a previously started stepper motor movement gets aborted, but no reference movement is performed. Here, too, the position variable and set-position value get set to the value <code>Pos</code>. This value then represents the current stepper motor position. The <code>DIRECTION</code> signal remains unchanged. If <code>Period = 0</code>, then the <code>INIT</code> and <code>BUSY</code> statuses get reset. No further clock pulses are outputted until <code>Period</code> is again set to a positive value (which then also affects subsequent set-position commands such as stepper_abs or stepper_rel).
RTC4→RTC5	New command.
Version info	<ul style="list-style-type: none"> Available as of version DLL 527, OUT 529, RBF 519. For older RTC5 Boards with DSP version numbers < 2 (get_RTC_version bits #16-23), the command is not executed (get_last_error return code: <code>RTC5_TYPE_REJECTED</code>).



Ctrl Command	stepper_rel
Function	Triggers set-position movements to the specified relative set positions by both stepper motor outputs.
Call	<code>stepper_rel(dPos1, dPos2, WaitTime)</code>
Parameters	<p>dPos1, Relative set positions in CLOCK pulse units for stepper motor outputs 1 and 2 dPos2 as signed 32-bit values. Allowed range: [-2³¹ ... +(2³¹-1)].</p>
Comments	<ul style="list-style-type: none"> The desired set positions should be specified relative to the current internal set-position values (the internal set-position values are correspondingly get newly set). Otherwise the command is identical to stepper_abs (see comments there).
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> See stepper_abs.
References	stepper_abs, stepper_rel_list

Undelayed Short List Command	stepper_rel_list
Function	Same as stepper_rel , but a list command and without <code>WaitTime</code> parameter.
Call	<code>stepper_rel_list(dPos1, dPos2)</code>
Parameters	<p>dPos1, See stepper_rel. dPos2</p>
Comments	<ul style="list-style-type: none"> See stepper_rel. During performance of a reference movement (see stepper_init), execution of stepper_rel_list is delayed until the reference movement completes.
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> Available as of version DLL 527, OUT 529, RBF 519. For older RTC5 Boards with DSP version numbers < 2 (<code>get_RTC_version</code> bits #16-23), the command is neither executed nor replaced by list_nop (<code>get_last_error</code> return code: <code>RTC5_TYPE_REJECTED</code>).
References	stepper_rel



Ctrl Command	stepper_rel_no
Function	Triggers set-position movements to the specified relative set positions by the desired stepper motor output.
Call	<code>stepper_rel_no(No, dPos, WaitTime)</code>
Parameters	<p>No See stepper_abs_no.</p> <p>dPos Relative set position in CLOCK pulse units as a signed 32-bit value. Allowed range: $[-2^{31} \dots + (2^{31}-1)]$.</p> <p>WaitTime See stepper_abs_no.</p>
Comments	<ul style="list-style-type: none"> The desired set positions should be specified relative to the current internal set-position values (the internal set-position values are correspondingly get newly set) and a set-position movement is only performed at the stepper motor output specified by <code>No</code>. Otherwise the command is identical to stepper_abs (see comments there).
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> See stepper_abs.
References	stepper_abs_no , stepper_abs , stepper_rel_no_list

Undelayed Short List Command	stepper_rel_no_list
Function	Same as stepper_rel_no , but a list command and without <code>WaitTime</code> parameter.
Call	<code>stepper_rel_no_list(No, dPos)</code>
Parameters	<p>No Number of the stepper motor output as an unsigned 32-bit value. Allowed values: = 1: Stepper motor output 1. = 2: Stepper motor output 2. If the value is invalid, then the command is, already during loading, replaced by a list_nop (get_last_error return code <code>RTC5_PARAM_ERROR</code>).</p> <p>dPos See stepper_rel_no.</p>
Comments	<ul style="list-style-type: none"> See stepper_rel_no. During performance of a reference movement (see stepper_init), execution of stepper_rel_no_list is delayed until the reference movement completes.
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> Available as of version DLL 527, OUT 529, RBF 519. For older RTC5 Boards with DSP version numbers < 2 (get_RTC_version bits #16-23), the command is neither executed nor replaced by list_nop (get_last_error return code: <code>RTC5_TYPE_REJECTED</code>).
References	stepper_rel_no



Normal List Command	stepper_wait
Function	Interrupts further execution of a list until a previously started (at the specified stepper motor output) stepper motor movement completes.
Call	<code>stepper_wait(No)</code>
Parameter	No Number of the stepper motor output as an unsigned 32-bit value. Allowed values: = 1: Stepper motor output 1. = 2: Stepper motor output 2. = 0, 3: Both stepper motor outputs. Only the two least-significant bits are evaluated.
Comments	<ul style="list-style-type: none"> For programming the stepper motor signals, see chapter 9.1.5 "Stepper Motor Control", page 234. If no stepper motor movement had been previously started at the specified stepper motor output, then the command still needs 10 µs to execute, even though it otherwise has no effect. <code>stepper_wait</code> influences neither the "laser active" laser control signals, nor the list status or the list execution status.
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> Available as of version DLL 527, OUT 529, RBF 519. For older RTC5 Boards with DSP version numbers < 2 (<code>get_RTC_version</code> bits #16-23), the command is neither executed nor replaced by <code>list_nop</code> (<code>get_last_error</code> return code: <code>RTC5_TYPE_REJECTED</code>).



Ctrl Command	stop_execution
Function	Stops execution of the list and deactivates the "laser active" laser control signals immediately.
Call	<code>stop_execution()</code>
Comments	<ul style="list-style-type: none"> With versions OUT 534 and lower, the command is only executed if a list is BUSY or PAUSED (else <code>get_last_error</code> return code: RTC5_BUSY). As of Version OUT 535, however, stop_execution deactivates the "laser active" laser control signals even if no list is active (here the command has no other effects; here too: <code>get_last_error</code> return code: RTC5_BUSY). With stop_execution, the mirrors stay in the current position, unless a home jump has been previously defined by home_position or home_position_xyz (a home jump is executed). Therefore, before a new list is loaded, the mirrors should be set to a defined position using the command goto_xy. The external START inputs are disabled (see "External List Start", page 240). The Processing-on-the-fly correction is turned off (Processing-on-the-fly option only). The BUSY list status values (see read_status) and the BUSY list execution status value (see get_status) are reset. A list that was interrupted with stop_execution cannot be resumed and must instead be newly started (e.g. by execute_list_pos). To only temporarily halt a list and later resume it, you can use the commands pause_list or stop_list. stop_execution only affects the addressed RTC5 Board. Even in a master/slave chain, it is not passed on to the downstream slave boards. If all boards of a master/slave chain shall be synchronously stopped, then simulate_ext_stop or an external stop signal must be applied to the master board (see also page 93).
RTC4→ RTC5	Unchanged.
Version info	Last change with version OUT 535 (see comment).
References	get_startstop_info

Ctrl Command	stop_list
Function	Pauses execution of the list and deactivates the "laser active" laser control signals.
Call	<code>stop_list()</code>
Comments	<ul style="list-style-type: none"> The command stop_list is synonymous with pause_list – see the comments there.
RTC4→ RTC5	Essentially unchanged functionality, however: The RTC5 also has a PAUSED status. It is set by stop_list .
References	pause_list



Ctrl Command	stop_trigger
Function	Resets (to <code>Busy = 0</code>) a measurement session's status that can be queried by measurement_status .
Call	<code>stop_trigger()</code>
Comments	<ul style="list-style-type: none"> This command is only needed if a measurement session was started by set_trigger or set_trigger4, but not subsequently terminated by set_trigger(Period = 0) or set_trigger4(Period = 0). Here, you can reset the measurement session's status even when no list is active (see set_trigger comments). The command is ignored (get_last_error return code: <code>RTC5_BUSY</code>) if the addressed board's BUSY status is currently set (list is being processed or has been halted by pause_list) or the board's INTERNAL-BUSY status is currently set. In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set).
RTC4→RTC5	New command.
Version info	Available as of version DLL 517, OUT 516, RBF 512.
References	measurement_status , set_trigger , set_trigger4

Undelayed Short List Command	store_encoder
Function	Stores the current counts of the two RTC5 encoder counters in a buffer on the RTC5.
Call	<code>store_encoder(Pos)</code>
Parameters	<code>Pos</code> Choice of buffer position as an unsigned 32-bit value: <ul style="list-style-type: none"> Bit#0 = 0: The counter values are stored in buffer position 0. Bit#0 = 1: The counter values are stored in buffer position 1.
Comments	<ul style="list-style-type: none"> <code>store_encoder</code> can be used, for instance, to determine the exact number of encoder increments occurring within a specific list-processing period. Here, you could begin the command list by storing the counts in buffer position 0 by <code>store_encoder(0)</code>. At the end of the command list, you could store the counts to buffer position 1 by <code>store_encoder(1)</code> and subsequently retrieve the stored values by read_encoder. The used buffer is reserved exclusively for storing encoder values. See also "Processing-on-the-fly (Optional)", page 199 and "Synchronization by Encoder Signals", page 246.
RTC4→RTC5	New command.
Version info	Available as of version DLL 520, OUT 519.
References	read_encoder , get_encoder , set_fly_x , set_fly_y , set_fly_rot , wait_for_encoder



Undelayed Short List Command	sub_call
Function	Causes an unconditional jump to an indexed subroutine.
Call	sub_call(Index)
Parameter	Index Index of the called indexed subroutine as an unsigned 32-bit value. Allowed range: [0 ... 1023].
Comments	<ul style="list-style-type: none"> The sub_call command reads the indexed subroutine's starting address from the internal management table based on the supplied index and then calls the command list_call (see also the comments there), which then triggers the jump to the subroutine. The sub_call command starts indexed subroutines in protected memory (that were loaded and/or referenced by load_sub, load_disk or copy_dst_src) as well as indexed subroutines in the unprotected list area (that were referenced by set_sub_pointer or copy_dst_src). If no subroutine is referenced for the supplied index, then the jump is suppressed and execution continues at the command located after the calling position. In some circumstances, a list_nop might be executed. <code>get_sub_pointer(Index)</code> can be used to determine whether a subroutine has been referenced for a particular index. If no subroutine has been referenced, this command returns the value “-1” (d.h. $2^{32}-1$). If <code>Index > 1023</code>, then sub_call is, already during loading, replaced by a list_nop (<code>get_last_error</code> return code <code>RTC5_PARAM_ERROR</code>). Absolute vector and arc commands execute absolutely after being called with sub_call. If the subroutine needs to execute at various locations within the image field, then either the subroutine can only contain relative mark, arc or jump commands or sub_call_abs must be used instead.
RTC4→ RTC5	New command.
References	list_call , sub_call_abs , sub_call_cond



Undelayed Short List Command	sub_call_abs
Function	Causes an unconditional jump to an indexed subroutine. In the called subroutine, any absolute vector and arc commands receive an offset (based on the current coordinates at the time of the call).
Call	sub_call_abs(Index)
Parameter	Index Index of the called indexed subroutine as an unsigned 32-bit value. Allowed range: [0 ... 1023].
Comments	<ul style="list-style-type: none"> The sub_call_abs command reads the indexed subroutine's starting address from the internal management table based on the supplied index and then calls the command list_call_abs (see also the comments there), which then triggers the jump to the subroutine. If the called subroutine contains no absolute commands, then there is no difference between sub_call_abs and sub_call.
RTC4→ RTC5	New command.
References	sub_call , sub_call_abs_cond

Undelayed Short List Command	sub_call_abs_cond						
Function	<p><i>Conditional call (AbsCall) of an indexed subroutine:</i> This command executes the command sub_call_abs(Index), if the current IOvalue at the 16-bit digital input port of the EXTENSION 1 socket connector meets the following condition:</p> $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } (((\text{not IOvalue}) \text{ AND Mask0}) = \text{Mask0})$ <p>(i.e. if the bits specified in Mask1 are 1 and the bits specified in Mask0 are 0). Otherwise, the directly following list command is immediately executed.</p>						
Call	sub_call_abs_cond(Mask1, Mask0, Index)						
Parameters	<table> <tr> <td>Mask1,</td> <td>16-bit masks as unsigned 32-bit values.</td> </tr> <tr> <td>Mask0</td> <td>Only the least significant 16 bits are evaluated.</td> </tr> <tr> <td>Index</td> <td>Index of the called indexed subroutine as an unsigned 32-bit value. Allowed range: [0 ... 1023].</td> </tr> </table>	Mask1,	16-bit masks as unsigned 32-bit values.	Mask0	Only the least significant 16 bits are evaluated.	Index	Index of the called indexed subroutine as an unsigned 32-bit value. Allowed range: [0 ... 1023].
Mask1,	16-bit masks as unsigned 32-bit values.						
Mask0	Only the least significant 16 bits are evaluated.						
Index	Index of the called indexed subroutine as an unsigned 32-bit value. Allowed range: [0 ... 1023].						
Comments	<ul style="list-style-type: none"> See sub_call_abs. See also "Conditional Command Execution", page 244. 						
RTC4→ RTC5	New command.						
References	sub_call_abs						



Undelayed Short List Command	sub_call_abs_repeat
Function	Causes an unconditional jump to an indexed subroutine and executes its body several times.
Call	sub_call_abs_repeat (Index, Number)
Parameters	Index Index of the to be called indexed subroutine (as with sub_call_abs). Number Number of repetitions as an unsigned 32-bit value.
Comments	<ul style="list-style-type: none"> The command <code>sub_call_abs(Index)</code> is synonymous with <code>sub_call_abs_repeat(Index, 1)</code>. See sub_call_repeat and sub_call_abs.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 538, OUT 538.
References	sub_call_repeat , sub_call_abs , sub_call

Undelayed Short List Command	sub_call_cond
Function	<i>Conditional call of an indexed subroutine:</i> This command executes the command sub_call (Index), if the current IOvalue at the 16-bit digital input port of the EXTENSION 1 socket connector meets the following condition: $((\text{IOvalue AND Mask1}) = \text{Mask1}) \text{ AND } ((\text{not IOvalue AND Mask0}) = \text{Mask0})$ (i.e. if the bits specified in Mask1 are 1 and the bits specified in Mask0 are 0). Otherwise, the directly following list command is immediately executed.
Call	sub_call_cond(Mask1, Mask0, Index)
Parameters	Mask1, 16-bit masks as unsigned 32-bit values. Mask0 Only the least significant 16 bits are evaluated. Index Index of the called indexed subroutine as an unsigned 32-bit value. Allowed range: [0 ... 1023].
Comments	<ul style="list-style-type: none"> See sub_call. See also chapter 9.3.2 "Conditional Command Execution", page 244.
RTC4→ RTC5	New command.
References	sub_call



Undelayed Short List Command	sub_call_repeat
Function	Causes an unconditional jump to an indexed subroutine and executes its body several times.
Call	<code>sub_call_repeat (Index, Number)</code>
Parameters	<p>Index Index of the to be called indexed subroutine (as with sub_call).</p> <p>Number Number of repetitions as an unsigned 32-bit value. Number = 0 is treated like Number = 1.</p>
Comments	<ul style="list-style-type: none"> The command <code>sub_call(Index)</code> is synonymous with <code>sub_call_repeat(Index, 1)</code>. The command sub_call_repeat avoids an empty cycle at the repetition, which otherwise inevitably occurs with <code>sub_call...sub_call</code> or <code>list_repeat...sub_call...list_until</code> constructions. With the command sub_call_repeat, for example, trajectories from micro vector commands for runup curves and coast down curves can be seamlessly joined together with shapes from subroutines.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 538, OUT 538.
References	sub_call_abs_repeat , sub_call , sub_call_abs , micro_vector_abs , micro_vector_abs_3d , micro_vector_rel , micro_vector_rel_3d



Undelayed Short List Command	switch_ioport
Function	Executes a relative list jump list_jump_rel (Pos) whose jump distance Pos (>1) is determined at runtime by the current value (IOvalue) at the 16-bit digital input port of the EXTENSION 1 socket connector. You can specify which of the 16-bit digital input port's bits should be evaluated for this purpose.
Call	<code>switch_ioport(MaskBits, ShiftBits)</code>
Parameters	MaskBits Number of contiguous bits (as an unsigned 32-bit value) of the 16-bit digital input port to be evaluated for determining the jump distance. Allowed range: [1 ... 16].
	ShiftBits Position of the least significant to-be-evaluated bit of the 16-bit digital input port (as an unsigned 32-bit value). Allowed range: [0 ... 15].
Comments	<ul style="list-style-type: none"> With invalid values of MaskBits or ShiftBits and with (MaskBits + ShiftBits) > 16, the command is replaced by a list_nop (get_last_error return code RTC5_PARAM_ERROR). The following applies: $\text{Mask} = ((1 << \text{MaskBits}) - 1) << \text{ShiftBits}$ and $\text{SwitchNo} = (\text{Mask} \& \text{IOvalue}) >> \text{ShiftBits}$. Here, a list_jump_rel(Pos) with Pos = (SwitchNo + 1) list positions are then executed. The jump distance is at least 1. This prevents infinite loops when no signal is present. Jumps to the same address (Pos = 0) are not possible with switch_ioport, but can be simulated by list_jump_rel (-1) as the directly subsequent command. The maximum jump distance is 2^{16} list positions. See also list_jump_rel. See also section "16-Bit Digital Input and Output", page 51 and chapter 9.3.2 "Conditional Command Execution", page 244.
RTC4→ RTC5	New command.
Version info	Last change with version DLL 517, OUT 516.
References	list_jump_rel , list_jump_rel_cond



Ctrl Command	sync_slaves
Function	Stably synchronizes (with the addressed RTC5 Board's 10 µs clock) all slave boards connected in a master/slave chain to the addressed RTC5 Board's SLAVE connector.
Call	<code>sync_slaves()</code>
Comments	<ul style="list-style-type: none"> For usage of this command, see chapter 6.6.3 "Master/Slave Operation", page 93. SCANLAB recommends executing synchronization immediately after all boards have been initialized by load_program_file and load_correction_file. Otherwise, all involved boards (i.e. the master board and the downstream slave boards allocated to the user program) should already have been halted prior to the call of sync_slaves. To avoid irregularities during execution of this command, you should neither apply external stop signals to the boards nor trigger external list starts (these do not get automatically suppressed). During execution of sync_slaves, a simulate_ext_stop command is executed for the addressed board. This halts the addressed board and all downstream slave boards in the master/slave chain (including boards not allocated to the user program). It is the responsibility of users to ensure that a running process is not disrupted by sync_slaves. After successful execution of the command, the scan system axes of all involved boards are in either the coordinate center position (0, 0 [,0]) or the HomeJump position (possibly shifted by an offset set by set_offset, set_defocus or set_hi). The command sync_slaves (relating to synchronization) only affects RTC5 Boards connected in a master/slave chain to the addressed RTC5 Board's MASTER connector. It does not affect the addressed board itself or any boards connected to the addressed board's SLAVE connector. Therefore, if all slave boards of a master/slave chain shall be synchronized with the master board, then sync_slaves must address the master board of the master/slave chain. If no board is connected to the addressed board's MASTER connector, then the command has no effect. Synchronization of downstream slave boards by sync_slaves occurs even when the boards' BUSY or INTERNAL-BUSY status is set (they are automatically halted). Nevertheless, the only slave boards to get synchronized are those allocated to the user program (allocation is not requested automatically). If the user program possesses access rights for the addressed board but no further boards, then the command has no effect. During execution of sync_slaves, all get_startstop_info error bits are cleared on all boards (including upstream boards) allocated to the user program. For each downstream slave board, the 10 µs clock is interrupted for a short term and newly synchronized. Therefore, data transmission between the RTC5 and scan system may be disrupted for up to 20 µs per downstream board. Such disturbances could induce hard galvanometer scanner jumps if the actual output values do not correspond to (0,0,[0]).



Ctrl Command	sync_slaves
Comments (cont'd)	<ul style="list-style-type: none"> The command is ignored (get_last_error return code: RTC5_BUSY) if the addressed board's BUSY status is currently set (list is being processed or has been halted by pause_list) or the board's INTERNAL-BUSY status is currently set. In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set).
RTC4→ RTC5	New command.
Version info	<ul style="list-style-type: none"> Available as of version DLL 515, OUT 514, RBF 512. Updated with version DLL 518, OUT 517, RBF 515 (enhanced master/slave synchronization, see page 692). It is recommended to use the command not till this version. Updated with version DLL 523, OUT 524: also triggers matching of short-command counts when necessary (see page 94).
References	get_master_slave , get_sync_status

Normal List Command	time_fix
Function	Stores the current time and date of the RTC clock/calender in a buffer for use with mark_date and mark_time .
Call	<code>time_fix()</code>
Comments	<ul style="list-style-type: none"> The command time_fix is synonymous with time_fix_f_off with FirstDay = 0 and Offset = 0 (see comments there).
RTC4→ RTC5	<p>Unchanged. The command was previously only available for the RTC SCANalone Board, i.e. the standalone version of the RTC4 Board.</p>
Version info	Last change with version OUT 517.
References	time_fix_f , time_fix_f_off

Normal List Command	time_fix_f
Function	Stores the current time and date of the RTC clock/calender in a buffer for use with mark_date and mark_time .
Call	<code>time_fix_f(FirstDay)</code>
Parameter	FirstDay See time_fix_f_off .
Comments	<ul style="list-style-type: none"> The command time_fix_f is synonymous with time_fix_f_off with Offset = 0 (see comments there).
RTC4→ RTC5	New command.
Version info	Last change with version OUT 517.
References	time_fix_f , time_fix_f_off , time_update , mark_time , mark_date



Normal List Command	time_fix_f_off
Function	Stores in a buffer the current time and date of the RTC clock/calender or a forward-dated date and time for use with mark_date and mark_time .
Call	<code>time_fix_f_off(FirstDay, Offset)</code>
Parameters	FirstDay This parameter (an unsigned 32-bit value) defines the starting number for determining the Julian calendar day from the current date of the RTC calendar: Counting proceeds from <code>FirstDay</code> to <code>FirstDay + 364</code> (+1 for leap years).
	Offset Forward dating in <i>seconds</i> (as an unsigned 32-bit value). Allowed range: [0 ... ($2^{32}-1$)].
Comments	<ul style="list-style-type: none"> Before calling time_fix_f_off, time_fix_f or time_fix, synchronization of the RTC5 and PC time should be performed (at least once after each load_program_file) by time_update. The complete time can be marked through multiple calls of mark_time and the complete date through multiple calls of mark_date. The commands time_fix_f_off, time_fix_f or time_fix must therefore be called <i>before</i> these marking commands so that the to-be-marked time or date do not change during marking. If time_fix_f_off, time_fix_f or time_fix are not called again before a time or date marking, then the last marked time is marked again. If time_fix_f_off, time_fix_f or time_fix are not called at all after a load_program_file, then a time of 00:00 or a date of January 1, 2000 is marked. If <code>Offset = 0</code>, then the current date and current time are fixed. One practical use of forward dating (<code>Offset > 0</code>) is for setting a date of expiry based on the current date. Backdating (<code>Offset < 0</code>) is not possible.
RTC4→RTC5	New command.
Version info	Last change with version DLL 524, OUT 526.
References	time_fix , time_fix_f , time_update , mark_time , mark_date



Ctrl Command	time_update
Function	Sets the RTC5's 24-hour clock and calendar to the current PC time.
Call	time_update()
Comments	<ul style="list-style-type: none"> The RTC5 Board clock does not contain a battery backup. Therefore, the time_update command must be called after each load_program_file if the 24-hour clock and calendar of the RTC5 are to be calibrated. The base value for internal time counting is set to January 1, 2000, 00:00 by load_program_file whereas to the PC time by time_update. An internal seconds counter is set to 0 by load_program_file or by time_update, but is always driven by the quartz-controlled 10 µs clock. Before marking with mark_date or mark_time, you must call time_fix, time_fix_f or time_fix_f_off so that the current time can be captured (as sum of the base value and the current value of the internal seconds counter) and formatted.
RTC4→RTC5	<p>Unchanged. The command was previously only available for the RTC SCANalone Board, i.e. the standalone version of the RTC4 Board.</p>
References	time_fix , time_fix_f , time_fix_f_off , mark_date , mark_time



Normal List Command	timed_arc_abs
Function	Moves the laser focus for the specified marking duration from the current position along an arc with the specified angle and center point (absolute coordinate values) within a two-dimensional image field.
Call	<code>timed_arc_abs(X, Y, Angle, T)</code>
Parameters	<p>X, Y Absolute coordinates of the arc center in <i>bits</i> as signed 32-bit value. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.</p> <p>Angle Arc angle in ° as a 64-bit IEEE floating point value. Positive angle values correspond to clockwise angles. Allowed range: [-3600.0° ... +3600.0°] (± 10 full circles). Out-of-range values are edge-clipped.</p> <p>T Duration of the complete arc marking process in <i>microseconds</i> (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If $T < 5$, then timed_arc_abs behaves like arc_abs.</p>
Comments	<ul style="list-style-type: none"> Unlike arc_abs, the timed_arc_abs command does not execute the marking process with the specified (by set_mark_speed or set_mark_speed_ctrl) marking speed. Instead, the speed (i.e. the number of microsteps) is adjusted so that the arc lasts as long as specified (see chapter 8.10 "Timed Vector and Arc Commands", page 223). The total marking time is (for $T \geq 5$) the sum of the specified (rounded) time and the set delays. The “laser active” laser control signals are automatically turned on at the beginning of the command (or remain on after a directly preceding mark or arc command). The defined scanner and laser delays are thereby taken into account (see chapter 7.2 “Delay Settings for Synchronizing Scan Head and Laser Control”, page 111). Note that other delays are executed in Sky writing mode (see page 127). Exception: zero-length arc commands (see notes on page 115). During the conversion of specified coordinate values into scan system output values, any previously defined coordinate transformations, assigned correction tables etc. are taken into account after successful microvectorization (see page 141). The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified arc center coordinate values by 16 (the allowed range of values is correspondingly reduced to [-524288 ... 524287]).
References	arc_abs , timed_arc_rel



Normal List Command	timed_arc_rel
Function	Moves the laser focus for the specified marking duration from the current position along an arc with the specified angle and center point (relative coordinate values) within a two-dimensional image field.
Call	<code>timed_arc_rel(dX, dY, Angle, T)</code>
Parameters	<p>dX, dY <i>Relative coordinates of the arc center in bits as signed 32-bit value. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.</i></p> <p>Angle <i>Arc angle in ° as a 64-bit IEEE floating point value. Positive angle values correspond to clockwise angles. Allowed range: [-3600.0° ... +3600.0°] (± 10 full circles). Out-of-range values are edge-clipped.</i></p> <p>T <i>Duration of the complete arc marking process in microseconds (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If $T < 5$, then timed_arc_rel behaves like arc_rel.</i></p>
Comments	<ul style="list-style-type: none"> The coordinates for the arc center are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to timed_arc_abs (see the comments there, exception: $T < 5$, see above). The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified arc center coordinate values by 16 (the allowed range of values is correspondingly reduced to [-524288 ... 524287]).
References	timed_arc_abs, arc_rel



Normal List Command	timed_jump_abs
Function	Moves the output point (of the laser focus) for the specified jump duration along a 2D vector from the current position to the specified position (absolute coordinate values) within a two-dimensional image field.
Call	<code>timed_jump_abs(X, Y, T)</code>
Parameters	<p>X, Y Absolute coordinates of the jump vector end point in <i>bits</i> as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</p> <p>T Duration of the complete jump vector in <i>microseconds</i> (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If $T < 5$, then timed_jump_abs behaves like jump_abs.</p>
Comments	<ul style="list-style-type: none"> Unlike jump_abs, the timed_jump_abs command does not execute the jump with the specified (by set_jump_speed or set_jump_speed_ctrl) jump speed. Instead, the speed (i.e. the number of microsteps) is adjusted so that the vector lasts as long as specified (see chapter 8.10 "Timed Vector and Arc Commands", page 223). The total jump time is (for $T \geq 5$) the sum of the specified (rounded) time and the set delays. The "laser active" laser control signals are off during the jump (if necessary, they are switched off before the jump). After a jump command, a (variable) jump delay is inserted. Exception: a zero-length jump vector's subsequent (variable) jump delay is not executed. However, the command itself still requires a 10 µs clock cycle for execution. During the conversion of specified coordinate values into scan system output values, any previously defined coordinate transformations, assigned correction tables etc. are taken into account after successful microvectorization (see page 141).
RTC4 → RTC5	<p>Unchanged functionality (except for the extended range of values, and $T < 5$ does not execute timed_jump_abs, but instead jump_abs).</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).</p>
References	jump_abs , timed_jump_rel , timed_jump_abs_3d



Normal List Command	timed_jump_abs_3d
Function	Moves the output point (of the laser focus) for the specified jump duration along a 3D vector from the current position to the specified position (absolute coordinate values) within a three-dimensional process volume.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as timed_jump_abs . However, microvectorization is calculated like a 3D command and hence influences the effective jump speed in the XY plane.
Call	<code>timed_jump_abs_3d(X, Y, Z, T)</code>
Parameters	<p>X, Y, Z Absolute coordinates of the jump vector end point in <i>bits</i> as signed 32-bit values. Allowed range:</p> <ul style="list-style-type: none"> For X and Y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped. <p>T Duration of the complete jump vector in <i>microseconds</i> (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If T < 5, then timed_jump_abs_3d behaves like jump_abs_3d.</p>
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, this command functions similarly to the timed_jump_abs command (see the comments there, exception: T < 5, see above). The X and Y axes can be controlled with 20-bit resolution, the Z axis with 16-bit resolution. The RTC5 (in RTC5 mode as well as in RTC4 compatibility mode) automatically upscales Z coordinate values internally to 20-bit values, so that the three dimensions of space can be handled equivalently in terms of the supplied jump speed value. The DirectMove3D parameter of the set_delay_mode command determines the type of Z-axis motion (linear or with stepwise correction). During calculation of the Z output value to the scan system, any previously defined offset to the Z coordinate and focal length is taken into account (see page 141).
RTC4 → RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 mode and the RTC4 compatibility mode.
References	timed_jump_abs , jump_abs_3d , timed_jump_rel_3d



Normal List Command	timed_jump_rel
Function	Moves the output point (of the laser focus) for the specified jump duration along a 2D vector from the current position to the specified position (relative coordinate values) within a two-dimensional image field.
Call	<code>timed_jump_rel(dX, dY, T)</code>
Parameters	<p>dX, dY <i>Relative coordinates of the jump vector end point in bits as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.</i> <i>The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</i></p> <p>T <i>Duration of the complete jump vector in microseconds (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If T < 5, then timed_jump_rel behaves like jump_rel.</i></p>
Comments	<ul style="list-style-type: none"> The coordinates for the jump vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to timed_jump_abs (see the comments there, exception: T < 5, see above).
RTC4→ RTC5	<p>Unchanged functionality (except for the extended range of values, and T < 5 does not execute timed_jump_rel, but instead jump_rel). In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).</p>
References	timed_jump_abs, jump_rel, timed_jump_rel_3d



Normal List Command	timed_jump_rel_3d
Function	Moves the output point (of the laser focus) for the specified jump duration along a 3D vector from the current position to the specified position (relative coordinate values) within a three-dimensional process volume.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as timed_jump_rel . However, microvectorization is calculated like a 3D command and hence influences the effective jump speed in the XY plane.
Call	<code>timed_jump_rel_3d(dx, dy, dz, T)</code>
Parameters	<p><code>dx, dy, dz</code> <i>Relative coordinates of the jump vector end point in bits as signed 32-bit values. Allowed range:</i></p> <ul style="list-style-type: none"> • For X and Y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. • For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped. <p><code>T</code> <i>Duration of the complete jump vector in microseconds (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If T < 5, then <code>timed_jump_rel_3d</code> behaves like <code>jump_rel_3d</code>.</i></p>
Comments	<ul style="list-style-type: none"> • The coordinates for the jump vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to timed_jump_abs_3d (see the comments there, exception: T < 5, see above).
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 mode and the RTC4 compatibility mode.
References	timed_jump_abs_3d , jump_rel_3d , timed_jump_rel



Normal List Command	timed_mark_abs
Function	Moves the laser focus for the specified marking duration along a 2D vector from the current position to the specified position (absolute coordinate values) within a two-dimensional image field.
Call	<code>timed_mark_abs(X, Y, T)</code>
Parameters	<p>X, Y Absolute coordinates of the mark vector end point in <i>bits</i> as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</p> <p>T Duration of the complete mark vector in <i>microseconds</i> (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If $T < 5$, then timed_mark_abs behaves like mark_abs.</p>
Comments	<ul style="list-style-type: none"> Unlike mark_abs, the timed_mark_abs command does not execute the marking process with the specified (by set_mark_speed or set_mark_speed_ctrl) marking speed. Instead, the speed (i.e. the number of microsteps) is adjusted so that the vector lasts as long as specified (see chapter 8.10 "Timed Vector and Arc Commands", page 223). The total marking time is (for $T \geq 5$) the sum of the specified (rounded) time and the set delays. The “laser active” laser control signals are automatically turned on at the beginning of the command (or remain on after a directly preceding mark or arc command). The defined scanner and laser delays are thereby taken into account (see chapter 7.2 “Delay Settings for Synchronizing Scan Head and Laser Control”, page 111). Note that other delays are executed in Sky writing mode (see page 127). Exception: zero-length mark commands (see notes on page 115). During the conversion of specified coordinate values into scan system output values, any previously defined coordinate transformations, assigned correction tables etc. are taken into account after successful microvectorization (see page 141).
RTC4→ RTC5	Unchanged functionality (except for the extended range of values, and $T < 5$ does not execute timed_mark_abs , but instead mark_abs). In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).
References	mark_abs , timed_mark_rel , timed_mark_abs_3d



Normal List Command	timed_mark_abs_3d
Function	Moves the laser focus for the specified marking duration along a 3D vector from the current position to the specified position (absolute coordinate values) within a three-dimensional process volume.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as timed_mark_abs . However, microvectorization is calculated like a 3D command and hence influences the effective marking speed in the XY plane.
Call	<code>timed_mark_abs_3d(X, Y, Z, T)</code>
Parameters	<p>X, Y, Z Absolute coordinates of the mark vector end point in <i>bits</i> as signed 32-bit values. Allowed range:</p> <ul style="list-style-type: none"> For X and Y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped. <p>T Duration of the complete mark vector in <i>microseconds</i> (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If T < 5, then timed_mark_abs_3d behaves like mark_abs_3d.</p>
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, this command functions similarly to the timed_mark_abs command (see the comments there, exception: T < 5, see above). The X and Y axes can be controlled with 20-bit resolution, the Z axis with 16-bit resolution. The RTC5 (in RTC5 mode as well as in RTC4 compatibility mode) automatically upscales Z coordinate values internally to 20-bit values, so that the three dimensions of space can be handled equivalently in terms of the supplied jump speed value. During calculation of the Z output value to the scan system, any previously defined offset to the Z coordinate and focal length is taken into account (see page 141).
RTC4 → RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 mode and the RTC4 compatibility mode.
References	timed_mark_abs , mark_abs_3d , timed_mark_rel_3d



Normal List Command	timed_mark_rel
Function	Moves the laser focus for the specified marking duration along a 2D vector from the current position to the specified position (relative coordinate values) within a two-dimensional image field.
Call	<code>timed_mark_rel(dx, dy, T)</code>
Parameters	<p><code>dx, dy</code> <i>Relative coordinates of the mark vector end point in bits as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.</i> <i>The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</i></p> <p><code>T</code> <i>Duration of the complete mark vector in microseconds (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If $T < 5$, then timed_mark_rel behaves like mark_rel.</i></p>
Comments	<ul style="list-style-type: none"> The coordinates for the mark vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to timed_mark_abs (see the comments there, exception: $T < 5$, see above).
RTC4→ RTC5	<p>Unchanged functionality (except for the extended range of values, and $T < 5$ does not execute timed_mark_rel, but instead mark_rel).</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).</p>
References	timed_mark_abs, mark_rel, timed_mark_rel_3d



Normal List Command	timed_mark_rel_3d
Function	Moves the laser focus for the specified marking duration along a 3D vector from the current position to the specified position (relative coordinate values) within a three-dimensional process volume.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as timed_mark_rel . However, microvectorization is calculated like a 3D command and hence influences the effective marking speed in the XY plane.
Call	<code>timed_mark_rel_3d(dx, dy, dz, T)</code>
Parameters	<p><code>dx, dy, dz</code> Relative coordinates of the mark vector end point in <i>bits</i> as signed 32-bit values. Allowed range:</p> <ul style="list-style-type: none"> • For X and Y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. • For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped. <p><code>T</code> Duration of the complete mark vector in <i>microseconds</i> (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If <code>T < 5</code>, then timed_mark_rel_3d behaves like mark_rel_3d.</p>
Comments	<ul style="list-style-type: none"> • The coordinates for the mark vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to timed_mark_abs_3d (see the comments there, exception: <code>T < 5</code>, see above).
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 mode and the RTC4 compatibility mode.
References	timed_mark_abs_3d, timed_mark_abs, mark_rel_3d, timed_mark_rel



Normal List Command	timed_para_jump_abs
Function	Moves the output point (of the laser focus) for the specified jump duration along a 2D vector from the current position to the specified position (absolute coordinate values) within a two-dimensional image field and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Call	timed_para_jump_abs(X, Y, P, T)
Parameters	<p>X, Y Absolute coordinates of the jump vector end point in <i>bits</i> as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</p> <p>P End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (<i>Ctrl</i> parameter), identical with set_vector_control (<i>Value</i> parameter)</p> <p>T Duration of the complete jump vector in <i>microseconds</i> (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If $T < 5$, then timed_para_jump_abs behaves like para_jump_abs.</p>
Comments	<ul style="list-style-type: none"> Unlike para_jump_abs, the timed_para_jump_abs command does not execute the jump with the specified (by set_jump_speed or set_jump_speed_ctrl) jump speed. Instead, the speed (i.e. the number of microsteps) is adjusted so that the vector lasts as long as specified (see Chapter 8.10 "Timed Vector and Arc Commands", page 223). The total jump time is (for $T \geq 5$) the sum of the specified (rounded) time and the set delays. timed_para_jump_abs requires two list entries. During runtime, the command's part on the first list entry is executed as a short list command and afterward the second part is executed as a normal list command. Thereby, both command parts are executed within the same 10 µs clock, unless further (previous) short list commands induce a list_nop between the two parts. If $T < 5$, then timed_para_jump_abs behaves like para_jump_abs. Then it requires only one list entry. The "laser active" laser control signals are off during the jump (if necessary, they are switched off before the jump). After a jump command, a (variable) jump delay is inserted. Exception: a zero-length jump vector's subsequent (variable) jump delay is not executed. However, the command itself still requires a 10 µs clock cycle for execution. During the conversion of specified coordinate values into scan system output values, any previously defined coordinate transformations, assigned correction tables etc. are taken into account after successful microvectorization (see page 141).



Normal List Command	timed_para_jump_abs
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).
Version info	Available as of version DLL 517, OUT 516, RBF 512.
References	para_jump_abs , timed_jump_abs , jump_abs , timed_para_jump_rel , timed_para_jump_abs_3d

List Multi-Command	timed_para_jump_abs_3d
Function	Moves the output point (of the laser focus) for the specified jump duration along a 3D vector from the current position to the specified position (absolute coordinate values) within a three-dimensional process volume and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as timed_para_jump_abs . However, microvectorization is calculated like a 3D command and hence influences the effective jump speed in the XY plane.
Call	timed_para_jump_abs_3d(X, Y, Z, P, T)
Parameters	<p>X, Y, Z Absolute coordinates of the jump vector end point in <i>bits</i> as signed 32-bit values. Allowed range:</p> <ul style="list-style-type: none"> For X and Y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped. <p>P End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (<i>Ctrl</i> parameter), identical with set_vector_control (<i>Value</i> parameter)</p> <p>T Duration of the complete jump vector in <i>microseconds</i> (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If T < 5, then timed_para_jump_abs_3d behaves like para_jump_abs_3d.</p>
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, this command functions similarly to the timed_para_jump_abs command (see the comments there, exception: T < 5, see above). The X and Y axes can be controlled with 20-bit resolution, the Z axis with 16-bit resolution. The RTC5 (in RTC5 mode as well as in RTC4 compatibility mode) automatically upscales Z coordinate values internally to 20-bit values, so that the three dimensions of space can be handled equivalently in terms of the supplied jump speed value. The DirectMove3D parameter of the set_delay_mode command determines the type of Z-axis motion (linear or with stepwise correction). During calculation of the Z output value to the scan system, any previously defined offset to the Z coordinate and focal length is taken into account (see page 141). For T ≥ 5 the command occupies two list buffer positions. The initial component executes as an undelayed short list command before the primary component (a normal list command).



List Multi-Command	timed_para_jump_abs_3d
RTC4→ RTC5	<p>New command.</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 mode and the RTC4 compatibility mode.</p>
Version info	Available as of version DLL 517, OUT 516, RBF 512.
References	timed_para_jump_abs , para_jump_abs_3d , jump_abs_3d , timed_para_jump_rel_3d

Normal List Command	timed_para_jump_rel
Function	Moves the output point (of the laser focus) for the specified jump duration along a 2D vector from the current position to the specified position (relative coordinate values) within a two-dimensional image field and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Call	<code>timed_para_jump_rel(dx, dy, P, T)</code>
Parameters	<p>dx, dy Relative coordinates of the jump vector end point in <i>bits</i> as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</p> <p>P End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (<i>Ctrl</i> parameter), identical with set_vector_control (<i>Value</i> parameter)</p> <p>T Duration of the complete jump vector in <i>microseconds</i> (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If $T < 5$, then timed_para_jump_rel behaves like para_jump_rel.</p>
Comments	<ul style="list-style-type: none"> The coordinates for the jump vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to timed_para_jump_abs (see the comments there, exception: $T < 5$, see above).
RTC4→ RTC5	<p>New command.</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).</p>
Version info	<ul style="list-style-type: none"> Available as of version DLL 517, OUT 516, RBF 512. Last change with version DLL 518.
References	timed_para_jump_abs , para_jump_rel , timed_jump_rel , timed_para_jump_rel_3d



List Multi-Command	timed_para_jump_rel_3d
Function	Moves the output point (of the laser focus) for the specified jump duration along a 3D vector from the current position to the specified position (relative coordinate values) within a three-dimensional process volume and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as timed_para_jump_rel . However, microvectorization is calculated like a 3D command and hence influences the effective jump speed in the XY plane.
Call	<code>timed_jump_rel_3d(dx, dy, dz, P, T)</code>
Parameters	<p>dx, dy, dz Relative coordinates of the jump vector end point in <i>bits</i> as signed 32-bit values. Allowed range:</p> <ul style="list-style-type: none"> For x and y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. For z: [-32768 ... 32767]. Out-of-range values are edge-clipped. <p>P End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (<i>Ctrl</i> parameter), identical with set_vector_control (<i>Value</i> parameter)</p> <p>T Duration of the complete jump vector in <i>microseconds</i> (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If T < 5, then timed_para_jump_rel_3d behaves like para_jump_rel_3d.</p>
Comments	<ul style="list-style-type: none"> The coordinates for the jump vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to timed_para_jump_abs_3d (see the comments there). For T ≥ 5 the command occupies two list buffer positions. The initial component executes as an undelayed short list command before the primary component (a normal list command).
RTC4→ RTC5	<p>New command.</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 mode and the RTC4 compatibility mode.</p>
Version info	<ul style="list-style-type: none"> Available as of version DLL 517, OUT 516, RBF 512. Last change with version DLL 518.
References	timed_para_jump_abs_3d , para_jump_rel_3d , timed_jump_rel_3d , timed_para_jump_rel



Normal List Command	timed_para_mark_abs
Function	Moves the laser focus for the specified marking duration along a 2D vector from the current position to the specified position (absolute coordinate values) within a two-dimensional image field and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Call	timed_para_mark_abs(X, Y, P, T)
Parameters	<p>X, Y Absolute coordinates of the mark vector end point in <i>bits</i> as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</p> <p>P End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (<i>Ctrl</i> parameter), identical with set_vector_control (<i>Value</i> parameter)</p> <p>T Duration of the complete mark vector in <i>microseconds</i> (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If $T < 5$, then timed_para_mark_abs behaves like para_mark_abs.</p>
Comments	<ul style="list-style-type: none"> Unlike para_mark_abs, the timed_para_mark_abs command does not execute the marking process with the specified (by set_mark_speed or set_mark_speed_ctrl) marking speed. Instead, the speed (i.e. the number of microsteps) is adjusted so that the vector lasts as long as specified (see chapter 8.10 "Timed Vector and Arc Commands", page 223). The total marking time is (for $T \geq 5$) the sum of the specified (rounded) time and the set delays. timed_para_mark_abs requires two list entries. During runtime, the command's part on the first list entry is executed as a short list command and afterward the second part is executed as a normal list command. Thereby, both command parts are executed within the same 10 µs clock, unless further (previous) short list commands induce a list_nop between the two parts. If $T < 5$, then timed_para_mark_abs behaves like para_mark_abs. Then it requires only one list entry. The “laser active” laser control signals are automatically turned on at the beginning of the command (or remain on after a directly preceding mark or arc command). The defined scanner and laser delays are thereby taken into account (see chapter 7.2 “Delay Settings for Synchronizing Scan Head and Laser Control”, page 111). Note that other delays are executed in Sky writing mode (see page 127). Exception: zero-length mark commands (see notes on page 115). During the conversion of specified coordinate values into scan system output values, any previously defined coordinate transformations, assigned correction tables etc. are taken into account after successful microvectorization (see page 141).



Normal List Command	timed_para_mark_abs
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).
Version info	Available as of version DLL 517, OUT 516, RBF 512.
References	para_mark_abs , timed_mark_abs , mark_abs , timed_para_mark_rel , timed_para_mark_abs_3d

List Multi-Command	timed_para_mark_abs_3d
Function	Moves the laser focus for the specified marking duration along a 3D vector from the current position to the specified position (absolute coordinate values) within a three-dimensional process volume and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as timed_para_mark_abs . However, microvectorization is calculated like a 3D command and hence influences the effective marking speed in the XY plane.
Call	timed_para_mark_abs_3d(X, Y, Z, P, T)
Parameters	<p>X, Y, Z Absolute coordinates of the mark vector end point in <i>bits</i> as signed 32-bit values. Allowed range:</p> <ul style="list-style-type: none"> For X and Y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. For Z: [-32768 ... 32767]. Out-of-range values are edge-clipped. <p>P End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (<i>Ctrl</i> parameter), identical with set_vector_control (<i>Value</i> parameter)</p> <p>T Duration of the complete mark vector in <i>microseconds</i> (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If T < 5, then timed_para_mark_abs_3d behaves like para_mark_abs_3d.</p>
Comments	<ul style="list-style-type: none"> Except for the additional motion in the third dimension, this command functions similarly to the timed_para_mark_abs command (see the comments there, exception: T < 5, see above). The X and Y axes can be controlled with 20-bit resolution, the Z axis with 16-bit resolution. The RTC5 (in RTC5 mode as well as in RTC4 compatibility mode) automatically upscales Z coordinate values internally to 20-bit values, so that the three dimensions of space can be handled equivalently in terms of the supplied jump speed value. During calculation of the Z output value to the scan system, any previously defined offset to the Z coordinate and focal length is taken into account (see page 141). The command occupies two list buffer positions. The initial component executes as an undelayed short list command before the primary component (a normal list command).
RTC4 → RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 mode and the RTC4 compatibility mode.
Version info	Available as of version DLL 517, OUT 516, RBF 512.
References	timed_para_mark_abs, para_mark_abs_3d, mark_abs_3d, timed_para_mark_rel_3d



Normal List Command	timed_para_mark_rel
Function	Moves the laser focus for the specified marking duration along a 2D vector from the current position to the specified position (relative coordinate values) within a two-dimensional image field and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Call	<code>timed_mark_rel(dx, dy, P, T)</code>
Parameters	<p>dx, dy <i>Relative coordinates of the mark vector end point in bits as signed 32-bit values. Allowed range: [-8388608 ... 8388607]. Out-of-range values are edge-clipped.</i> <i>The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table.</i></p> <p>P <i>End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (Ctrl parameter), identical with set_vector_control (Value parameter)</i></p> <p>T <i>Duration of the complete mark vector in microseconds (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If T < 5, then timed_para_mark_rel behaves like para_mark_rel.</i></p>
Comments	<ul style="list-style-type: none"> The coordinates for the mark vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to timed_para_mark_abs (see the comments there, exception: T < 5, see above).
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified coordinate values by 16 (the allowed range of values is correspondingly reduced).
Version info	<ul style="list-style-type: none"> Available as of version DLL 517, OUT 516, RBF 512. Last change with version DLL 518.
References	timed_para_mark_abs , para_mark_rel , timed_mark_rel , timed_para_mark_rel_3d



List Multi-Command	timed_para_mark_rel_3d
Function	Moves the laser focus for the specified marking duration along a 3D vector from the current position to the specified position (relative coordinate values) within a three-dimensional process volume and, simultaneously as well as linearly, changes the signal parameter selected by set_vector_control to the specified value.
Restriction	If the 3D option is not enabled or no 3D correction table has been assigned (see select_cor_table), then the command has the same effect as timed_para_mark_rel . However, microvectorization is calculated like a 3D command and hence influences the effective marking speed in the XY plane.
Call	timed_para_mark_rel_3d(dx, dy, dz, P, T)
Parameters	<p>dx, dy, dz <i>Relative coordinates of the mark vector end point in bits as signed 32-bit values. Allowed range:</i></p> <ul style="list-style-type: none"> • For x and y: [-8388608 ... 8388607]. Out-of-range values are edge-clipped. The complete value range [-8388608 ... 8388607] is only usable as a virtual image field e.g. for (enabled) Processing-on-the-fly applications. The current coordinates are clipped to [-524288 ... 524287] (real image field size) during runtime directly prior to use of the correction table. • For z: [-32768 ... 32767]. Out-of-range values are edge-clipped. <p>P <i>End value of the signal parameter as an unsigned 32-bit value. Allowed range: dependent on the selection made by set_vector_control (Ctrl parameter), identical with set_vector_control (Value parameter)</i></p> <p>T <i>Duration of the complete mark vector in microseconds (as a 64-bit IEEE floating point value). Allowed range: [0 ... 167772160]. The parameter is rounded to an integer-multiple of 10. Out-of-range values are clipped. If T < 5, then timed_para_mark_rel_3d behaves like para_mark_rel_3d.</i></p>
Comments	<ul style="list-style-type: none"> • The coordinates for the mark vector's end point are to be supplied as relative coordinates with respect to the current position. Otherwise, the command is identical to timed_para_mark_abs_3d (see the comments there, exception: T < 5, see above). • The command occupies two list buffer positions. The initial component executes as an undelayed short list command before the primary component (a normal list command).
RTC4→ RTC5	New command. In RTC4 compatibility mode, the RTC5 multiplies the specified value for the X and Y coordinates by 16 (the allowed range of values is correspondingly reduced). The value range for Z coordinates is identical for the RTC5 mode and the RTC4 compatibility mode.
Version info	<ul style="list-style-type: none"> • Available as of version DLL 517, OUT 516, RBF 512. • Last change with version DLL 518.
References	timed_para_mark_abs_3d, para_mark_rel_3d, timed_mark_rel_3d, timed_para_mark_rel



Ctrl Command	transform																					
Function	Performs a backward transformation of individual position values.																					
Call	TransformErrorCode = transform(&Sig1, &Sig2, Ptr, Code)																					
Parameters and Returned parameter values	<p>Sig1, Parameters: to-be-transformed position values as pointers to signed 32-bit values. Sig2 Returned parameter values: transformed position values as signed 32-bit values (the input values are overwritten).</p>																					
Parameters	Ptr	Pointer (in C and C++ data type ULONG_PTR, i.e. an unsigned 32-bit or 64-bit value) to the area of PC main memory to which the correction and transformation settings for backward transformation were previously transferred by upload_transform .																				
	Code	<p>This parameter (an unsigned 32-bit value) controls aspects of the backward transformation, particularly which partial transformations to perform: If a partial transformation is <i>not</i> to be performed, then its corresponding bit (#2...#5) should be set to 1.</p> <p>The parameter's meaning is similar to that of get_transform (Sig1 corresponds to Ptr1 and Sig2 to Ptr2).</p> <p>If bit #0 = 0, then both supplied position values (Sig1 and Sig2) are backward transformed as XY coordinates:</p> <table> <tr> <td>Bit #1</td> <td>= 0:</td> <td>The value supplied by Sig1 is backward transformed as the X coordinate and the value supplied by Sig2 as the Y coordinate.</td> </tr> <tr> <td></td> <td>= 1:</td> <td>The value supplied by Sig1 is backward transformed as the Y coordinate and the value supplied by Sig2 as the X coordinate.</td> </tr> <tr> <td>Bit #2</td> <td>= 0:</td> <td>The gain/offset correction of automatic self-calibration is backward transformed.</td> </tr> <tr> <td>Bit #3</td> <td>= 0:</td> <td>The image field correction is backward transformed.</td> </tr> <tr> <td>Bit #4</td> <td>= 0:</td> <td>The offset of the defined coordinate transformation is backward transformed.</td> </tr> <tr> <td>Bit #5</td> <td>= 0:</td> <td>The total matrix of the defined coordinate transformation is backward transformed.</td> </tr> <tr> <td>Bits #6...</td> <td>Reserved.</td> <td></td> </tr> </table>	Bit #1	= 0:	The value supplied by Sig1 is backward transformed as the X coordinate and the value supplied by Sig2 as the Y coordinate.		= 1:	The value supplied by Sig1 is backward transformed as the Y coordinate and the value supplied by Sig2 as the X coordinate.	Bit #2	= 0:	The gain/offset correction of automatic self-calibration is backward transformed.	Bit #3	= 0:	The image field correction is backward transformed.	Bit #4	= 0:	The offset of the defined coordinate transformation is backward transformed.	Bit #5	= 0:	The total matrix of the defined coordinate transformation is backward transformed.	Bits #6...	Reserved.
Bit #1	= 0:	The value supplied by Sig1 is backward transformed as the X coordinate and the value supplied by Sig2 as the Y coordinate.																				
	= 1:	The value supplied by Sig1 is backward transformed as the Y coordinate and the value supplied by Sig2 as the X coordinate.																				
Bit #2	= 0:	The gain/offset correction of automatic self-calibration is backward transformed.																				
Bit #3	= 0:	The image field correction is backward transformed.																				
Bit #4	= 0:	The offset of the defined coordinate transformation is backward transformed.																				
Bit #5	= 0:	The total matrix of the defined coordinate transformation is backward transformed.																				
Bits #6...	Reserved.																					

31



Ctrl Command	transform														
Parameters (cont'd)	<p>Code If bit #0 = 1, then one of the two supplied position values (specifiable as Sig1 or Sig2) is backward transformed as the Z coordinate:</p> <table> <tr> <td>Bit #1 = 0:</td><td>The value supplied by Sig1 is backward transformed as the Z coordinate (Sig2 remains unchanged).</td></tr> <tr> <td> = 1:</td><td>The value supplied by Sig2 is backward transformed as the Z coordinate (Sig1 remains unchanged).</td></tr> <tr> <td>Bit #2 = 0:</td><td>The offset to the focal length defined by set_defocus or set_defocus_list is backward transformed.</td></tr> <tr> <td>Bit #3 = 0:</td><td>The ABC correction is backward transformed.</td></tr> <tr> <td>Bit #4 = 0:</td><td>The offset to the Z coordinate defined by set_offset_xyz or set_offset_xyz_list is backward transformed.</td></tr> <tr> <td>Bits #5... Reserved.</td><td></td></tr> <tr> <td>31</td><td></td></tr> </table>	Bit #1 = 0:	The value supplied by Sig1 is backward transformed as the Z coordinate (Sig2 remains unchanged).	= 1:	The value supplied by Sig2 is backward transformed as the Z coordinate (Sig1 remains unchanged).	Bit #2 = 0:	The offset to the focal length defined by set_defocus or set_defocus_list is backward transformed.	Bit #3 = 0:	The ABC correction is backward transformed.	Bit #4 = 0:	The offset to the Z coordinate defined by set_offset_xyz or set_offset_xyz_list is backward transformed.	Bits #5... Reserved.		31	
Bit #1 = 0:	The value supplied by Sig1 is backward transformed as the Z coordinate (Sig2 remains unchanged).														
= 1:	The value supplied by Sig2 is backward transformed as the Z coordinate (Sig1 remains unchanged).														
Bit #2 = 0:	The offset to the focal length defined by set_defocus or set_defocus_list is backward transformed.														
Bit #3 = 0:	The ABC correction is backward transformed.														
Bit #4 = 0:	The offset to the Z coordinate defined by set_offset_xyz or set_offset_xyz_list is backward transformed.														
Bits #5... Reserved.															
31															
Result	<p>Error code as an unsigned 32-bit value:</p> <table> <tr> <td>Value</td><td>Description</td></tr> <tr> <td>0</td><td>Success.</td></tr> <tr> <td>1</td><td>Ptr = NULL (no memory area specified).</td></tr> <tr> <td>2</td><td>No valid data at Ptr (upload_transform did not execute).</td></tr> <tr> <td>3</td><td>Erroneous data at Ptr (a corresponding error indication has been stored by upload_transform).</td></tr> <tr> <td>4</td><td>Z-axis inversion not possible.</td></tr> </table>	Value	Description	0	Success.	1	Ptr = NULL (no memory area specified).	2	No valid data at Ptr (upload_transform did not execute).	3	Erroneous data at Ptr (a corresponding error indication has been stored by upload_transform).	4	Z-axis inversion not possible.		
Value	Description														
0	Success.														
1	Ptr = NULL (no memory area specified).														
2	No valid data at Ptr (upload_transform did not execute).														
3	Erroneous data at Ptr (a corresponding error indication has been stored by upload_transform).														
4	Z-axis inversion not possible.														
Comments	<ul style="list-style-type: none"> For backward transformation of position values see page 174. The execution of transform must be preceded by a call to upload_transform. Additionally, position values should have been requested by get_values. If execution of transform results in an error (returned error code > 0), then no transformation occurs (Sig1 and Sig2 then remain unchanged). Errors also include Ptr = NULL (error code = 1) or errors resulting from prior, erroneous execution of upload_transform (error code = 3). If backward transformation of Z values is requested (Code bit #0 = 1), but only a 2D correction table was assigned at the timepoint of the prior successful call to upload_transform, then the offsets to the focal length and Z coordinates are initialized with 0 and the values A, B and C are initialized with 0, 1, 0 (1-to-1 backward transformation). For backward transformation of XY position values (Code bit #0 = 0), only the Z = 0 plane is transformed. XY stretching and Z defocus resulting from Z deviations (particularly with non-F-Theta systems) are not taken into account. 														



Ctrl Command	transform
Comments (cont'd)	<ul style="list-style-type: none"> Because the command transform does not access any RTC5 Boards, calling it does not require explicit access rights to a specific board. If both the upload_transform data and the queried data recorded by get_values or get_waveform have been binarily stored on the PC, then offline operation of transform is also possible (then transform does not require the presence of an RTC5 Board on the PCI bus). transform is not available as a multi-board command. The board-specific error variables <code>LastErrorHandler</code> and <code>AccErrorHandler</code> (see "Error Handling", page 98) are neither generated nor altered by transform.
RTC4→ RTC5	<p>New command.</p> <p>Even in the RTC5's RTC4 compatibility mode, all back transformed values (including Z values) are in the 20-bit range and must, when necessary, be converted (by dividing by 16) by users.</p>
Version info	<ul style="list-style-type: none"> Available as of version DLL 516, OUT 515, RBF 512. Change with version DLL 517: parameter <code>Code</code> changed.
References	upload_transform , get_transform , get_values



Ctrl Command	upload_transform																														
Function	Transfers from the RTC5 Board to the PC all correction and transformation settings currently assigned to the scan system.																														
Call	<code>UploadErrorCode = upload_transform(HeadNo, Ptr)</code>																														
Parameters	<p>HeadNo Number of the scan head connector whose settings should be queried, as an unsigned 32-bit value. Allowed values: = 1: Primary scan head connector. = 2: Secondary scan head connector (activation required).</p> <p>Ptr Pointer (in C and C++ data type <code>ULONG_PTR</code>, i.e. an unsigned 32-bit or 64-bit value) to the PC's area of memory that should receive the queried settings.</p>																														
Result	<p>Error code as an unsigned 32-bit value:</p> <table> <tr><td>Bit #0</td><td>=1:</td><td>X gain (gain of automatic self-calibration for galvanometer 2) = 0.</td></tr> <tr><td>Bit #1</td><td>=1:</td><td>Y gain (gain of automatic self-calibration for galvanometer 1) = 0.</td></tr> <tr><td>Bit #2</td><td>=1:</td><td>The total matrix of the defined coordinate transformation is noninvertable.</td></tr> <tr><td>Bit #3</td><td>=1:</td><td>No correction table assigned.</td></tr> <tr><td>Bit #4</td><td>=1:</td><td>The ABC values (Z axis) are noninvertable.</td></tr> <tr><td>Bit #5</td><td>=1:</td><td>Error querying correction table.</td></tr> <tr><td>Bit #6</td><td>=1:</td><td>Parameter error: invalid <code>HeadNo</code> or <code>Ptr</code> = 0.</td></tr> <tr><td>Bit #7</td><td>=1:</td><td>BUSY error, board was BUSY or INTERNAL-BUSY (get_last_error return code <code>RTC5_BUSY</code>).</td></tr> <tr><td>Bits #8...</td><td>Reserved.</td><td></td></tr> <tr><td></td><td>31</td><td></td></tr> </table>	Bit #0	=1:	X gain (gain of automatic self-calibration for galvanometer 2) = 0.	Bit #1	=1:	Y gain (gain of automatic self-calibration for galvanometer 1) = 0.	Bit #2	=1:	The total matrix of the defined coordinate transformation is noninvertable.	Bit #3	=1:	No correction table assigned.	Bit #4	=1:	The ABC values (Z axis) are noninvertable.	Bit #5	=1:	Error querying correction table.	Bit #6	=1:	Parameter error: invalid <code>HeadNo</code> or <code>Ptr</code> = 0.	Bit #7	=1:	BUSY error, board was BUSY or INTERNAL-BUSY (get_last_error return code <code>RTC5_BUSY</code>).	Bits #8...	Reserved.			31	
Bit #0	=1:	X gain (gain of automatic self-calibration for galvanometer 2) = 0.																													
Bit #1	=1:	Y gain (gain of automatic self-calibration for galvanometer 1) = 0.																													
Bit #2	=1:	The total matrix of the defined coordinate transformation is noninvertable.																													
Bit #3	=1:	No correction table assigned.																													
Bit #4	=1:	The ABC values (Z axis) are noninvertable.																													
Bit #5	=1:	Error querying correction table.																													
Bit #6	=1:	Parameter error: invalid <code>HeadNo</code> or <code>Ptr</code> = 0.																													
Bit #7	=1:	BUSY error, board was BUSY or INTERNAL-BUSY (get_last_error return code <code>RTC5_BUSY</code>).																													
Bits #8...	Reserved.																														
	31																														
Comments	<ul style="list-style-type: none"> The queried and transferred data can be used for backward transforming actual position values by transform or get_transform (see also page 174). For storage of each queried data set, the user program must provide (at an address specified by <code>Ptr</code>) an area of PC main memory equal to 528520 bytes. In case of error (except for bit#6 = 1), an error indication is stored at <code>Ptr</code> to indicate that the data are erroneous. The commands transform and get_transform recognize this error information and ensure that the backward transformation is not executed (transform then generates a corresponding error code, get_transform generates a get_last_error return code of <code>RTC5_PARAM_ERROR</code>). The command is ignored (get_last_error return code: <code>RTC5_BUSY</code>) if the addressed board's BUSY status is currently set (list is being processed or has been halted by pause_list) or the board's INTERNAL-BUSY status is currently set. In contrast, the command is executed when a list has been paused by set_wait (PAUSED status set). During execution of the command, external list starts is suppressed. 																														
RTC4→RTC5	New command.																														
Version info	Available as of version DLL 516, OUT 515, RBF 512.																														
References	transform , get_transform																														



Ctrl Command	verify_checksum
Function	Tests for the presence of a checksum or creates a checksum for a correction file.
Call	<code>verify_checksum(Name)</code>
Parameter	Name name of the correction file as a pointer to a null-terminated ANSI string
Result	<p>Result of the test as an unsigned 32-bit value.</p> <ul style="list-style-type: none"> = 0: No error (the tested checksum is OK.). = 1: A checksum was newly created (no info about file integrity). = 2: The tested checksum is incorrect. = 3: A checksum could not be determined (file error, etc.).
Comments	<ul style="list-style-type: none"> • Verification of correction file downloads only works for files that contain a checksum (see page 99 and set_verify). • The verify_checksum command is also available without explicit access rights to a particular RTC5 Board. • verify_checksum is not available as a multi-board command. • The programs <code>CorrectionFileConverter.exe</code> (version 1.04) and <code>correXion5.exe</code> (version 1.01) together with <code>RTC5Base.dll</code> (version 1.0.0.4) already automatically create check sums for the output files. • The board-specific error variables <code>LastError</code> and <code>AccError</code> (see "Error Handling", page 98) are neither generated nor altered by verify_checksum.
RTC4→ RTC5	New command.
References	set_verify

Normal List Command	wait_for_encoder				
Function	Waits until the selected encoder counter has overstepped or understepped the specified count for the first time.				
Call	<code>wait_for_encoder(Value, EncoderNo)</code>				
Parameters	<table border="0"> <tr> <td>Value</td> <td>Count as a signed 32-bit value. Allowed range: $[-2^{31} \dots + (2^{31}-1)]$</td> </tr> <tr> <td>EncoderNo</td> <td>Number of the to-be-used encoder counter as an unsigned 32-bit value. Allowed values: = 0: Encoder counter Encoder0. = 1: Encoder counter Encoder1.</td> </tr> </table>	Value	Count as a signed 32-bit value. Allowed range: $[-2^{31} \dots + (2^{31}-1)]$	EncoderNo	Number of the to-be-used encoder counter as an unsigned 32-bit value. Allowed values: = 0: Encoder counter Encoder0. = 1: Encoder counter Encoder1.
Value	Count as a signed 32-bit value. Allowed range: $[-2^{31} \dots + (2^{31}-1)]$				
EncoderNo	Number of the to-be-used encoder counter as an unsigned 32-bit value. Allowed values: = 0: Encoder counter Encoder0. = 1: Encoder counter Encoder1.				
Comments	<ul style="list-style-type: none"> • The command is synonymous with wait_for_encoder_mode with parameter <code>Mode = 0</code> (see comments there). 				
RTC4→ RTC5	New command.				
References	wait_for_encoder_mode				



List Multi-Command	wait_for_encoder_in_range
Function	Waits until both encoder counters simultaneously lie within the specified range (including limits).
Call	<code>wait_for_encoder_in_range(EncXmin, EncXmax, EncYmin, EncYmax)</code>
Parameters	EncXmin, Limit values as signed 32-bit values. EncXmax, Allowed range: $[-2^{31} \dots + (2^{31}-1)]$. EncYmin, EncYmax
Comments	<ul style="list-style-type: none"> For usage of this command, see "Synchronization by Encoder Signals", page 246 and "Synchronization of Processing-on-the-fly Applications", page 211. The command requires two list memory positions. The first part is executed as an undelayed short list command prior to the second part, which executes as a normal list command. Any pending delayed short list commands execute first. If $\text{EncXmin} > \text{EncXmax}$, then both values are interchanged. If $\text{EncYmin} > \text{EncYmax}$, then both values are interchanged. If no encoder-based Processing-on-the-fly correction is active, then the command merely creates a waiting period (without galvanometer scanner motion). If $\text{EncXmin} = \text{EncXmax}$ (or $\text{EncYmin} = \text{EncYmax}$), then waiting until a specific encoder value is possible. The command is available even if the Processing-on-the-fly option is not enabled. The command does not alter the laser control signals. If you want the laser off during the wait, then this command must be preceded by some other command that switches off the "laser active" laser control signals (e.g. a list_nop). The active Processing-on-the-fly mode determines whether the galvanometer scanners remain stationary during the wait or move in accordance with encoder changes (see page 211): They move with set_fly_2d, but otherwise remain stationary.
RTC4→ RTC5	New command.
Version info	Available as of version DLL 536, OUT 536.
References	wait_for_encoder_mode , park_position , park_return



Normal List Command	wait_for_encoder_mode						
Function	Waits until the selected encoder counter has overstepped or understepped the specified count for the first time.						
Call	<code>wait_for_encoder_mode(Value, EncoderNo, Mode)</code>						
Parameters	<table> <tr> <td>Value</td> <td>Count as a signed 32-bit value. Allowed range: $[-2^{31} \dots +2^{31}-1]$</td> </tr> <tr> <td>EncoderNo</td> <td>Number of the to-be-used encoder counter as an unsigned 32-bit value. Allowed values: = 0: Encoder counter Encoder0. = 1: Encoder counter Encoder1.</td> </tr> <tr> <td>Mode</td> <td>Signed 32-bit value. = 0: Waits for understepping/overstepping (position dependent). > 0: Waits for overstepping (position independent). < 0: Waits for understepping (position independent).</td> </tr> </table>	Value	Count as a signed 32-bit value. Allowed range: $[-2^{31} \dots +2^{31}-1]$	EncoderNo	Number of the to-be-used encoder counter as an unsigned 32-bit value. Allowed values: = 0: Encoder counter Encoder0. = 1: Encoder counter Encoder1.	Mode	Signed 32-bit value. = 0: Waits for understepping/overstepping (position dependent). > 0: Waits for overstepping (position independent). < 0: Waits for understepping (position independent).
Value	Count as a signed 32-bit value. Allowed range: $[-2^{31} \dots +2^{31}-1]$						
EncoderNo	Number of the to-be-used encoder counter as an unsigned 32-bit value. Allowed values: = 0: Encoder counter Encoder0. = 1: Encoder counter Encoder1.						
Mode	Signed 32-bit value. = 0: Waits for understepping/overstepping (position dependent). > 0: Waits for overstepping (position independent). < 0: Waits for understepping (position independent).						
Comments	<ul style="list-style-type: none"> For usage of this command, see chapter 9.3.3 "Synchronization by Encoder Signals", page 246. For Mode = 0, ensure that the size and sign of the parameter Value is appropriate for the counting direction of the selected encoder (for external triggering, this corresponds to the workpiece's direction of motion). If Value > 0, the command waits for overstepping, otherwise for understepping. If Value is positive and already less than the current encoder count, then the command waits for a complete traversal of the counter (likewise if Value is negative and larger than the current encoder count). At a 1 MHz counter rate, this can take up to approx. 36 minutes! If Mode <> 0, then the command waits for overstepping/understepping of the Value parameter independently of the current position and direction of motion. If EncoderNo > 1, then wait_for_encoder_mode is replaced with a list_nop (get_last_error return code RTC5_PARAM_ERROR). If no encoder-based Processing-on-the-fly correction is active, then the command merely creates a waiting period (without galvanometer scanner motion) that allows implementation of an externally triggered synchronization (as an alternative to external starts, set_wait or if_cond, etc.). The command is available even if the Processing-on-the-fly option is not enabled. For Mode = 0, the command is synonymous with wait_for_encoder. The command does not alter the laser control signals. If you want the laser off during the wait, then this command must be preceded by some other command that switches off the "laser active" laser control signals (e.g. a list_nop). The active Processing-on-the-fly mode determines whether the galvanometer scanners remain stationary during the wait or move in accordance with encoder changes (see page 211): They move with set_fly_2d, but otherwise remain stationary. 						
RTC4→ RTC5	New command.						
References	get_encoder , store_encoder , read_encoder , simulate_encoder , wait_for_encoder , wait_for_encoder_in_range , park_position , park_return						



Normal List Command	wait_for_mcbsp
Function	Waits until the input value at the McBSP/SPI interface has reached, overstepped or understepped the specified value for the first time.
Call	<code>wait_for_mcbsp(Axis, Value, Mode)</code>
Parameters	<p>Axis Selects which half-word of the input value should be used for evaluation (see below) as an unsigned 32-bit value. Allowed values: = 0: lower half-word (X axis, galvanometer scanner 2) = 1: upper half-word (Y axis, galvanometer scanner 1)</p> <p>Value Threshold value as a signed 32-bit value</p> <p>Mode Signed 32-bit value. = 0: Waits for equality. > 0: Waits for overstepping. < 0: Waits for understepping.</p>
Comments	<ul style="list-style-type: none"> For information on using the command, see "Synchronization and Online Positioning by McBSP/SPI Signals", page 248. The command is similar to wait_for_encoder_mode, but the McBSP/SPI interface is queried. If set_fly_x_pos and set_fly_y_pos are <i>simultaneously</i> activated, then Value consists of two 16-bit half-words (see page 202). Only in this case does Axis control which half-word is used for evaluation. In all other cases, Axis is irrelevant and Value is interpreted as a signed 32-bit value. Axis must be either 0 or 1 (even if it is irrelevant). Otherwise, the command is replaced by list_nop (get_last_error return code RTC5_PARAM_ERROR). If neither set_fly_x_pos, set_fly_y_pos nor set_fly_rot_pos are activated, then the command merely creates a waiting period (without galvanometer scanner motion) that allows implementation of an externally triggered synchronization (as an alternative to external starts, set_wait or if_cond, etc.). The command is available even if the Processing-on-the-fly option is not enabled. The command does not alter the laser control signals. If you want the laser off during the wait, then this command must be preceded by some other command that switches off the "laser active" laser control signals (e.g. a list_nop).
RTC4→ RTC5	New command.
References	wait_for_encoder_mode

Ctrl Command	write_8bit_port
Function	Writes a value to the 8-bit digital output port on the EXTENSION 2 socket connector.
Call	<code>write_8bit_port(Value)</code>
Parameter	Value 8-bit output value (DATA0...DATA7) as an unsigned 32-bit value. Only the least significant 8 bits are evaluated.
Comments	<ul style="list-style-type: none"> See also chapter 9.1.2 "8-Bit Digital Output Port", page 233.
RTC4→ RTC5	Unchanged functionality.
References	write_8bit_port_list



Delayed Short List Command	write_8bit_port_list
Function	Same as write_8bit_port , but a list command.
Call	<code>write_8bit_port_list(Value)</code>
Parameter	Value 8-bit output value (DATA0...DATA7) as an unsigned 32-bit value. Only the least significant 8 bits are evaluated.
Comments	<ul style="list-style-type: none"> • See write_8bit_port.
RTC4→ RTC5	Unchanged functionality.

Ctrl Command	write_abc_to_file																					
Function	Writes the ABC values directly into a specified correction file on the PC.																					
Call	<code>ErrorNo = write_abc_to_file(Name, A, B, C)</code>																					
Result	<table> <tr> <td>ErrorNo</td> <td>Error code as an unsigned 32-bit value:</td> </tr> <tr> <td>0</td> <td>No error.</td> </tr> <tr> <td>1</td> <td>A exceeded the maximum allowed value.</td> </tr> <tr> <td>2</td> <td>A undercut the minimum allowed value.</td> </tr> <tr> <td>4</td> <td>B exceeded the maximum allowed value.</td> </tr> <tr> <td>8</td> <td>B undercut the minimum allowed value.</td> </tr> <tr> <td>16</td> <td>C exceeded the maximum allowed value.</td> </tr> <tr> <td>32</td> <td>C undercut the minimum allowed value.</td> </tr> <tr> <td>3</td> <td>File-open error (empty string, file not found etc.).</td> </tr> <tr> <td>12</td> <td>File error (checksum could not be determined, file corrupt).</td> </tr> </table>		ErrorNo	Error code as an unsigned 32-bit value:	0	No error.	1	A exceeded the maximum allowed value.	2	A undercut the minimum allowed value.	4	B exceeded the maximum allowed value.	8	B undercut the minimum allowed value.	16	C exceeded the maximum allowed value.	32	C undercut the minimum allowed value.	3	File-open error (empty string, file not found etc.).	12	File error (checksum could not be determined, file corrupt).
ErrorNo	Error code as an unsigned 32-bit value:																					
0	No error.																					
1	A exceeded the maximum allowed value.																					
2	A undercut the minimum allowed value.																					
4	B exceeded the maximum allowed value.																					
8	B undercut the minimum allowed value.																					
16	C exceeded the maximum allowed value.																					
32	C undercut the minimum allowed value.																					
3	File-open error (empty string, file not found etc.).																					
12	File error (checksum could not be determined, file corrupt).																					
Parameter	Name	Correction file name as a pointer to a null-terminated ANSI string.																				
	A, B, C	Coefficients (as 64-bit IEEE floating point values) of the parabolic function $z_{out} = A + B/z + C/z^2$ which is used for calculating the Z output values. Allowed ranges: see command load_z_table .																				
Comments	<ul style="list-style-type: none"> The command write_abc_to_file is also available without explicit access rights to a specific RTC5 Board. The command write_abc_to_file is not available as a multi-board command. The board-specific error variables <code>LastErrorHandler</code> and <code>AccErrorHandler</code> (see chapter 6.8 "Error Handling", page 98) are neither generated nor altered by write_abc_to_file. If the error code is not 0 or 12, ABC values are not outputted. 																					
RTC4→ RTC5	New command.																					
Version info	Available as of version DLL 538, OUT 538.																					
References	read_abc_from_file																					



Ctrl Command	write_da_1
Function	See write_da_x .
Call	<code>write_da_1(Value)</code>
Parameter	<p>Value 12-bit output value for the ANALOG OUT1 analog output port as an unsigned 32-bit value. Higher bits are ignored.</p> <p>Value = 0 corresponds to an output value of 0 V. Value = $2^{12}-1$ corresponds to an output value of 10 V.</p>
RTC4→ RTC5	Unchanged functionality. In RTC4 compatibility mode: as write_da_x

Delayed Short List Command	write_da_1_list
Function	See write_da_x_list .
Call	<code>write_da_1_list(Value)</code>
Parameter	<p>Value 12-bit output value for the ANALOG OUT1 analog output port as an unsigned 32-bit value. Higher bits are ignored.</p> <p>Value = 0 corresponds to an output value of 0 V. Value = $2^{12}-1$ corresponds to an output value of 10 V.</p>
RTC4→ RTC5	Unchanged functionality. In RTC4 compatibility mode: as write_da_x

Ctrl Command	write_da_2
Function	See write_da_x .
Call	<code>write_da_2(Value)</code>
Parameter	<p>Value 12-bit output value for the ANALOG OUT2 analog output port as an unsigned 32-bit value. Higher bits are ignored.</p> <p>Value = 0 corresponds to an output value of 0 V. Value = $2^{12}-1$ corresponds to an output value of 10 V.</p>
RTC4→ RTC5	Unchanged functionality. In RTC4 compatibility mode: as write_da_x



Delayed Short List Command	write_da_2_list
Function	See write_da_x_list .
Call	<code>write_da_2_list(Value)</code>
Parameter	<p>Value 12-bit output value for the ANALOG OUT2 analog output port as an unsigned 32-bit value. Higher bits are ignored.</p> <p>Value = 0 corresponds to an output value of 0 V. Value = $2^{12}-1$ corresponds to an output value of 10 V.</p>
RTC4→ RTC5	<p>Unchanged functionality.</p> <p>In RTC4 compatibility mode: as write_da_x.</p>

Ctrl Command	write_da_x
Function	Writes an output value to one of the analog output ports of the RTC5.
Call	<code>write_da_x(x, Value)</code>
Parameters	<p>x Number of the analog output port as an unsigned 32-bit value. Allowed range: [1, 2] (1: ANALOG OUT1, 2: ANALOG OUT2)</p> <p>Value 12-bit output value for the selected analog output port as an unsigned 32-bit value. Higher bits are ignored. Value = 0 corresponds to an output value of 0 V. Value = $2^{12}-1$ corresponds to an output value of 10 V.</p>
Comments	<ul style="list-style-type: none"> See also "12-Bit Analog Output Ports", page 233. The output range of the analog output ports is 0 V ... 10 V. For $x < 1$ or $x > 2$, the command is ignored (get_last_error return code <code>RTC5_PARAM_ERROR</code>). The commands write_da_1 / write_da_2 can be used alternatively to write_da_x (without parameter x).
RTC4→ RTC5	<p>Unchanged functionality.</p> <p>In RTC4 compatibility mode, the RTC5 multiplies the specified <code>Value</code> by 4 (the RTC4's analog outputs only had 10-bit resolution).</p>



Delayed Short List Command	write_da_x_list
Function	Same as write_da_x , but a list command.
Call	<code>write_da_x_list(x, Value)</code>
Parameters	x Number of the analog output port as an unsigned 32-bit value. Allowed range: [1, 2] (1: ANALOG OUT1, 2: ANALOG OUT2)
	Value 12-bit output value for the selected analog output port as an unsigned 32-bit value. Higher bits are ignored. Value = 0 corresponds to an output value of 0 V. Value = $2^{12}-1$ corresponds to an output value of 10 V.
Comments	<ul style="list-style-type: none"> For $x < 1$ or $x > 2$, the command is replaced with a list_nop (get_last_error return code RTC5_PARAM_ERROR).
RTC4→RTC5	Unchanged functionality. In RTC4 compatibility mode: as write_da_x .



Ctrl Command	write_hi_pos														
Function	Writes the specified values to the EEPROM of the RTC5 Board to be used as ASC reference values (= Home-In reference positions, not to confuse with Home-In positions).														
Call	Result = write_hi_pos (HeadNo, X1, X2, Y1, Y2)														
Parameters	<p>HeadNo Number of the scan head connector as an unsigned 32-bit value. Allowed values: = 1: Primary scan head connector. = 2: Secondary scan head connector (activation required).</p> <p>X1, X2, Reference positions in bits as signed 32-bit values Y1, Y2</p>														
Result	<p>Error code as an unsigned 32-bit value.</p> <table> <tr> <td>0</td> <td>Kein Fehler.</td> </tr> <tr> <td>Bit #0</td> <td>=1: Wrong HeadNo.</td> </tr> <tr> <td>Bit #1</td> <td>=1: Wrong sensor position for X1.</td> </tr> <tr> <td>Bit #2</td> <td>=1: Wrong sensor position for X2.</td> </tr> <tr> <td>Bit #3</td> <td>=1: Wrong sensor position for Y1.</td> </tr> <tr> <td>Bit #4</td> <td>=1: Wrong sensor position for Y2.</td> </tr> <tr> <td>Bit #5</td> <td>=1: Invalid ASC version.</td> </tr> </table>	0	Kein Fehler.	Bit #0	=1: Wrong HeadNo.	Bit #1	=1: Wrong sensor position for X1.	Bit #2	=1: Wrong sensor position for X2.	Bit #3	=1: Wrong sensor position for Y1.	Bit #4	=1: Wrong sensor position for Y2.	Bit #5	=1: Invalid ASC version.
0	Kein Fehler.														
Bit #0	=1: Wrong HeadNo.														
Bit #1	=1: Wrong sensor position for X1.														
Bit #2	=1: Wrong sensor position for X2.														
Bit #3	=1: Wrong sensor position for Y1.														
Bit #4	=1: Wrong sensor position for Y2.														
Bit #5	=1: Invalid ASC version.														
Comments	<ul style="list-style-type: none"> The command write_hi_pos is used in cases when a scan head is moved from RTC5 Board "A" to RTC5 Board "B" in order to transfer its Home-In reference positions from RTC5 Board "A" to RTC5 Board "B". Use get_hi_pos(HeadNo) to read out the Home-In reference positions of RTC5 Board "A" (only after init_RTC5_dll, see the get_hi_pos command description). The command write_hi_pos is also executed if the scan head is switched off or even a scan head is not attached. With an ASC type-1 equipped scan head attached, auto_cal(Command = 4) (or auto_cal(Command = 0) because this command implicitly executes auto_cal(Command = 4)) must have been successfully executed at last on the RTC5 Board "B". A cross-check with get_auto_cal(HeadNo) needs to return 100. Otherwise, write_hi_pos would return error bit #5 = 1. If required, connect the scan head to RTC5 Board "B" and execute auto_cal(HeadNo, 4) before executing write_hi_pos. 														
RTC4→RTC5	New command.														
Version info	Available as of version DLL 538, OUT 538.														
References	get_hi_pos, get_auto_cal, auto_cal														



Ctrl Command	write_io_port
Function	Writes a value to the 16-bit digital output port on the EXTENSION 1 socket connector.
Call	<code>write_io_port(Value)</code>
Parameter	Value 16-bit output value (DIGITAL OUT0 ... DIGITAL OUT15) as an unsigned 32-bit value. Only the least significant 16 bits are evaluated.
Comments	<ul style="list-style-type: none"> • Use the commands <code>set_io_cond_list</code> and <code>clear_io_cond_list</code> to set or clear individual bits of the 16-bit digital output port, depending on the state of the <i>input</i> port. • The <code>write_io_port_mask</code> and <code>write_io_port_mask_list</code> commands also allow changing selectable bits of the 16-bit digital output port as desired. • See also chapter 9.1.1 "16-Bit Digital Output Port", page 232.
RTC4→ RTC5	Unchanged functionality.
References	write_io_port_list , write_io_port_mask , write_io_port_mask_list , set_io_cond_list , clear_io_cond_list , get_io_status

Delayed Short List Command	write_io_port_list
Function	Same as <code>write_io_port</code> , but a list command.
Call	<code>write_io_port_list(Value)</code>
Parameter	Value 16-bit output value (DIGITAL OUT0 ... DIGITAL OUT15) as an unsigned 32-bit value. Only the least significant 16 bits are evaluated.
Comments	<ul style="list-style-type: none"> • See <code>write_io_port</code>.
RTC4→ RTC5	Unchanged functionality.



Ctrl Command	write_io_port_mask	
Function	Writes those bits of <code>Value</code> specified by the <code>Mask</code> parameter to the 16-bit digital output port on the EXTENSION 1 socket connector.	
Call	<code>write_io_port_mask(Value, Mask)</code>	
Parameters	Value	16-bit output value (DIGITAL OUT0 ... DIGITAL OUT15) as an unsigned 32-bit value. Only the least significant 16 bits are evaluated.
	Mask	16-bit mask (for DIGITAL OUT0 ... DIGITAL OUT15) as an unsigned 32-bit value. Only the least significant 16 bits are evaluated.
Comments	<ul style="list-style-type: none"> The <code>Mask</code> parameter determines <i>which</i> bits of the 16-bit digital output port are to be altered, the <code>Value</code> parameter determines <i>how</i> they are altered. All bits of the 16-bit digital output port that were not set in <code>Mask</code> remain unaltered, i.e. they are outputted again as previously. For <code>Mask</code> = <code>0xFFFF</code>, <code>write_io_port_mask</code> behaves like <code>write_io_port</code>. See also chapter 9.1.1 "16-Bit Digital Output Port", page 232. 	
RTC4→ RTC5	New command.	
Version info	Available as of version DLL 517, OUT 516, RBF 512.	
References	write_io_port , write_io_port_list	

Delayed Short List Command	write_io_port_mask_list	
Function	Same as <code>write_io_port_mask</code> , but a list command.	
Call	<code>write_io_port_mask_list(Value, Mask)</code>	
Parameters	Value	16-bit output value (DIGITAL OUT0 ... DIGITAL OUT15) as an unsigned 32-bit value. Only the least significant 16 bits are evaluated.
	Mask	16-bit mask (for DIGITAL OUT0 ... DIGITAL OUT15) as an unsigned 32-bit value. Only the least significant 16 bits are evaluated.
Comments	<ul style="list-style-type: none"> See write_io_port_mask. 	
RTC4→ RTC5	New command.	
Version info	Available as of version DLL 517, OUT 516, RBF 512.	
References	write_io_port_mask , write_io_port_list	



10.3 Unsupported RTC2/RTC3/RTC4 Commands

Some RTC3/RTC4 commands and a few RTC2 commands emulated by the RTC3/RTC4 are not supported by the RTC5. But most of these can be replaced by RTC5 commands. The following table lists all unsupported commands and appropriate RTC5 replacements.

Ctrl Command	aut_change
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	auto_change

Ctrl Command	dsp_start
Support status	This RTC2/RTC3/RTC4 command is not supported by the RTC5.
RTC4→ RTC5	This command is not needed for normal operation. The DSP starts automatically after the program file is loaded by the command load_program_file .

List Command	field_jump
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	home_position , the “field jump” is automatically executed.

Ctrl Command	get_RTC2_mode
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	The RTC5 has no need to query the mode, which is now be set by the software command set_laser_mode instead of hardware.

Ctrl Command	get_RTC2_version
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	get_RTC_version

Ctrl Command	get_version
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	get_hex_version



Ctrl Command	get_xy_pos
Support status	This RTC2/RTC3/RTC4 command is not supported by the RTC5.
RTC4→ RTC5	Replaced by get_value

Ctrl Command	get_xyz_pos
Support status	This RTC2/RTC3/RTC4 command is not supported by the RTC5.
RTC4→ RTC5	Replaced by get_value

List Command	home_jump
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	home_position , the “home jump” is automatically executed.

List Command	laser_on
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	laser_on_list

Ctrl Command	load_cor
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	load_correction_file

Ctrl Command	load_pro
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	load_program_file

List Command	pola_abs, polb_abs, polc_abs
Support status	These RTC2 commands are not supported by the RTC5.
Replaced by	mark_abs

Ctrl Command	read_pixel_ad
Support status	This RTC2/RTC3/RTC4 command is not supported by the RTC5.
RTC4→ RTC5	There is no equivalent command for the RTC5.



Ctrl Command	rtc3_count_cards / rtc4_count_cards
Support status	This RTC3/RTC4 commands are not supported by the RTC5.
Replaced by	rtc5_count_cards

Ctrl Command	select_list
Support status	This RTC2/RTC3/RTC4 command is not supported by the RTC5.
RTC4→RTC5	select_list(0) can be replaced by set_extstartpos(0) , select_list(1) by set_extstartpos(Mem1) . Thereby, Mem1 is the memory size of "List 1" (and the absolute start address of "List 2"). After initialization (with load_program_file), Mem1 is 4000, otherwise as set by config_list . Mem1 can be determined (e.g. after a board changed "ownership") by set_start_list_2 and get_input_pointer .

Ctrl Command	set_base
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	There is no equivalent command for the RTC5 (because it is a plug-and-play board whose base address is automatically set).

Ctrl Command	set_co2_standby
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	set_standby

List Command	set_co2_standby_list
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	set_standby_list

List Command	set_delays
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	set_laser_delays, set_scanner_delays

List Command	set_encoder_values
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	set_fly_x, set_fly_y



Ctrl Command	set_gain
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	set_matrix, set_offset

Ctrl Command	set_list_mode
Support status	This RTC3/RTC4 command is not supported by the RTC5.
RTC4→RTC5	See " Circular Queue Mode ", page 90.

Ctrl Command	set_mode
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	There is no equivalent command for the RTC5 , because its image field correction algorithm is always enabled. Instead, a 1-to-1 correction file can be loaded.

Ctrl Command	set_piso_control
Support status	This RTC3/RTC4 command is not supported by the RTC5.
RTC4→RTC5	This command is not needed for operation of the RTC5. For data transfer in accordance with the SL2-100 protocol, the bi-directional communication between the scan system and the RTC5 does not depend on the data cable length. For data transfer in accordance with the XY2-100 protocol, the timing of the communication must be further adjusted to reflect the length of the data cable; but the adjustment is now realized by a jumper at the XY2-100 converter, see chapter 4.3.2 "XY2-100 Converter (Optional)" , page 43.

Ctrl Command	set_speed
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	set_jump_speed, set_mark_speed

Ctrl Command	set_wobbel_xy
Support status	This RTC4 command is not supported by the RTC5.
Replaced by	set_wobbel, set_wobbel_mode

Ctrl Command	set_yag_parameter
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	There is no equivalent command for the RTC5.



Ctrl Command	write_da
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	wwrite_da_1

List Command	write_da_list
Support status	This RTC2 command is not supported by the RTC5.
Replaced by	wwrite_da_1_list

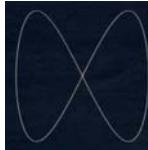
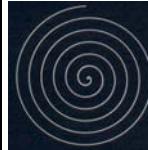
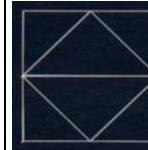
Ctrl Command	z_out
Support status	This RTC3/RTC4 command is not supported by the RTC5.
RTC4→RTC5	For setting the Z value in a 3-axis scan system, the command set_defocus can be used (only together with an RTC5 with enabled 3D option). After load_z_table(0.0, 1.0, 0.0) , set_defocus(z) has the same effect as z_out(z) (see also set_offset_xyz).

List Command	z_out_list
Support status	This RTC3/RTC4 command is not supported by the RTC5.
RTC4→RTC5	After load_z_table(0.0, 1.0, 0.0) , set_defocus_list(z) has the same effect as z_out_list(z) (see also set_offset_xyz_list).

11 Demo Programs

The RTC5 software package contains various program code samples in the folder `DemoFiles` (see also section "Demo Files", page 22). They demonstrate DLL and RTC5 initialization, error handling and usage of the diverse control and list commands.

They contain the required calling sequences of RTC5 commands, which you can easily translate into your preferred programming language. The following table summarizes the characteristics of the demo programs.

File name	Demo1	Demo2	Demo3	Demo4	Demo5	Demo6	Demo7
Marking Task	Square and triangle 	Lissajous figures 	Archimedean spirals 	Raster image reproduction 	Squares and triangles 	Raster image reproduction, Processing-on-the-fly	Text output, intelliSCAN
Laser type	CO ₂	YAG	YAG	CO ₂	YAG	CO ₂	YAG
Marking method	vector	vector	vector	pixel mode	vector	pixel mode, fly mode	vector
Additional features			home jump		home jump	data recording, time measurement, 16-bit IO port	data recording, time measurement, iDRIVE functions
DLL linking	implicit	explicit	explicit	explicit	explicit	implicit	implicit
List handling	use of a single list	single list, continuous transfer (circular queue)	two alternating lists, continuous transfer	two alternating lists, continuous transfer	use of two lists	two alternating lists, continuous transfer	two lists (default), protected area
External control inputs					/START, /STOP	allowed	
Exception handling	load_list	get_status, set_wait, release_wait, pause_list, restart_list, stop_execution	get_status, load_list, pause_list, restart_list, stop_execution	load_list	get_status, load_list, stop_execution	get_status, load_list, stop_execution	get_status
Other commands	config_list, set_end_of_list, execute_list_pos	config_list, set_start_list_pos, execute_list_pos, set_end_of_list	config_list, set_start_list_pos, set_end_of_list, execute_list_pos, auto_change	config_list, set_end_of_list, execute_list_pos, auto_change, set_pixel_line, set_n_pixel	config_list, set_end_of_list, execute_list_pos, set_extstartpos, auto_change, set_control_mode, get_startstop_info, bounce_supp	config_list, set_end_of_list, execute_list_pos, auto_change, set_control_mode, simulate_encoder, set_pixel_line, set_n_pixel, set_fly_x_fly_return, set_trigger, save_and_restart_timer, write_io_port_list, get_io_status	control_command, load_char, mark_text, mark_text_abs, set_start_list_pos, set_trigger, measurement_status, get_waveform, set_start_list_pos, save_and_restart_timer, set_end_of_list, execute_list_pos

12 Troubleshooting

Problem	Remedy
PC does not boot	<p>Switch off the PC and check the following:</p> <ul style="list-style-type: none"> • Check if the RTC5 Board is correctly seated in the PCI slot. Refer to the instructions in your PC manual. • Check for metal parts that may have fallen into the PC housing during installation of the RTC5. • Check for loose cables or connectors
RTC5 does not respond	<ul style="list-style-type: none"> • Check the driver installation. See chapter 5.4 "Installing the RTC5 Software", page 62. • Use the included HPGL converter program to check if the board can be properly accessed. If not: Check the cable type and the cable length. If the scan system is controlled by an XY2-100 converter, then check, whether the solder jumpers of the converter are set appropriate for the cable length. See figure 8. If yes: Check the DLL import declarations in your user program. See chapter 6.2.2 "Importing Commands", page 66.
User program fails	<ul style="list-style-type: none"> • Check the driver installation. See chapter 5.4 "Installing the RTC5 Software", page 62. • Check the RTC5 initialization in the user program. • Check the DLL import declarations in your user program. See chapter 6.2.2 "Importing Commands", page 66.
Scan head control fails	<ul style="list-style-type: none"> • Check if the scan head is properly connected to the RTC5 by the data cable. Make sure to follow the specifications for the data cable. See section chapter 4.3.3 "Data Cables (Accessories)", page 45. • Check the power supply of the scan head. Refer to your scan head operating manual. • Check your user program.
Laser control fails	<ul style="list-style-type: none"> • Check the interface between the RTC5 and the laser.
Irregular marking results	<ul style="list-style-type: none"> • Check the laser and scanner delays. See chapter 7.2.3 "Notes on Optimizing the Delays", page 121.
Laser does not switch off during jump commands	<ul style="list-style-type: none"> • Check the laser and scanner delays. See chapter 7.2 "Delay Settings for Synchronizing Scan Head and Laser Control", page 111.



Additionally, the following commands can be helpful for troubleshooting:

- With **get_error** and **get_last_error**, nearly any command can be checked for proper execution (see [chapter 6.8 "Error Handling", page 98](#)).
- With **set_verify**, you can verify that all downloads (commands, tables) were performed error-free (see [section "Download Verification", page 99](#)).
- With **get_value** or **get_values**, you can query specific values returned from the scan-system. With **set_trigger**/**set_trigger4** and **get_waveform**, you can record an entire series of returned values (see [section "Status Monitoring and Diagnostics", page 143](#); for iDRIVE scan systems see also [chapter 8.1, page 172](#)).
- With **set_wait** and **get_wait_status**, you can check if program branches (conditional jumps) executed as intended.
- With **get_overrun**, you can check if overruns of the 10 µs clock period occurred (see [section "Clock Overruns", page 142](#)).
- With **get_status** or **get_out_pointer**, you can determine which command number the program is currently executing (e.g. for "infinite loops" due to a circular argument in the program flow).
- **get_startstop_info** provides information about the laser signals and possible transmission errors to and from the attached scan system.

If specific outputs from a port have no effect, then check if the user program is performing directly consecutive accesses of that same port. Because so-called short list commands (see [page 250](#)) are typically used for this, one command might overwrite the other's output value within the same 10 µs clock period. In this situation, separate both short commands with a (normal) list command, e.g. **list_nop** or **list_continue**.

If the execution time (measured by **save_and_restart_timer** and **get_time**) does not correspond with your calculation, check if your user program contains so-called short list commands, which generally do not require their own clock period for execution. Another possibility is that additional scanner delays were automatically inserted to prevent (improper) overlap of LaserOn and LaserOff (see [section "Automatic Delay Adjustments", page 121](#)).

If the problems persist, contact SCANLAB.



13 Customer Service

13.1 Servicing and Repairs

All servicing and repairs should be performed only at SCANLAB. The warranty expires if the board has been altered.

13.2 Warranty

SCANLAB guarantees this product to be free of defects in manufacturing and material. The warranty is valid for 12 months after delivery. Repairs covered under the warranty are performed at SCANLAB.

The scope of the warranty is limited to repair or replacement of the SCANLAB product.

SCANLAB is responsible for the return delivery of products repaired under warranty; the customer is responsible for delivery to SCANLAB.

SCANLAB is not held responsible:

- when the product has been damaged through misuse or improper operation
- for repairs not performed by SCANLAB
- if the RTC5 Board has been altered
- for damage resulting from improper packaging of a product returned to SCANLAB
- for consequential damages

13.3 Contacting SCANLAB

For service, repairs, advice or information, simply contact SCANLAB using one of the contact possibilities listed below:

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

13.4 Product Disposal

The RTC5 can be returned to SCANLAB for a fee to be properly disposed of.

14 Technical Specifications of the RTC5 PCI Board

System Requirements		Interfaces to the Laser and Peripherals	
Windows PC with PCI bus interface		<ul style="list-style-type: none"> • LASER Connector 	
Operating system	Microsoft Windows 10, 8, 7, Vista, XP SP2, XP SP3 as 32-bit or 64-bit version	Connector	15-pin D-SUB connector, female
Dimensions		<ul style="list-style-type: none"> Laser signals 	
Length	160 mm	• TTL level	5 V, active-high or active-low (programmable)
Height	100 mm	• Max. current load	20 mA
Interface to Scan System		• Reference	GND (alternatively: GND2 with optional galvanic decoupling)
Connectors	Primary scan head connector SCAN HEAD (9-pin D-SUB, female)	Analog outputs	ANALOG OUT1, ANALOG OUT2
• optionally activated:	Secondary scan head connector 2. SCAN HEAD (10-pin socket connector)	• Output voltage range	0 V ... 10 V
Signals	SL2-100 protocol	• Resolution	12 Bit
• optional:	XY2-100 protocol by XY2-100 converter	• Max. current load	5 mA
Scan System Control		• Reference	GND
Number of list buffer areas	up to 3 (configurable)	Digital input	2 Bits
Capacity of list buffer	1000000 commands	• LOW level	< 0.6 V
Position update period (microstep period)	10 µs	• HIGH level	> 2.3 V
Maximum range for the image field coordinates	-524288 to +524287 (20-bit signed)	• Max. input voltage range	-0.5 V ... +5.5 V
Virtual image field (e.g. for Processing-on-the-fly)	-8388608 to +8388607 (24-bit signed)	• Input resistance (pull-up)	> 4.7 kΩ
		• Reference	GND
		Digital output	2 bits, buffered
		• LOW level	< 0.55 V
		• HIGH level	> 3.8 V
		• Max. current load	20 mA
		• Reference	GND



Inputs for external start and stop signals	TTL active-LOW, internally connected to +3.3 V by pull-up resistors (4.7 kΩ)	Other signals	
• /START	edge sensitive	• BUSY OUT	3.3 V or 5 V, TTL active-high, max. 10 mA
• /STOP	level sensitive	• VCC	3.3 V or 5 V, max. 100 mA
• Reference	GND	• +5 V	max. 100 mA
Other signals		• Reference	GND
• BUSY OUT	5 V, TTL active-high, max. 10 mA	• EXTENSION 2 Connector	
• +5 V	max. 100 mA	Connector	26-pin socket connector, configurable (by JP2-JP8)
• Reference	GND	Digital output	8 bits, buffered
• EXTENSION 1 Connector		• LOW level	< 0.4 V
Connector	40-pin socket connector, output signal level configurable (by JP1)	• HIGH level	> 2.0 V
Digital output	16 bits, buffered	• Max. current load	±8 mA
• LOW level	< 0.4 V	• Reference	GND
• HIGH level	> 2.0 V (3.3 V or 5 V)	• LATCH signal	5 V, TTL active-high, 5 µs pulse, max. 10 mA
• Max. current load	±8 mA	Laser signals	LASERON, LASER1, LASER2 (see LASER connector)
• Reference	GND	Other signals	
• LATCH signal	3.3 V or 5 V, TTL active-high, 5 µs pulse, max. 10 mA	• +5 V	max. 100 mA
Digital input	16 bits	• Reference	GND
• LOW level	< 0.5 V		
• HIGH level	> 2.6 V ... 24 V		
• Max. input voltage range	-0.5 V ... +26 V		
• Input resistance	> 10 kΩ		
• Reference	GND		
• SYNC signal	3.3 V or 5 V, TTL active-high, square wave signal (5 µs pulse, 10 µs period) max. 10 mA		



- **MARKING ON THE FLY Connector**

Connector	16-pin socket connector
2 Encoder inputs for incremental encoders	ENCODER X($1\pm,2\pm$) and ENCODER Y($1\pm,2\pm$), designed for a pair of standardized differential input signals (RS422) each. HIGH level ≥ 2.0 V LOW level ≤ 0.8 V $f \leq 4$ MHz
Analog outputs	ANALOG OUT2 (see LASER connector)
Inputs for external start and stop signals	/START2, /STOP2 (see /START, /STOP of the LASER connector)
Other signals	<ul style="list-style-type: none"> • BUSY OUT 5 V, TTL active-high, max. 10 mA • +5 V max. 100 mA • Reference GND

- **RS232 Interface**

Connector	10-pin socket connector
Input	RxD
• max. voltage range	-25 V ... +25 V
Output	TxD
• max. voltage range	-13 V ... +13 V
Reference	GND
Baud rate	300 ... 115200

- **"SPI / I2C" Connector**

Connector	10-pin socket connector
In McBSP configuration, see page 54 :	
• Transmitter signal level	3.3 V TTL
• Receiver signal level	3.3 V or 5 V TTL
• McBSP mode	Single Phase Frame Single Element per Frame 32 bits per Element DataDelay N bit
• Reference	GND
In SPI configuration, see page 57 :	
• Transmitter signal level	3.3 V TTL
• Receiver signal level	3.3 V or 5 V TTL
• SPI mode	Single Phase Frame Single Element per Frame 32 bits per Element DataDelay 1 bit
• Reference	GND

- **"STEPPER MOTOR" Connector**

Connector	10-pin socket connector
Signals for controlling two stepper motors:	
• ENABLE and DIRECTION outputs	5 V, TTL
• CLOCK output	5 V, TTL active high, 5 μ s pulse
• SWITCH input	TTL active-LOW, internally connected to +3.3 V by pull-up resistors (10 k Ω)
• Reference	GND



14.1 Compliance with EC Guidelines for Electromagnetic Compatibility (EMC)

The RTC5 PCI Board has been determined to be in compliance with EC directive 2004/108/EC (electromagnetic compatibility).

For that purpose, an RTC5 PCI Board has been integrated to a PC and was tested together with a hurrySCAN® 25 scan head (with SL2-100 interface).

Test Specifications

Evidence of fulfillment of the protection goals of EC directive 2004/108/EG (CE Conformity for EMC) based on

- EN 61000-6-2: 2005
- EN 61000-6-4: 2007 + A1: 2011

Result

The devices under test fulfill the specifications.

14.2 Compliance with FCC Rules

The RTC5 PCI Board has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC rules.

These limits are designed to provide reasonable protection against harmful interference when the RTC5 Board is operated in a commercial environment. The RTC5 Board generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with this instruction manual, may cause harmful interference to radio communications. Operation of the RTC5 Board in a residential area is likely to cause harmful interference in which case the user is required to correct the interference at his own expense.

15 Appendix A: The RTC5 PC/104-Plus Board

15.1 Product Overview

15.1.1 Intended Use – Comparison to the RTC5 PCI Board

Just like the RTC5 PCI Board, the SCANLAB RTC5 PC/104-Plus Board is intended for synchronous real-time control of scan systems, lasers and peripheral equipment. It differs from the RTC5 PCI Board in the following hardware-related details:

- The RTC5 PC/104-Plus Board is a PC/104-Plus module. Unlike an RTC5 PCI Board, it is not installed in a PC slot. Instead, it connects by its stack-through connectors (directly or indirectly) to a PC/104-Plus CPU board. Up to four RTC5 PC/104-Plus Boards can be operated with one CPU board.
- The RTC5 PC/104-Plus Board provides additional jumpers for customized configuration with respect to both the power supply and the position in the PC/104 stack.
- The number, type and positioning of connectors for controlling scan systems, lasers and peripheral equipment differ from that of the RTC5 PCI Board.

Otherwise, the RTC5 PC/104-Plus Board provides the same functionality for controlling scan systems, lasers and peripheral equipment as the RTC5 PCI Board. Specifications for signal inputs and outputs are the same as with the RTC5 PCI Board, as are software installation and developing user programs.

Both the RTC5 PC/104-Plus Board and the RTC5 PCI Board use the same software package (driver, DLL, import declarations, etc.) and same command set.

15.1.2 System Requirements

Hardware

The RTC5 PC/104-Plus Board is a PC/104-Plus module with a corresponding form factor. The dimensions of the RTC5 PC/104-Plus Board are shown in [figure 64](#) (note that the 100-pin MULTI connector projects slightly beyond the RTC5 PC/104-Plus Board's PCB edge). The RTC5 PC/104-Plus Board provides PCI and ISA stack-through connectors and can be directly or indirectly (in a stack of several PC/104-Plus modules) connected to a PC/104-Plus CPU board. If necessary, the board can be electrically configured for its position in a stack by jumpers (see [page 665](#)).

Data transfer is by the board's PCI bus in compliance with PCI Local Bus Specification Revision 2.2. Stacks of up to four PC/104-Plus modules (plus the PC/104-Plus CPU board) can be created. PC/104 standard modules (ISA modules not designed for the PCI bus) might need to be moved to higher stack positions.

RTC5 PC/104-Plus Boards intended for master/slave synchronization must be stacked adjacently.

To supply power for the RTC5 PC/104-Plus Board, the CPU board must provide at least +5 V at the PCI or ISA bus. A jumper on the RTC5 PC/104-Plus Board configures which bus is to provide the power (see [page 666](#)).

A supplementary ±12 V power supply at the PCI bus is advantageous, but not mandatory (see [page 666](#)).

Software

The RTC5 PC/104-Plus Board can use the same drivers and DLL files as the RTC5 PCI Board. Additionally to the operating systems listed in [chapter 2.2.2 "Software", page 26](#) for the RTC5 PCI Board, the RTC5 drivers and RTC5 DLL files also support the Microsoft Windows XP Embedded operating system (i.e. the 32-bit and 64-bit versions of Windows XP Embedded ≥ SP2 together with RTC5-software-package version 2011_09_29 or later or DLL version ≥ DLL 528). Up to four RTC5 PC/104-Plus Boards can be operated by a CPU board.



Caution!

- The RTC5 PC/104-Plus Board does not support power-saving modes that switch off power to the PCI bus. Accordingly, you must disable standby or sleep modes of the operating system. See also the note on [page 26](#).

15.1.3 Optional Functionality

The RTC5 PC/104-Plus Board can be configured for the same functionality as the RTC5 PCI Board (see [page 27](#)). Optional functionality must be enabled by SCANLAB or installed.

15.1.4 Labeling

The serial number of the RTC5 PC/104-Plus Board is printed on a white label attached to the board (format: "RTC SN...").

The board's ID number and configuration are described in the packaging list (see also "[Accessories](#)", [page 29](#) and "[Type Identification](#)", [page 656](#)).

15.1.5 Type Identification

The ID number of the RTC5 PC/104-Plus Board reveals the board's factory-equipped jumper configuration (JP1-JP8). The jumper configuration is additionally encoded in a three-digit type code scheme:

Digit 1	=0: Jumper JP1 open (no signal) =1: Jumper JP1 in position 1-2 (5 V) =2: Jumper JP1 in position 2-3 (3.3 V)
Digit 2 (relates to pin B40)	=0: Jumper JP2-JP4 open (no signal) =1: Jumper JP2 closed (+5 V) =2: Jumper JP3 closed (DATA7) =3: Jumper JP4 closed (GND)
Digit 3 (relates to pin B42)	=0: Jumper JP5-JP8 open (no signal) =1: Jumper JP5 closed (+5 V) =2: Jumper JP6 closed (DATA7) =3: Jumper JP7 closed (GND) =4: Jumper JP8 closed (LATCH)

Examples:

- "Type 000": jumpers JP1-JP8 open
- "Type 124": JP1 in position 1-2 (5 V signal level), JP3 closed (DATA7 at pin B40), JP8 closed (LATCH at pin B42)

The product is delivered with jumpers **JP10**, **JP12**, **J17**, **JP18** and **JP19** in their default configurations (see [page 665](#)).



15.1.6 Unpacking Instructions and Typical Package Contents

- ▶ Carefully remove the RTC5 PC/104-Plus Board from the package.
- ▶ Keep the packaging, including the antistatic bag the RTC5 PC/104-Plus Board is delivered in, so that in case of repair the board can be properly repackaged and returned to SCANLAB.
- ▶ Also remove all other articles from the package. Check that all parts have been delivered. Refer to the corresponding packaging list.
The package typically includes an RTC5 PC/104-Plus Board and a data CD (containing the corresponding software (see below) and this manual). Some product versions may also supply additional components such as data cables or converters (see "[Optional Accessories](#)", page [657](#)).

Delivered Software

The delivered software package is identical to the RTC5 software package (see [page 22](#)). It contains drivers for Microsoft's Windows 10, 8, 7, Vista, XP SP2, XP SP3 and Windows XP Embedded (SP2 or higher) (32-bit or 64-bit) operating systems, correction, DLL and program files, as well as utility files for C, C++ and C#.

15.1.7 Optional Accessories

In addition to the RTC5 PC/104-Plus Board and its software package, the following accessories – as with the RTC5 PCI Board – can be obtained from SCANLAB (only hardware extensions from SCANLAB should be used in combination with the RTC5 PC/104-Plus Board):

- XY2-100 converter (see [page 29](#)).
- Data cables (see [page 29](#)).
- Laser adapter (see [page 29](#)).
- Slot cover with a 15-pin D-SUB connector for using the inputs and signals of the LASER connector (see [page 661](#))

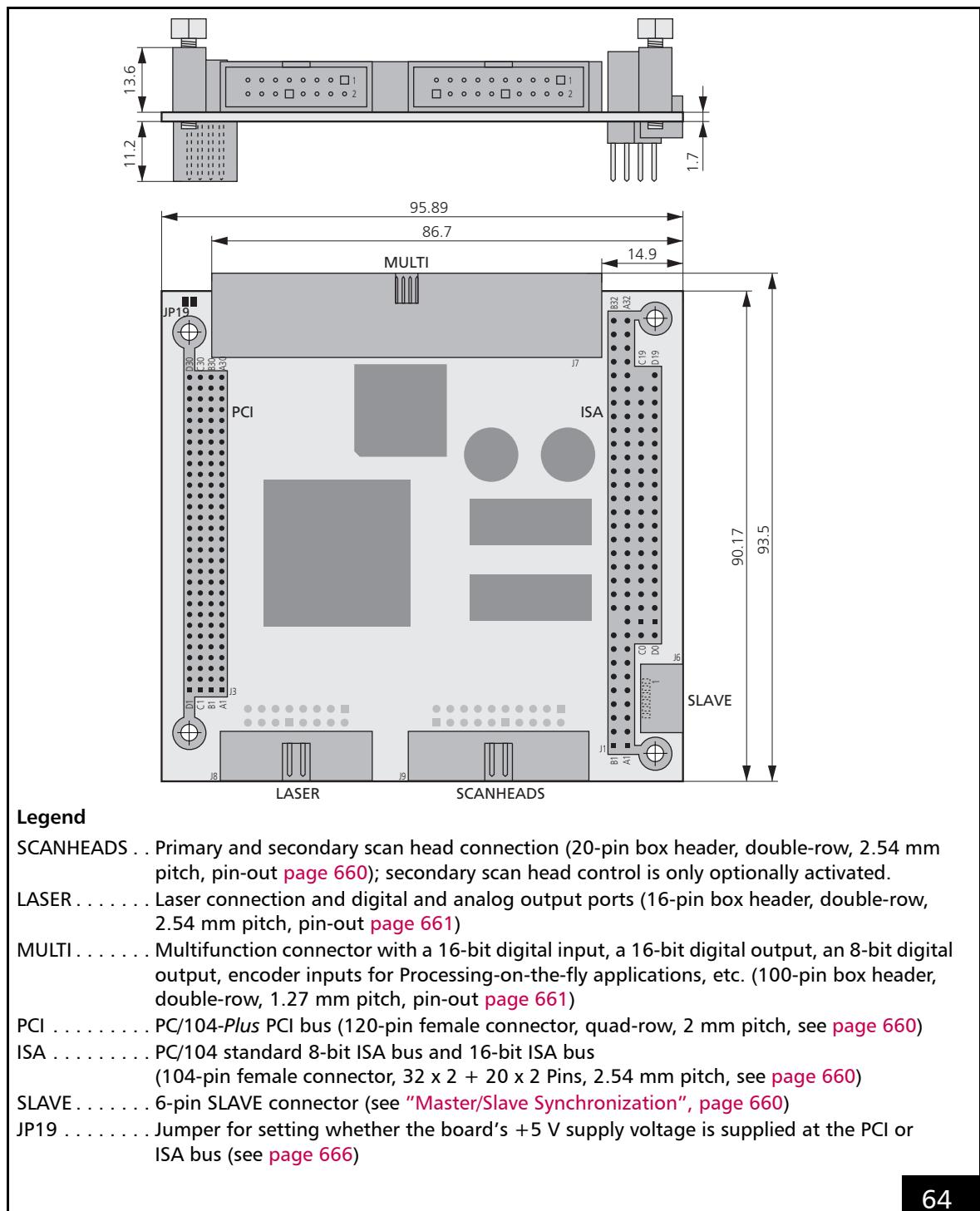
15.1.8 Supplementary Software

To facilitate customizing RTC correction files basing on your own test measurements, SCANLAB offers its correXion line of software and related documentation (see also [page 137](#)).

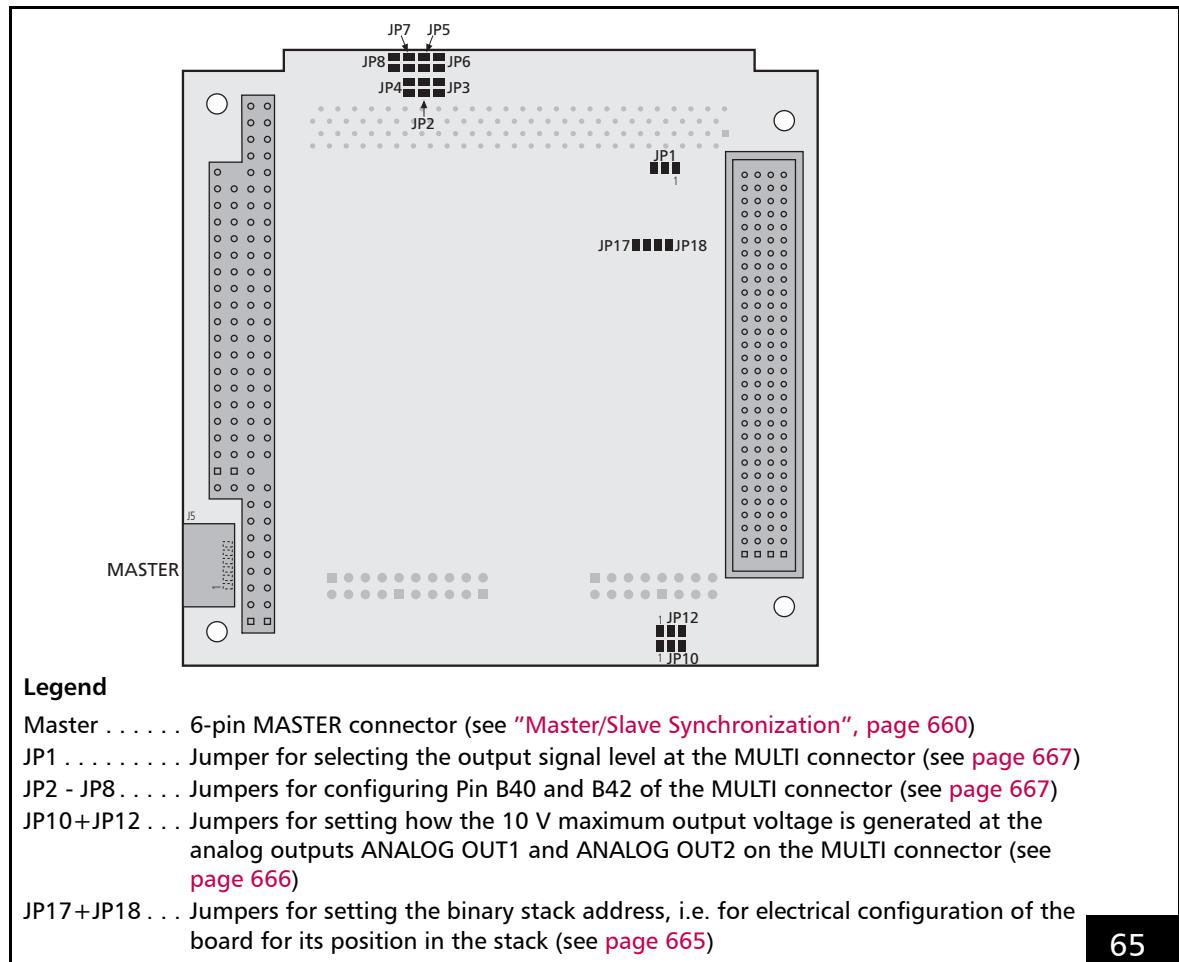
15.2 Layout and Interfaces

15.2.1 Connectors and Jumper Positions

Figure 64 and figure 65 show the positions of the connectors and jumpers on the front and back side of the RTC5 PC/104-Plus Board. All connectors and jumper settings are described in detail in the following sections.



Layout and dimensions of the RTC5 PC/104-Plus Board (top: side view, bottom: front side)



Layout of the RTC5 PC/104-Plus Board (back side)

15.2.2 Interface to the CPU board

Data transfer between the CPU board and the RTC5 PC/104-Plus Board is by the PCI bus. The RTC5 PC/104-Plus Board provides 120-pin PCI stack-through connectors (see [figure 64](#)). The timing and signal levels of the PCI signals are in compliance with PCI Local Bus Specification Revision 2.2. 64-bit extensions, JTAG, PRSNT and CLKRUN signals are not supported.

On the ISA bus, the RTC5 PC/104-Plus Board only uses the +5 V power provided at pins B3, B29 and D16 (for more power supply information, see also [page 666](#)). The other signals of the ISA bus are simply forwarded by the 104-pin ISA stack-through connectors to further boards in the stack and are not used by the RTC5 PC/104-Plus Board.



Caution!

- The RTC5 PC/104-Plus Board does not support power-saving modes that switch off power to the PCI bus. Accordingly, you must disable standby or sleep modes of the operating system. See also the note on [page 26](#).

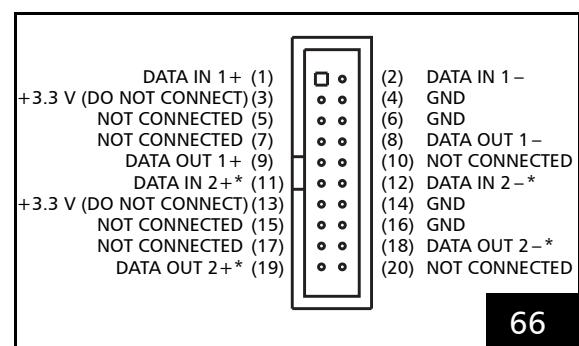
15.2.3 Master/Slave Synchronization

If multiple synchronously-clocked RTC5 PC/104-Plus Boards are to be used in a PC/104 stack, then the RTC5 PC/104-Plus Boards must be connected pairwise with each other by the MASTER and SLAVE connectors and installed in adjacent stack positions. Always connect a board's MASTER connector to the SLAVE connector of another board. Suitable connection cables are available from SCANLAB.

See also [chapter 6.6.3 "Master/Slave Operation"](#), [page 93](#) and ["Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization"](#), [page 239](#).

15.2.4 Interfaces to Scan System

For digitally controlling scan systems, the 20-pin SCANHEADS connector provides the primary scan head connection (pins 1 through 10) and the optionally activated secondary scan head connection (pins 11 through 20) (see [figure 64](#)). At this connector, scan-system control values are transmitted and scan-system status signals received. [Figure 66](#) shows the pin-out of the SCANHEADS connector.



66

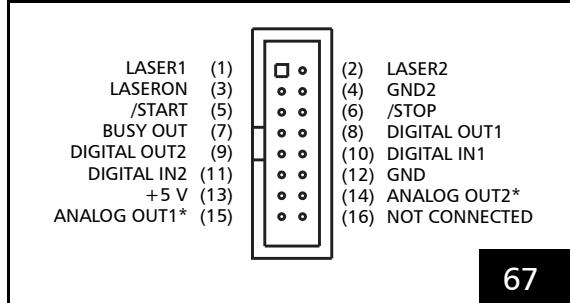
Pin-out of the SCANHEADS connector
(* Signal only optionally activated)

The SCANHEADS connector provides the same signals as the SCAN HEAD and 2. SCAN HEAD connectors of the RTC5 PCI Board. Scan system control (by an XY2-100 converter, if needed) is identical to that of the RTC5 PCI Board. Note the information in [chapter 4.3 "Interfaces to Scan System", page 42](#).

15.2.5 Interfaces for the Laser and Peripheral Equipment

LASER connector

Figure 67 shows the pin-out of the 16-pin LASER connector. This connector provides the same signals as the 15-pin D-SUB LASER connector of the RTC5 PCI Board. Note the information in "Laser Connector", page 47.



67

Pin-out of the 16-pin LASER connector
(* depends on jumper settings JP10 and JP12)

The connector provides the two analog output ports ANALOG OUT1 and ANALOG OUT2 (the ANALOG OUT2 signal is also available by the MULTI connector, see figure 68). Jumpers JP10+JP12 on the back side of the RTC5 PC/104-Plus Board allow setting how the 10 V maximum output voltage of these analog outputs is generated (see page 666).

A slot cover with a 15-pin D-SUB connector for using the inputs and signals of the LASER connector (with the same pin-out as the 15-pin D-SUB LASER connector of the RTC5 PCI Board, see figure 10) is available from SCANLAB.

MULTI connector

Figure 68 shows the pin-out of the 100-pin MULTI connector. This multifunction connector provides the same signals as the EXTENSION 1, EXTENSION 2, MARKING ON THE FLY, "RS232", "STEPPER MOTOR" and "SPI / I2C" socket connectors on the RTC5 PCI Board. Note the information in the following table.

(A1)	DIGITAL OUT0#
(A2)	DIGITAL IN0
(A3)	DIGITAL OUT1#
(A4)	DIGITAL IN1
(A5)	DIGITAL OUT2#
(A6)	DIGITAL IN2
(A7)	DIGITAL OUT3#
(A8)	DIGITAL IN3
(A9)	DIGITAL OUT4#
(A10)	DIGITAL IN4
(A11)	DIGITAL OUT5#
(A12)	DIGITAL IN5
(A13)	DIGITAL OUT6#
(A14)	DIGITAL IN6
(A15)	DIGITAL OUT7#
(A16)	DIGITAL IN7
(A17)	DIGITAL OUT8#
(A18)	DIGITAL IN8
(A19)	DIGITAL OUT9#
(A20)	DIGITAL IN9
(A21)	DIGITAL OUT10#
(A22)	DIGITAL IN10
(A23)	DIGITAL OUT11#
(A24)	DIGITAL IN11
(A25)	DIGITAL OUT12#
(A26)	DIGITAL IN12
(A27)	DIGITAL OUT13#
(A28)	DIGITAL IN13
(A29)	DIGITAL OUT14#
(A30)	DIGITAL IN14
(A31)	DIGITAL OUT15#
(A32)	DIGITAL IN15
(A33)	LATCH OUT#
(A34)	SYNC OUT#
(A35)	VCC OUT#
(A36)	BUSY OUT#
(A37)	+5 V
(A38)	+5 V
(A39)	GND
(A40)	GND
(A41)	CLKX0
(A42)	CLKR0
(A43)	FSX0
(A44)	FSR0
(A45)	DX0
(A46)	DRO
(A47)	+3.3 V
(A48)	GND
(A49)	RxD
(A50)	TxD

68

Pin-out of the 100-pin MULTI connectors
(* depends on JP10 and JP12 jumper settings)
\$ depends on JP2-JP8 jumper settings
Output signal level depends on JP1 jumper setting)

Pin at MULTI connector	Signal	Signal description	Connector on the RTC5 PCI Board
A1, A3, A5, A7, A9, A11, A13, A15, A17, A19, A21, A23, A25, A27, A29, A31	DIGITAL OUT0... DIGITAL OUT15	Buffered 16-bit digital output (output signal level either 5 V or 3.3 V, selectable by jumper JP1 , see page 667).	"EXTENSION 1 Socket Connector", page 51
A33	LATCH OUT	LATCH signal for synchronization of data transmission at the 16-bit digital output (output signal level either 5 V or 3.3 V, selectable by jumper JP1 , see page 667).	"EXTENSION 1 Socket Connector", page 51
A2, A4, A6, A8, A10, A12, A14, A16, A18, A20, A22, A24, A26, A28, A30, A32	DIGITAL IN0... DIGITAL IN15	16-bit digital input	"EXTENSION 1 Socket Connector", page 51
A34	SYNC OUT	SYNC signal for synchronization of data transmission at the 16-bit digital input (output signal level either 5 V or 3.3 V, selectable by jumper JP1 , see page 667).	"EXTENSION 1 Socket Connector", page 51
A36	BUSY OUT	BUSY status as BUSY OUT signal (output signal level either 5 V or 3.3 V, selectable by jumper JP1 , see page 667).	"EXTENSION 1 Socket Connector", page 51
A35	VCC OUT	Output signal level selected by jumper JP1 (5 V or 3.3 V).	"EXTENSION 1 Socket Connector", page 51
B26, B28, B30, B32, B34, B36, B38 (B40, B42)	DATA0...DATA6 (DATA7)	Buffered 8-bit digital output port (DATA7 optionally at B40, B42, dependent on the JP2-JP8 jumper configuration, see page 667)	"EXTENSION 2 Socket Connector", page 52
(B42)	(LATCH)	LATCH signal for synchronization of data transmission at the 8-bit digital output port (optionally at B42, dependent on the JP2-JP8 jumper configuration, see page 667)	"EXTENSION 2 Socket Connector", page 52
B44, B47, B48	LASER2, LASER1, GND2	The laser control signals LASER1 and LASER2 are referenced to GND2. If the RTC5 PC/104-Plus Board is equipped with optional optocouplers, then GND2 and the laser output signals are galvanically decoupled from CPU GND. Otherwise, GND2 are GND identical	"EXTENSION 2 Socket Connector", page 52
B3 - B10	ENCODER X1/2± ENCODER Y1/2±	Encoder inputs	"MARKING ON THE FLY Socket Connector", page 53
B11, B12	/STOP2, /START2	External stop signal, External start signal	"MARKING ON THE FLY Socket Connector", page 53



Pin at MULTI connector	Signal	Signal description	Connector on the RTC5 PCI Board
B13	BUSY OUT	BUSY status as BUSY OUT signal (5 V output signal level)	"MARKING ON THE FLY Socket Connector", page 53
B14	ANALOG OUT2	12-bit analog output port (the signal is also available by the LASER connector – see page 661. Jumpers JP10+JP12 on the back side of the RTC5 PC/104-Plus Board allow setting how the 10 V maximum output voltage is generated – see page 666).	"MARKING ON THE FLY Socket Connector", page 53 and "Laser Connector", page 47
A49, A50	RxD, TxD	RS232 interface	"RS232" (see page 54)
A41 - A46	CLKX0, CLKR0, FSX0, FSR0, DX0, DR0	McBSP/SPI interface	"SPI / I2C" (see page 54)
B16	SWITCH1/2, ENABLE1/2, DIRECTION1/2, CLOCK1/2	Signals for controlling two stepper motors	"STEPPER MOTOR" (see page 58)
A37, A38, B1 (B40, B42)	+5 V	5 V output signal level (optionally at B40, B42, dependent on the JP2-JP8 jumper configuration, see page 667)	EXTENSION 1, EXTENSION 2, MARKING ON THE FLY
A47	+3.3 V	3.3 V output signal level	"SPI / I2C"
A39, A40, A48, B2, B15, B24, B25, B43, B50 (B40, B42)	GND	CPU ground (optionally at B40, B42, dependent on the JP2-JP8 jumper configuration, see page 667)	EXTENSION 1, EXTENSION 2, MARKING ON THE FLY, "RS232", "STEPPER MOTOR", "SPI / I2C"
B45, B46		Internally connected with each other	EXTENSION 2, "RS232"

15.3 Installation and Start-Up

Installation of the RTC5 PC/104-Plus Board consists of the following steps:

- (1) Configuring the jumpers
- (2) Installing the RTC5 PC/104-Plus Board
- (3) Installing the drivers
- (4) Installing the RTC5 software

Proceed in the order described in the following sections.

The included HPGL converter program can be used for conducting a post-installation functionality test.

15.3.1 Safety During Installation and Operation

Read these instructions carefully before you proceed with installing and operating the RTC5 PC/104-Plus Board. In particular, observe all safety guidelines in this manual. Also note the information in [chapter 3 "Safety During Installation and Operation", page 39](#).

If there are any questions regarding the contents of this manual, please contact SCANLAB.



Caution!

- Carefully check your user program before running it. Programming errors can cause a break down of the system. In this case neither the laser nor the scan system can be controlled.
- Protect the board from humidity, dust, corrosive vapors and mechanical stress.
- For storage and operation, avoid electro-magnetic fields and static electricity. These can damage the electronics on the RTC5 PC/104-Plus Board. For storage, always use the antistatic bag the board is delivered in.
- The allowed operating temperature range is 15 °C to 60 °C.
- The storage temperature should be between –20 °C and +60 °C.

15.3.2 Jumper Settings

Before installing the RTC5 PC/104-*Plus* Board in a PC/104 stack, you might need to configure the board's jumpers. SCANLAB ships RTC5 PC/104-*Plus* Boards in various jumper configurations (see also "[Type Identification](#)", page 656). The jumpers are soldered junctions and customers can subsequently reconfigure them by using a soldering iron. If you do not want to change the factory settings, proceed with "[Installing the RTC5 PC/104-Plus Board](#)", page 668.



Caution!

- When altering a jumper configuration, take care not to damage the board's electronics!

Jumper 17+18: Setting the Binary Stack Address

Jumpers **JP17+JP18** on the back side of the RTC5 PC/104-*Plus* Board (see [figure 65](#)) allow the board's stack address to be defined. You can thereby electrically customize the board to its position in the PC/104 stack. SCANLAB recommends assigning stack addresses as follows (for an overview of PCI stack positions, see [figure 70](#)):

PCI stack position	stack address	JP17	JP18
1	0 *	closed	closed
2	1	closed	open
3	2	open	closed
4	3	open	open

* default configuration

Stack address configuration ensures that each RTC5 PC/104-*Plus* Board in a stack gets to have one of the four CLK, IDSEL, REQ and GNT signals of the PCI bus explicitly assigned to it:

Stack address	CLK	IDSEL	REQ	GNT
0	CLK0	IDSEL0	REQ0	GNT0
1	CLK1	IDSEL1	REQ1	GNT1
2	CLK2	IDSEL2	REQ2	GNT2
3	CLK3	IDSEL3	REQ3	GNT3

Selection of the REQ and GNT signals is only relevant for bus master boards. RTC5 PC/104-*Plus* Boards are target devices and therefore not dependent on master stack positions.

A configured stack address corresponding to the PCI stack position ensures optimal relationships between the signal path lengths (which differ for each PCI stack position). This results in stable PCI-bus transfer of assigned signals.

Notes

- The RTC5 PC/104-*Plus* Board's stack address is factory-set to 0. If you install the board directly onto a CPU board (without other modules in between), then jumpers **JP17+JP18** do not need to be changed.

JP19: +5 V Power Supply

The RTC5 PC/104-Plus Board requires a +5 V power supply. This is not available from all CPU boards on the PCI bus, but is almost always available on the ISA bus (if present). Therefore, the delivered RTC5 PC/104-Plus Board's default configuration is for 5 V power provided exclusively by the ISA bus (**JP19** open).

If a CPU board has no ISA bus, or if the ISA bus is not connected to the CPU board, then Jumper **JP19** (on the front side of the RTC5 PC/104-Plus Board, see [figure 64](#)) can be rejumpered so that power is supplied by the PCI bus, assuming that a +5 V supply voltage is available there (be sure to check your CPU board's corresponding specifications).

JP19	+5 V power supply
open *	by the ISA bus
closed	by the PCI bus (only if the ISA bus is <i>not</i> available)

* default configuration



Caution!

- **JP19** must *not* be closed if the RTC5 PC/104-Plus Board connects to the CPU board by both the PCI and ISA busses and the CPU board supplies +5 V at both busses. Power provided at both busses might damage the RTC5 PC/104-Plus Board.

JP10+12: 10 V Output Voltage for ANALOG OUT1 and ANALOG OUT2

Jumpers **JP10+JP12** on the back side of the RTC5 PC/104-Plus Board (see [figure 65](#)) allow setting how the 10 V maximum output voltage is generated at analog output ports ANALOG OUT1 and ANALOG OUT2 on the LASER and MULTI connectors:

Jumpers JP10+JP12	10 V Output Voltage
Position 1-2 	derived from the ±12 V supply of the PCI bus
Position 2-3* 	derived from the +5 V supply of the PCI/ISA bus
open 	not provided

* default configuration



Caution!

- Ensure that **JP10** and **JP12** are not jumpered differently (e.g. do not set **JP10** for position 1-2 and **JP12** for position 2-3). Otherwise, the electronics of the board is damaged.

PCI Local Bus Specification Revision 2.2 requires that the PCI bus (pins A29 and A30) provide a ±12 V supply. In reality, not all CPU boards supply this voltage. Therefore, the RTC5 PC/104-Plus Board is factory configured so that the 10 V maximum output voltage is derived from the +5 V supply voltage (provided either by the PCI bus or the ISA bus, see "[JP19: +5 V Power Supply](#)") (**JP10** and **JP12** are both set for position 2-3). In this configuration, the maximum output current is +5 mA (source only).

If the CPU board actually provides ±12 volts (check the corresponding specifications of your CPU board), then rejumper as follows: both jumpers (**JP10** and **JP12**) in position 1-2. Then the maximum output current is ±5 mA (source and sink).

Jumper JP1: Selecting the Output Signal Level at the MULTI connector

By Jumper **JP1** on the back side of the RTC5 PC/104-Plus Board (see [figure 65](#)), the level of the MULTI connector's 16-bit digital output (pins A1, A3, A5, A7, A9, A11, A13, A15, A17, A19, A21, A23, A25, A27, A29, A31), LATCH signal (pin A33), SYNC signal (pin A34), BUSY OUT signal (pin A36) and VCC_OUT signal (pin A35) can be configured for 5 V or 3.3 V (see also [page 661](#)). This corresponds to signal level selection at the EXTENSION 1 socket connector of the RTC5 PCI Board (see [page 51](#)).

Jumper JP1	Signal Level
Position 1-2	5 V
Position 2-3	3.3 V
open	no output signals



Caution!

- Be sure that not all three jumper pins are closed. Otherwise, the electronics of the board is damaged.

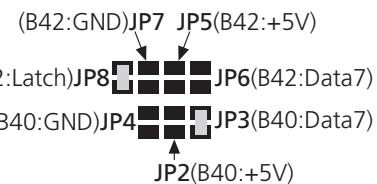
SCANLAB's RTC5 PC/104-Plus Board is supplied in various jumper configurations for **JP1** (see also "[Type Identification](#)", [page 656](#)).

Jumper JP2-JP8: Configuring Pin B40 and Pin B42 at the MULTI connector

Pins B40 and B42 of the MULTI connector can be configured by jumpers **JP2-JP8** on the back side of the RTC5 PC/104-Plus Board (see [figure 65](#) and [page 661](#)). This corresponds to the configuration possibilities at pins (15) and (17) of the RTC5 PCI Board EXTENSION 2 socket connector (see [page 52](#)).

The most significant bit (DATA7) of the 8-bit output value can be assigned to pin B40 or pin B42. Alternatively, each of the two pins can be set permanently to +5 V (HIGH) or to GND (LOW level). Alternatively, pin B42 can be configured for the LATCH signal by closing the correspondingly labeled jumper.

[Figure 69](#) shows an example jumper configuration assigning DATA7 to pin B40 and the LATCH signal to pin B42.



69

Example configuration for jumpers JP2-JP8:
Pin B40: DATA7, Pin B42: LATCH signal



Caution!

- Make sure that not more than **one** of the three jumpers for pin B40 is closed. Also make sure that not more than **one** of the four jumpers for pin B42 is closed.
Closing more than one jumper for pin B40 or B42 damages the electronics of the board.

SCANLAB's RTC5 PC/104-Plus Board is supplied in various jumper configurations for **JP2-JP8** (see also "[Type Identification](#)", [page 656](#)).

15.3.3 Installing the RTC5 PC/104-Plus Board

Figure 70 shows a typical module stack (PC/104 stack) consisting of a CPU board, an RTC5 PC/104-Plus Board, a further PC/104-Plus module, a PC/104 16-bit module and a PC/104 8-bit module.

In general, these modules provide stack-through connectors to pass the signals on for possible use by the next module in a stack. Thus, you can install modules above or below other modules as well as on both sides of the CPU board.

PC/104-Plus modules are equipped with both PCI and ISA busses. In contrast, PC/104 standard modules are only equipped with an ISA bus and might therefore interrupt PCI data transfer within a stack. As RTC5 PC/104-Plus Board's data transfer is by the PCI bus, PC/104 standard modules in the stack must be moved to higher stack positions.

The maximum stack height for the PCI bus is four PC/104-Plus modules (plus the CPU board). The modules must be adjacently stacked on the same side of the CPU board. Electrical configuration of a board for its position in a stack can be specified by jumpers (see [page 665](#)).

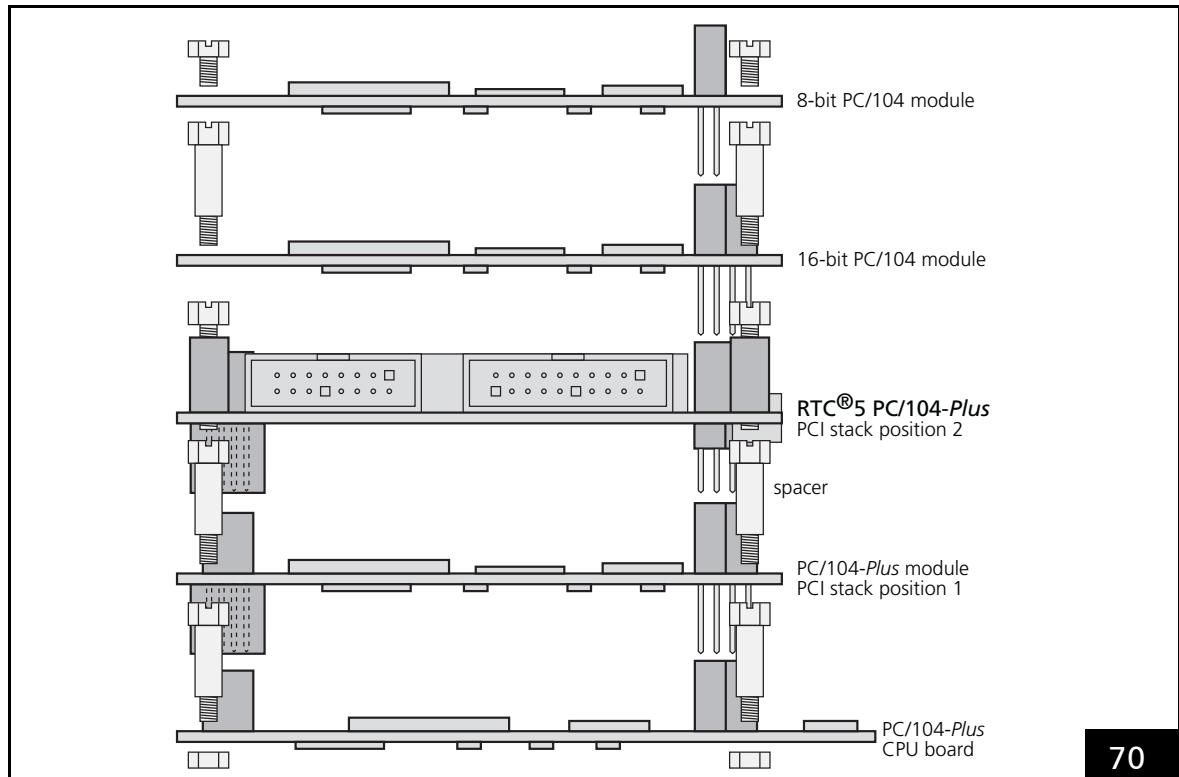
If multiple synchronously-clocked RTC5 PC/104-Plus Boards are to be used in a PC/104 stack, then the RTC5 PC/104-Plus Boards must be installed in adjacent stack positions.

For mechanical stability, all modules and the CPU board are attached to each other by screws or spacer bolts.



Caution!

- If several PCI boards in a PC/104-Plus stack are operated by a PC/104-Plus CPU board, then you should test the overall configuration (i.e. the functionality of all boards in the stack). Some CPU boards can create problems on the PCI bus. To test the functionality of an RTC5 PC/104-Plus Board in a stack, you should at least check if **rtc5_count_cards** returns the correct number of boards.



Typical module stack



Caution!

- Carry out the installation in an area that complies with Electro Static Discharge (ESD) directives. When installing the board, also comply with instructions supplied in your controller's user manual.
- Do not touch the contacts of the board.
- Protect the board from humidity, dust, corrosive vapors and mechanical stress.

Perform the following steps to install the RTC5 PC/104-Plus Board in a PC/104 stack:

- ▶ Shut down the operating system and switch off the controller. Disconnect the controller from the mains.
- ▶ Disconnect all peripherals (e.g. monitor, mouse, keyboard, printer etc.) from the controller.
- ▶ Place the controller on a work table that complies with ESD directives.
- ▶ Remove the housing of the controller.
- ▶ Check whether a PC/104-Plus connector is available on the CPU board and check the stack height (a PCI stack can contain up to four PC/104-Plus modules).
- ▶ Remove the RTC5 PC/104-Plus Board from the antistatic bag. Do not touch the contacts of the board.
- ▶ If you want to use the signals on the RTC5 PC/104-Plus Board's SCANHEADS, LASER and/or MULTI connectors, then attach the appropriate cables (connection cables should only be attached to equipment after the RTC5 PC/104-Plus Board is installed). All RTC5 connectors and signals are described in [chapter 15.2 "Layout and Interfaces", page 658](#).
- ▶ Insert the RTC5 PC/104-Plus Board in the desired position (1 to 4) within the PC/104 stack:
 - The position in the stack should correspond to the stack address set by jumpers **JP17+JP18** (see [page 665](#)). Also check the other jumper settings.
 - Check if components on the RTC5 PC/104-Plus Board collide with components on the CPU board (or a neighboring module in the stack).
 - If multiple synchronously-clocked RTC5 PC/104-Plus Boards are to be used in a PC/104 stack, then the RTC5 PC/104-Plus Boards must be connected pairwise with each other by the MASTER and SLAVE connectors and installed in adjacent stack positions (see also "[Master/Slave Synchronization](#)", [page 660](#)).
 - Check the above information on stacking PC/104-Plus modules.
 - Check that the board's PCI and ISA connectors are correctly connected to the CPU board or the neighboring modules in the stack.
 - Use screws or spacer bolts to attach the board to the CPU board or neighboring modules in the stack.
- ▶ Close the housing of the controller.
- ▶ Place the controller at its operational location and connect all peripheral equipment.
- ▶ If necessary, connect one or more of the RTC5 PC/104-Plus Board's connectors to the scan system (using the XY2-100 converter, if necessary), to the laser, a handling system or other supplementary equipment of the overall system.
- ▶ Check all connections and turn on the controller.

15.3.4 Installing the Drivers and the RTC5 Software

RTC5 PC/104-Plus Boards use the same drivers and other software files as the RTC5 PCI Board. Installation is also identical to the RTC5 PCI Board (see [page 61](#) and [page 62](#)).

Notes

- Windows runs the RTC5 PC/104-Plus Board as a PCI board.
- To install the driver, make sure you are logged on with administrator rights (not necessary when later using the RTC5 PC/104-Plus Board).



Caution!

- The RTC5 PC/104-Plus Board does not support power-saving modes that switch off power to the PCI bus. Accordingly, you must disable standby or sleep modes of the operating system. Particularly disable the Windows sleep mode option (some Windows operating systems enable this option by default). See also the note on [page 26](#).

15.3.5 Functionality Test, User Programs and Demo Software

The HPGL conversion program HPGL.EXE is supplied for testing control of the scan system (see [page 63](#)). Developing user programs is identical to that of the RTC5 PCI Board (see [page 64](#)). You can run the same demo programs used for the RTC5 PCI Board.



Danger!

- Always turn on the controller with RTC5 PC/104-Plus Board and the power supply for the scan system first before turning on the laser. Otherwise, there is the danger of uncontrolled deflection of the laser beam.



Caution!

- Carefully check your user program before running it. Programming errors can cause a system breakdown. In this case, neither the laser nor the scan head can be controlled.



15.4 Technical Specifications of RTC5 PC/104-Plus Board

		Connectors to PC/104-Plus CPU board (or to PC/104-Stack)	
System Requirements		PC/104-Plus PCI bus	120-pin stack-trough connectors incl. shroud, operating voltage: +5 V
PC/104-Plus CPU board + free position in the PC/104 stack		ISA bus	104-pin stack-through connectors incl. socket for passing through the signals and +5 V power supply
Operating system	Microsoft (32 bit or 64 bit) Windows 10, 8, 7, Vista, XP SP2, XP SP3 and Windows XP Embedded ≥ SP2		
Dimensions		Other connectors (signal specifications identical to RTC5 PCI Board)	
Length x Width	96 mm x 93.5 mm	Interface to scan system	SCANHEADS connector (20-pin box header) with primary and secondary scan head connection (secondary scan head control is only optionally activated)
Stack height	16.8 mm		
Max. component height	9.3 mm	Interfaces for laser and peripheral equipment	LASER connector (16-pin box header) MULTI multifunction connector (100-pin box header)
Weight	approx. 115 g		
Operating and Storage Conditions			
Operating temperature	+15 °C to +60 °C		
Storage temperature	–20 °C to +60 °C		
Environmental conditions	non-condensing non-corrosive		

16 Appendix B: The RTC5 PCI Express Board

16.1 Product Overview

16.1.1 Intended Use – Comparison to the RTC5 PCI Board

Just like the RTC5 PCI Board, the SCANLAB RTC5 PCI Express Board is intended for synchronous real-time control of scan systems, lasers and peripheral equipment. It differs from the RTC5 PCI Board in the following hardware-related details:

- The RTC5 PCI Express Board provides a PCI-Express interface and must be connected to a PCI-Express slot of the PC.
- The “SPI / I²C” connector provides two 10 V analog inputs instead of an I²C interface (see [page 676](#)).

Otherwise, the RTC5 PCI Express Board provides the same functionality for controlling scan systems, lasers and peripheral equipment as the RTC5 PCI Board. Neither the number, type and positioning of connectors for controlling scan systems, lasers and peripheral equipment nor the jumpers for customized configuration differ from that of the RTC5 PCI Board. Specifications for signal inputs and outputs are the same as with the RTC5 PCI Board (with above-mentioned exception), as are software installation and developing user programs. Both the RTC5 PCI Express Board and the RTC5 PCI Board use the same software package (driver, DLL, import declarations, etc.) and same command set.

16.1.2 System Requirements

Hardware

The RTC5 PCI Express Board requires a Windows PC with a PCI-Express bus interface and at least one free PCI-Express slot. The dimensions of the RTC5 Board are shown in [figure 71](#).

RTC5 PCI Express Boards intended for synchronized master/slave operation must be installed in adjacent PCI-Express slots.

Software

The RTC5 PCI Express Board can use the same drivers and DLL files as the RTC5 PCI Board (about the supported operating systems see [section 2.2.2 on page 26](#)).



Caution!

- The RTC5 PCI Express Board does not support power-saving modes that switch off power to the PCI-Express bus. Accordingly, you must disable standby or sleep modes of the operating system. See also the note on [page 26](#).

16.1.3 Optional Functionality

The RTC5 PCI Express Board can be configured for the same functionality as the RTC5 PCI Board (see [page 27](#)). Optional functionality must be enabled by SCANLAB or installed.



16.1.4 Labeling

The serial number of the RTC5 PCI Express Board is printed on a white label attached to the board (format: "RTC SN...").

The board's ID number and configuration are described in the packaging list (see also "[Accessories](#)", [page 29](#) and "[Type Identification](#)", [page 656](#)).

16.1.5 Type Identification

The ID number of the RTC5 PCI Express Board reveals the board's factory-equipped jumper configuration (**JP1-JP8**). The jumper configuration is additionally encoded in a three-digit type code scheme. The type code scheme is identical to that of the RTC5 PCI Board (see [page 28](#)).

16.1.6 Unpacking Instructions and Typical Package Contents

- ▶ Carefully remove the RTC5 PCI Express Board from the package.
- ▶ Keep the packaging, including the antistatic bag the RTC5 PCI Express Board is delivered in, so that in case of repair the board can be properly repackaged and returned to SCANLAB.
- ▶ Also remove all other articles from the package. Check that all parts have been delivered. Refer to the corresponding packaging list.
The package typically includes an RTC5 PCI Express Board and a data CD (containing the corresponding software (see below) and this manual). Some product versions may also supply additional components such as data cables or converters (see "[Optional Accessories](#)", [page 657](#)).

Delivered Software

The delivered software package is identical to the RTC5 software package (see [page 22](#)). It contains drivers for Microsoft's Windows 10, 8, 7, Vista, XP SP2, XP SP3 (32-bit or 64-bit) operating systems, correction, DLL and program files, as well as utility files for C, C++ and C#.

16.1.7 Optional Accessories

In addition to the RTC5 PCI Express Board and its software package, the following accessories – as with the RTC5 PCI Board – can be obtained from SCANLAB (only hardware extensions from SCANLAB should be used in combination with the RTC5 PCI Express Board):

- XY2-100 converter (see [page 29](#)).
- Data cables (see [page 29](#)).
- Laser adapter (see [page 29](#)).
- Slot cover with a 9-pin D-SUB connector for using the 2. SCAN HEAD socket connector (see [page 43](#)).
- Slot cover with a 15-pin D-SUB connector for using the MARKING ON THE FLY socket connector (see [page 53](#)).

16.1.8 Supplementary Software

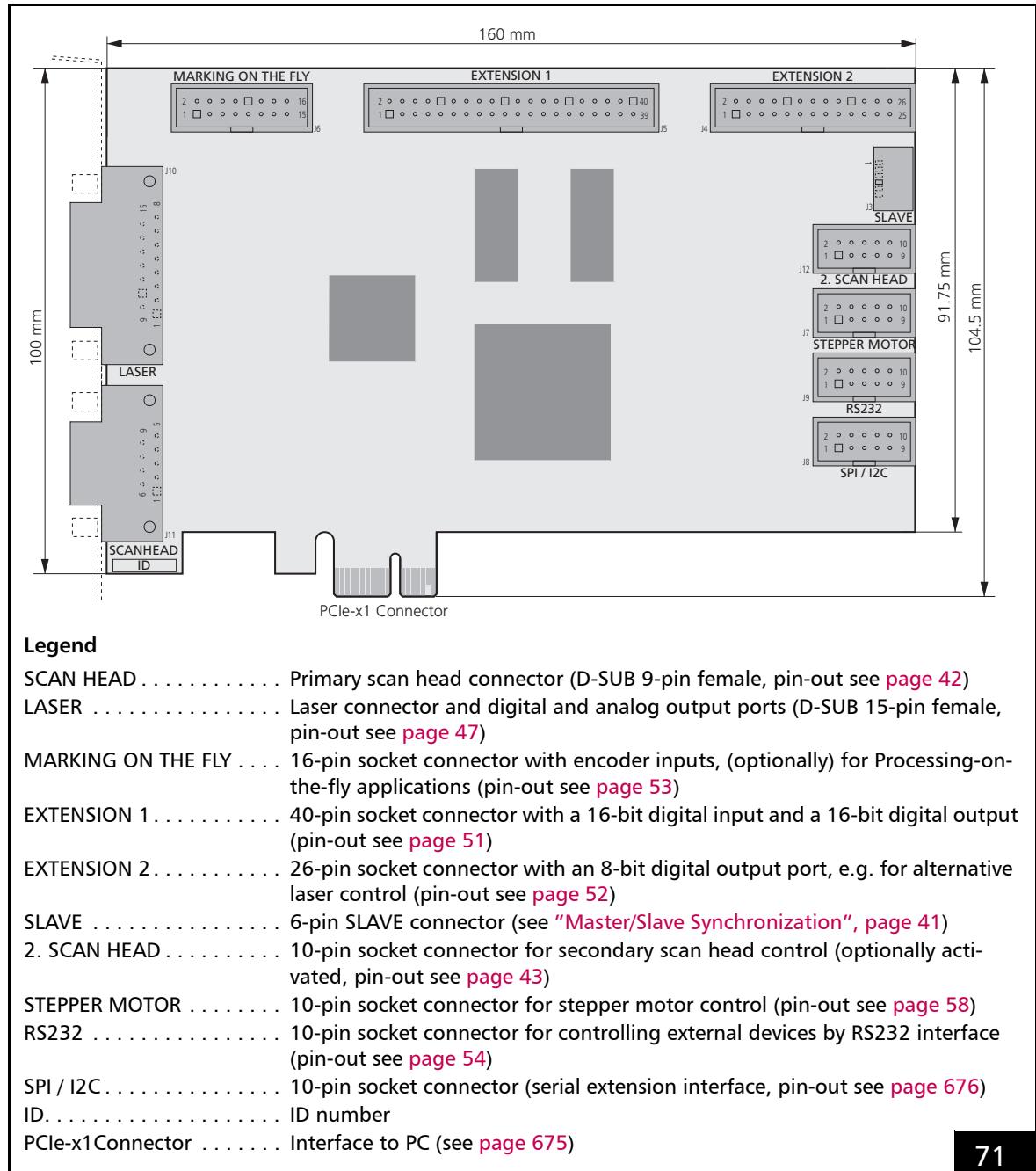
To facilitate customizing RTC correction files basing on your own test measurements, SCANLAB offers its correXion line of software and related documentation (see also [page 137](#)).

16.2 Layout and Interfaces

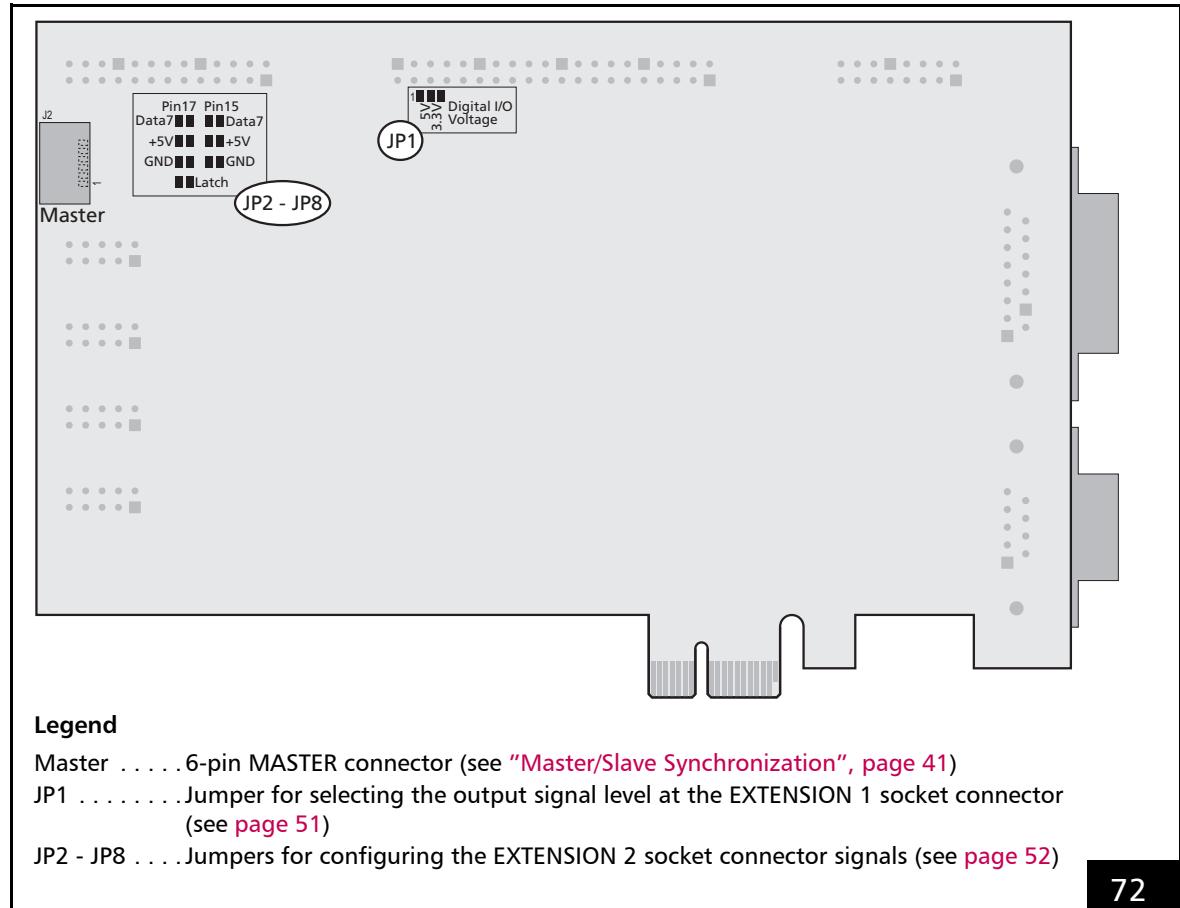
16.2.1 Connectors and Jumper Positions

Figure 71 and **figure 72** show the positions of the connectors and jumpers on the front and back side of the RTC5 PCI Express Board.

Relating the connectors and jumper settings, the RTC5 PCI Express Board differs from the RTC5 PCI Board only in its interface to the PC and in its configuration of the "SPI / I2C" socket connector. These two deviations from the RTC5 PCI Board are described in the following sections.



Layout of the RTC5 PCI Express Board (front side)



Layout of the RTC5 PCI Express Board (back side)

72

16.2.2 Interface to PC

The RTC5 PCI Express Board’s PCIe-x1 connector is the interface to the PC.

The RTC5 PCI Express Board can be installed into any Windows PC with a PCI-Express bus interface and at least one free PCI-Express slot.



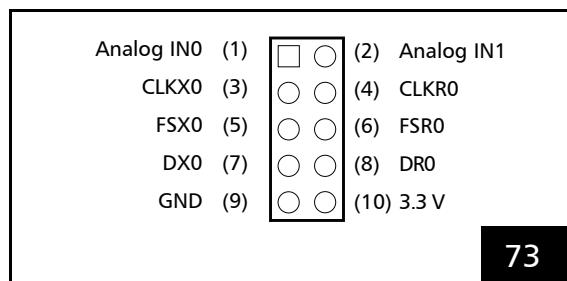
Caution!

- The RTC5 PCI Express Board does not support power-saving modes that switch off power to the PCI-Express bus. Accordingly, you must disable standby or sleep modes of the operating system. See also the note on [page 26](#).

16.2.3 McBSP/SPI Interface and Analog Inputs

The "SPI / I²C" socket connector provides an McBSP/SPI interface (Multi channel Buffered Serial Port) and two analog inputs.

The pin-out is shown in [figure 73](#).



Pin-out of the (on-board) "SPI / I²C" socket connector

16.3 Installation and Start-Up

For installing and starting-up an RTC5 PCI Express Board and its software proceed as with an RTC5 PCI Board (see [page 60](#)).



Caution!

- The RTC5 PCI Express Board does not support power-saving modes that switch off power to the PCI-Express bus. Accordingly, you must disable standby or sleep modes of the operating system. Particularly disable the Windows sleep mode option (some Windows operating systems enable this option by default). See also the note on [page 26](#).

McBSP/SPI Interface

The McBSP/SPI interface and the corresponding commands are identical to that of the RTC5 PCI Board (see [page 54](#)).

Analog Inputs

The RTC5 PCI Express Board provides two 10 V analog inputs (ANALOG IN0 and ANALOG IN1) at the "SPI / I²C" socket connector (instead of the I²C interface which is provided by the RTC5 PCI Board; similar to the RTC5 PCI Board's optional ADC add-on board).

For using these analog inputs and for their specifications, refer to [chapter 4.4.8 "Analog Inputs \(Optional Accessory\)"](#), page 59.

16.4 Technical Specifications of the RTC5 PCI Express Board

System Requirements

Windows PC with PCI-Express bus interface

Operating system	Microsoft Windows 10, 8, 7, Vista, XP SP2, XP SP3 as 32-bit or 64-bit version
------------------	---

Dimensions

Length	160 mm
Height	104.5 mm

Interface to Scan System

PCI-Express x1, version 1.0

"SPI / I2C" Socket Connector

Connector	10-pin socket connector
McBSP/SPI interface	identical to RTC5 PCI Board (see page 54)

2 Analog inputs (ANALOG IN0, ANALOG IN1):

- Input voltage range 0 V ... 10 V
- Input resistance 4 kΩ
- ADC resolution 12 bit
- Reference GND

Other Connectors and Specifications

Identical to RTC5 PCI Board

16.5 Compliance with EC Guidelines for Electromagnetic Compatibility (EMC)

The RTC5 PCI Express Board has been determined to be in compliance with EC directive 2004/108/EC (electromagnetic compatibility).

For that purpose, an RTC5 PCI Express Board has been integrated to a PC and was tested together with a varioSCAN_{de} 40i Flex scan head (with SL2-100 interface).

Test Specifications

Evidence of fulfillment of the protection goals of EC directive 2004/108/EG (CE Conformity for EMC) based on

- EN 61000-6-2: 2005
- EN 61000-6-4: 2007 + A1:2011

Result

The devices under test fulfill the specifications.

16.6 Compliance with FCC Rules

The RTC5 PCI Express Board has been tested and found to comply with the limits for a Class A digital device, pursuant to part 15 of the FCC rules.

These limits are designed to provide reasonable protection against harmful interference when the RTC5 PCI Express Board is operated in a commercial environment. The RTC5 PCI Express Board generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with this instruction manual, may cause harmful interference to radio communications. Operation of the RTC5 PCI Express Board in a residential area is likely to cause harmful interference in which case the user is required to correct the interference at his own expense.



17 Appendix C: The RTC5 PCIe/104 Board

17.1 Product Overview

17.1.1 Intended Use – Comparison to the RTC5 PCI Board

Just like the RTC5 PCI Board, the SCANLAB RTC5 PCIe/104 Board is intended for synchronous real-time control of scan systems, lasers and peripheral equipment. It differs from the RTC5 PCI Board in the following hardware-related details:

- The RTC5 PCIe/104 Board is a PCIe/104 module. Unlike an RTC5 PCI Board, it is not installed in a PC slot. Instead, it connects by its PCI-Express connectors (directly or indirectly) to a PCIe/104 CPU board. Up to four RTC5 PCIe/104 Boards can be operated with a CPU board.
- The number, type and positioning of connectors for controlling scan systems, lasers and peripheral equipment differ from that of the RTC5 PCI Board.

Otherwise, the RTC5 PCIe/104 Board provides the same functionality for controlling scan systems, lasers and peripheral equipment as the RTC5 PCI Board. Specifications for signal inputs and outputs are the same as with the RTC5 PCI Board as are software installation and developing user programs. Both the RTC5 PCIe/104 Board and the RTC5 PCI Board use the same software package (driver, DLL, import declarations, etc.) and same command set.

17.1.2 Comparison to the RTC5 PC/104-Plus Board

The RTC5 PCIe/104 Board is equipped with the same connectors for controlling scan systems, lasers and peripheral equipment as the RTC5 PC/104-Plus Board. It differs from the RTC5 PC/104-Plus Board in the following hardware-related details:

- For connecting to a CPU board or in the stack, the RTC5 PCIe/104 Board provides (only) a PCI Express connector instead of PCI and ISA connectors.
- The RTC5 PCIe/104 Board does *not* provide additional jumpers (for customized configuration with respect to both the power supply and the position in the PCIe/104 stack).
 - PCIe/104 modules automatically assign the stack address.
 - The RTC5 PCIe/104 Board does not have an ISA interface and related possibilities for customizing power supply voltages.
 - The 10 V maximum output voltage at analog output ports ANALOG OUT1 and ANALOG OUT2 on the LASER and MULTI connectors is derived from the +5 V supply of the PCI Express bus.
- The RTC5 PCIe/104 Board can be optionally equipped with an additional MARKING ON THE FLY connector (not included in the standard version).

17.1.3 System Requirements

Hardware

The RTC5 PCIe/104 Board is a PCIe/104 module with a corresponding form factor. The dimensions of the RTC5 PCIe/104 Board are shown in [figure 74](#) (note that the 100-pin MULTI connector projects slightly beyond the RTC5 PCIe/104 Board's PCB edge). The RTC5 PCIe/104 Board provides PCI-Express connectors and can be directly or indirectly (in a stack of several PCIe/104 modules) connected to a PCIe/104 CPU board.

Data transfer is by the board's PCI-Express bus in compliance with PCI-Express Base Specification Revision 2.0 and with PCI Local Bus Specification Revision 2.3. Stacks of up to four PCIe/104 modules (plus the PCIe/104 CPU board) can be created.

RTC5 PCIe/104 Boards intended for master/slave synchronization must be stacked adjacently.

To supply power for the RTC5 PCIe/104 Board, the CPU board must provide at least +5 V at the PCI-Express bus.

Software

The RTC5 PCIe/104 Board can use the same drivers and DLL files as the RTC5 PCI Board. Additionally to the operating systems listed in [chapter 2.2.2 "Software", page 26](#) for the RTC5 PCI Board, the RTC5 drivers and RTC5 DLL files also support Microsoft Windows XP Embedded operating system (i.e. the 32-bit and 64-bit versions of Windows XP Embedded ≥ SP2 together with RTC5-software-package version 2011_09_29 or later or DLL version ≥ DLL 528). Up to four RTC5 PCIe/104 Boards can be operated by a CPU board.



Caution!

- The RTC5 PCIe/104 Board does not support power-saving modes, which switch off the power supplies of the PCI-Express bus. According standby or sleep modes of the operating system must be deactivated.

17.1.4 Optional Functionality

The RTC5 PCIe/104 Board can be configured for the same functionality as the RTC5 PCI Board (see [page 27](#)). Optional functionality must be enabled by SCANLAB or installed.

17.1.5 Optional Connector

The RTC5 PCIe/104 Board can be optionally equipped with an additional MARKING ON THE FLY connector (not included in the standard version).

17.1.6 Labeling

The serial number of the RTC5 PCIe/104 Board is printed on a white label attached to the board (format: "RTC SN...").

The board's ID number and configuration are described in the packaging list (see also "[Accessories](#)", [page 29](#) and "[Type Identification](#)", [page 680](#)).

17.1.7 Type Identification

The ID number of the RTC5 PCIe/104 Board reveals the board's factory-equipped jumper configuration (**JP1-JP8**). The jumper configuration is additionally encoded in a three-digit type code scheme:

Digit 1	=0: Jumper JP1 open (no signal) =1: Jumper JP1 in position 1-2 (5 V) =2: Jumper JP1 in position 2-3 (3.3 V)
Digit 2 (relates to pin B40)	=0: Jumper JP2-JP4 open (no signal) =1: Jumper JP2 closed (+5 V) =2: Jumper JP3 closed (DATA7) =3: Jumper JP4 closed (GND)
Digit 3 (relates to pin B42)	=0: Jumper JP5-JP8 open (no signal) =1: Jumper JP5 closed (+5 V) =2: Jumper JP6 closed (DATA7) =3: Jumper JP7 closed (GND) =4: Jumper JP8 closed (LATCH)

Examples:

- "Type 000": jumpers **JP1-JP8** open
- "Type 124": **JP1** in position 1-2 (5 V signal level), **JP3** closed (DATA7 at pin B40), **JP8** closed (LATCH at pin B42)

17.1.8 Unpacking Instructions and Typical Package Contents

- ▶ Carefully remove the RTC5 PCIe/104 Board from the package.
- ▶ Keep the packaging, including the antistatic bag the RTC5 PCIe/104 Board is delivered in, so that in case of repair the board can be properly repackaged and returned to SCANLAB.
- ▶ Also remove all other articles from the package. Check that all parts have been delivered. Refer to the corresponding packaging list.
The package typically includes an RTC5 PCIe/104 Board and a data CD (containing the corresponding software (see below) and this manual). Some product versions may also supply additional components such as data cables or converters (see "**Optional Accessories**", page 680).

Delivered Software

The delivered software package is identical to the RTC5 software package (see [page 22](#)). It contains drivers for the 32-bit and 64-bit versions of Microsoft Windows 10, 8, 7, Vista, XP SP2, XP SP3, Windows XP Embedded ≥ SP2, as well as correction, DLL, program and utility files for C, C++ and C#.

17.1.9 Optional Accessories

In addition to the RTC5 PCIe/104 Board and its software package, the following accessories – as with the RTC5 PCI Board – can be obtained from SCANLAB (only hardware extensions from SCANLAB should be used in combination with the RTC5 PCIe/104 Board):

- XY2-100 converter (see [page 29](#)).
- Data cables (see [page 29](#)).
- Laser adapter (see [page 29](#)).
- Slot cover with a 15-pin D-SUB connector for using the inputs and signals of the LASER connector (see [page 661](#))

17.1.10 Supplementary Software

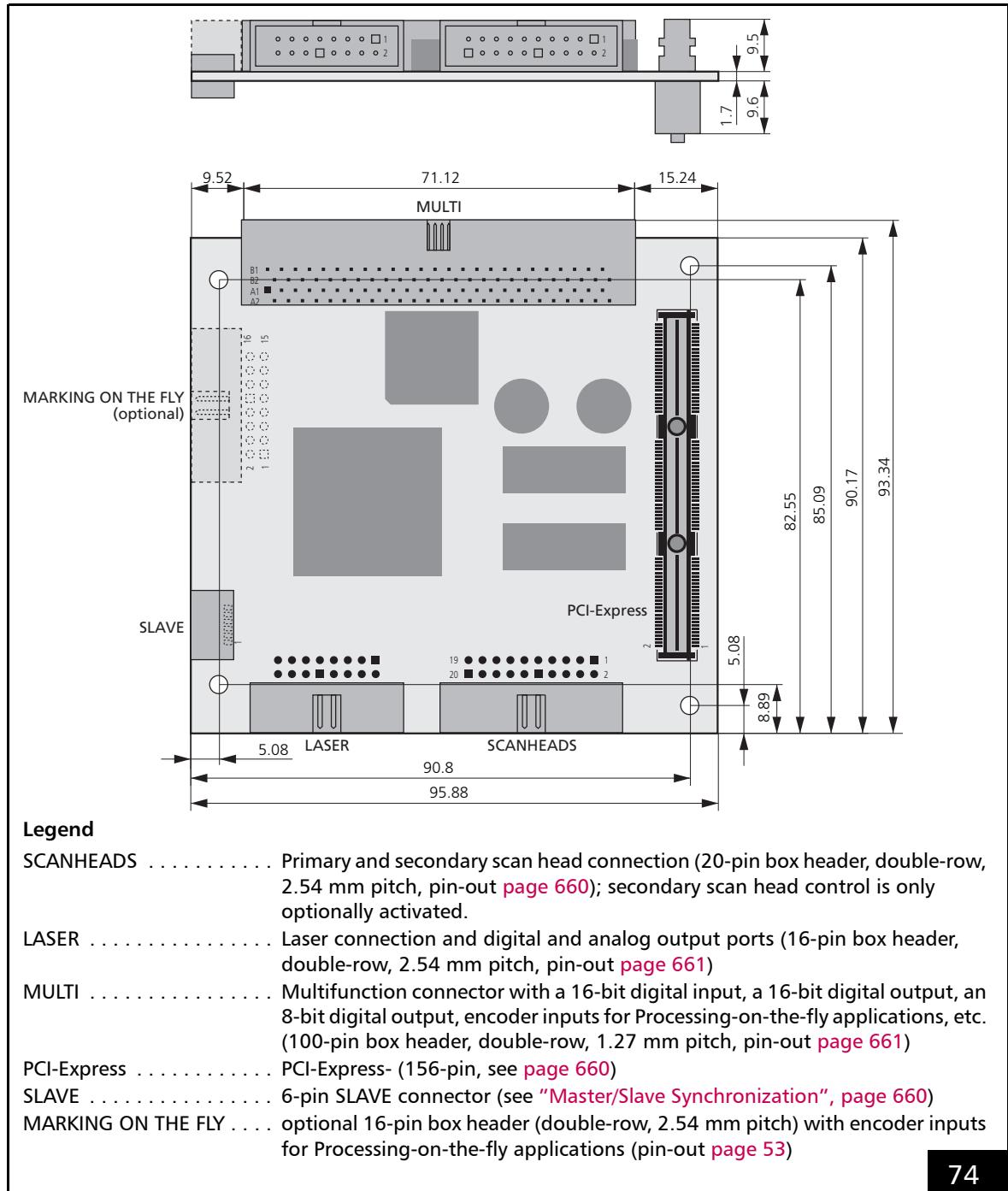
To facilitate customizing RTC correction files basing on your own test measurements, SCANLAB offers its correXion line of software and related documentation (see also [page 137](#)).

17.2 Layout and Interfaces

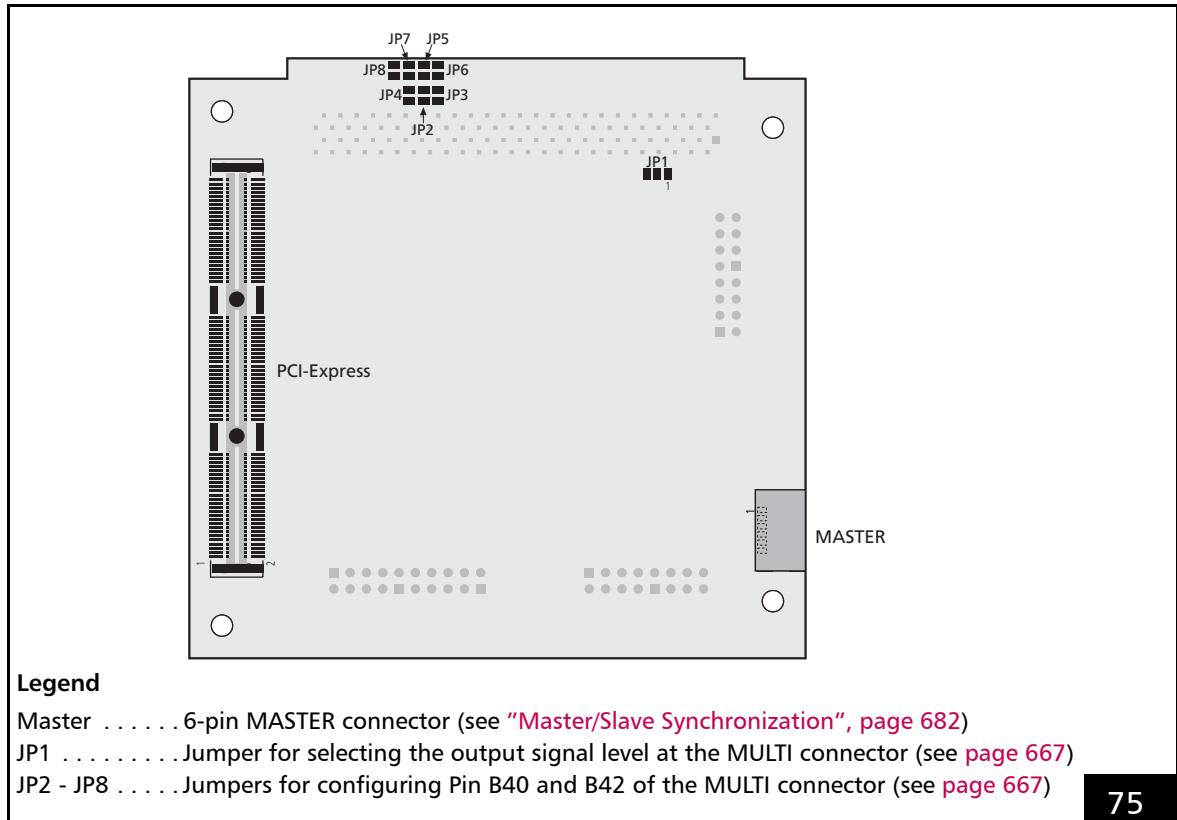
the RTC5 PCIe/104 Board. All connectors and jumper settings are described in detail in the following sections.

17.2.1 Connectors and Jumper Positions

Figure 74 and **figure 75** show the positions of the connectors and jumpers on the front and back side of



Layout and dimensions of the RTC5 PCIe/104 Board (top: side view, bottom: front side)



Layout of the RTC5 PCIe/104 Board (back side)

75

17.2.2 Interface to the CPU Board

Data transfer between the CPU board and the RTC5 PCIe/104 Board is by the PCI-Express bus. The RTC5 PCIe/104 Board provides 156-pin PCIe/104 connectors (see figure 74). The timing and signal levels of the PCI signals are in compliance with PCI-Express Base Specification Revision 2.0 and with PCI Local Bus Specification Revision 2.3.



Caution!

- The RTC5 PCIe/104 Board does not support power-saving modes that switch off power to the PCI bus. Accordingly, you must disable standby or sleep modes of the operating system. See also the note on page 26.

17.2.3 Master/Slave Synchronization

If multiple synchronously-clocked RTC5 PCIe/104 Boards are to be used in a PCIe/104 stack, then the RTC5 PCIe/104 Boards must be connected pairwise with each other by the MASTER and SLAVE connectors and installed in adjacent stack positions. Always connect a board’s MASTER connector to the SLAVE connector of another board. Suitable connection cables are available from SCANLAB.

See also chapter 6.6.3 “Master/Slave Operation”, page 93 and “Starting and Stopping Lists by External Control Signals and Master/Slave Synchronization”, page 239.



17.2.4 Interfaces to Scan System

For digitally controlling scan systems, the 20-pin SCANHEADS connector provides the primary scan head connection and the optionally activated secondary scan head connection. The pin-out and signals are identical to the RTC5 PC/104-Plus Board's SCANHEADS connector (see [page 660](#)).

17.2.5 Interfaces for the Laser and Peripheral Equipment

The RTC5 PCIe/104 Board's 16-pin LASER connector and 100-pin MULTI connector serve as the interface for the laser and peripheral equipment. Their pin-out and signals are identical to the RTC5 PC/104-Plus Board's corresponding connectors (see [page 661](#)).

The pin-out and signals of the optional MARKING ON THE FLY connector are identical to the MARKING ON THE FLY connector of the RTC5 PCI Board (see [page 53](#)).

17.3 Installation and Start-Up

Installation of the RTC5 PCIe/104 Board consists of the following steps:

(1) Configuring the jumpers

Configuration of jumpers JP1 and JP2-JP8 is identical to that of the RTC5 PC/104-Plus Board (see [chapter 15.3.2, page 665](#)).

(2) Installing the RTC5 PCIe/104 Board

The board is installed in the PCIe/104 stack by a PCI Express connector. Additionally, SCANLAB recommends securing the board (by screws and stand-off spacers not included with the product) to the CPU board or adjoining modules in the stack. Electrical configuration of the board's position in the stack is not necessary. In other respects, hardware installation is similar to that of the RTC5 PC/104-Plus Board (see [chapter 15.3.3, page 668](#); you must particularly ensure adherence to all safety notices in this chapter and in [chapter 15.3.1, page 664](#)).

(3) Installing the drivers and the RTC5 software (see [chapter 15.3.4, page 670](#))

Proceed in the order described in the following sections.

The included HPGL converter program can be used for conducting a post-installation functionality test (see [chapter 15.3.5, page 670](#)).

17.4 Technical Specifications of RTC5 PCIe/104 Board

System Requirements

PCIe/104 CPU board
+ free position in the PCIe/104 stack

Operating system Microsoft Windows 10, 8,
7, Vista, XP SP2, XP SP3,
Windows XP Embedded
≥ SP2 as 32 bit or 64 bit
version

Dimensions

Length x Width 96 mm x 93.5 mm
Stack height approx. 16 mm
Max. component height 9.3 mm

Weight

approx. 85 g

Operating and Storage Conditions

Operating temperature +15 °C to +60 °C
Storage temperature –20 °C to +60 °C
Environmental conditions non-condensing
non-corrosive

Connectors to PCIe/104 CPU board (or to PCIe/104-Stack)

PCI-Express bus 156-pin PCIe/104
standard top and bottom
connectors (PCIe x 1, only
one lane used),
operating voltage: +5 V

Other connectors (signal specifications identical to RTC5 PCI Board)

Interface to scan system SCANHEADS connector
(20-pin box header) with primary and secondary
scan head connection (secondary scan head
control is only optionally activated)

Interfaces for laser and peripheral equipment LASER connector (16-pin
box header)
MULTI multifunction connector (100-pin box
header)
optional:
MARKING ON THE FLY connector (16-pin box
header)

Note on ANALOG OUT1 and ANALOG OUT2
(applies only to the RTC5 PCIe/104 Board):
Depending on the applied load, the two analog
outputs put out a minimum voltage, even when
commands like `write_da_x` define output voltages
below this minimum voltage. Provided that the
applied load does not exceed the specified
maximum load, the minimum voltage is in the
range 0 V...0.1 V.

18 Revision History

This list ignores spelling/grammar corrections, syntax reformulations or additions (such as new cross-references) that do not affect the technical meaning. The supplied page numbers refer to this document.

Changes from revision 0.9 to revision 0.11

Page	Name of chapter / command	Change
21	chapter 1.2 "Labeling"	ID number is not written on the board.
45	chapter 4.3.3 "Data Cables (Accessories)"	Recommend cable impedance added.
135	chapter 7.3.3 "Virtual Image Field"	Chapter added (also usage for coordinate transformations).
226	chapter 8.11.3 "Determining Reference Values"	Reference values for automatic self calibration are stored in DLL or EEPROM.
240	External List Start	Information on master/slave synchronization with external start and activated home jump.
250	Normal, Short and Variable List Commands and List Multi-Commands	Expanded information on short list commands.
270	auto_cal command	Reference values for automatic self calibration are stored in DLL or EEPROM.
369	list_nop command	New comment.
464	set_auto_laser_control command	Corrected: return value (error code) 5.
475	set_delay_mode command	Corrected: RTC4 compatibility mode.
480	set_end_of_list command	Corrected: comments.
537	set_port_default command	Corrected: comments.
655	chapter 15 "Appendix A: The RTC5 PC/104-Plus Board"	Chapter added.



Changes from revision 0.11 to revision 0.12

Page	Name of chapter / command	Change
44	chapter 4.3.2 "XY2-100 Converter (Optional)"	Corrected: recommended jumper settings and cable lengths.
54	chapter 4.4.6 "SPI / I2C Socket Connector"	Section about McBSP interface added.
96	chapter 6.7 "Usage by Multiple Applications"	Corrected: RTC5_TIMEOUT error during version check eliminated.
168	Encoder-Speed-Dependent Laser Control	Section added.
199	chapter 8.7 "Processing-on-the-fly (Optional)"	Chapter revised and expanded (now Processing-on-the-fly correction also with direct position signals by the McBSP interface).
217	chapter 8.8 "Pixel Output Mode – Scanning Raster Images (Bitmaps)"	Chapter expanded – new command: <code>set_n_pixel</code> .
224	chapter 8.11 "Automatic Self-Calibration"	Corrections corresponding to the changes of the commands <code>auto_cal</code> and <code>get_hi_pos</code> (see below).
237	chapter 9.1.7 "McBSP/SPI Interface"	Section added.
238	chapter 9.2.4 "McBSP/SPI Interface"	Section added.
239	External List Stop	Corrected: the inputs for external stop signals can not be locked.
240	External List Start	Corrected: figure 61.
247	Inputs for External Encoder Signals	Corrected: maximum allowed frequency for encoder signals.
248	chapter 9.3.4 "Synchronization and Online Positioning by McBSP/SPI Signals"	Section added.
270	<code>auto_cal</code> command	Added: prerequisite for iDRIVE scan systems. Corrected: for Command = 3, the current Home-In positions are detected (no reference values are stored).
309	<code>get_hi_pos</code> command	Corrected: the command returns the current Home-In positions (normally not the reference values).
317	<code>get_mcbsp</code> command	Added.
385	<code>load_list</code> command	Corrected: change for ListNo = 3.
413	<code>mark_time</code> command	Corrected: parameter Mode (bit#0 = 1).
415	<code>mcbsp_init</code> command	Added.
464	<code>set_auto_laser_control</code> command	Added: encoder-speed-dependent laser control by Mode = 5.



Page	Name of chapter / command	Change
470	<code>set_control_mode</code> command	Corrected: the inputs for external stop signals can not be locked.
478	<code>set_encoder_speed</code> command	Added.
479	<code>set_encoder_speed_ctrl</code> command	Added.
488	<code>set_fly_rot_pos</code> command	Added.
491	<code>set_fly_x_pos</code> command	Added.
493	<code>set_fly_y_pos</code> command	Added.
510 511 512 557 558	<code>set_laser_pulses</code> command <code>set_laser_pulses_ctrl</code> command <code>set_laser_timing</code> command <code>set_standby</code> command <code>set_standby_list</code> command	Corrected: enhanced allowed range for parameter HalfPeriod.
521	<code>set_mcbsp_out</code> command	Added.
529	<code>set_n_pixel</code> command	Added.
633	<code>wait_for_encoder_mode</code> command	Added.
634	<code>wait_for_mcbsp</code> command	Added.
665	Jumper 17+18: Setting the Binary Stack Address	Table corrected: inverted jumper configurations.
666	JP10+12: 10 V Output Voltage for ANALOG OUT1 and ANALOG OUT2	Max. output current specifications added.
685	chapter 18 "Revision History"	Added.

Changes from revision 0.12 to revision 1.0

Page	Name of chapter / command	Change
	In general	The manual is no longer called "preliminary".
23	Revision History	Added.
45	chapter 4.3.3 "Data Cables (Accessories)"	Corrected: SL2-100 data cable does not need a ferrite ring.
63	chapter 5.5 "Safe Start-up and Shutdown Sequences"	Added.
71	chapter 6.2.5 "Example Code"	Added.
93	chapter 6.6.3 "Master/Slave Operation"	Added.
100	chapter 6.8.3 "Example Code"	Added.
99	chapter 6.8.1 "Download Verification"	Added: extended download verification.
103	chapter 7.1 "Marking Points, Lines and Arcs"	Added.
109	chapter 7.1.4 "Example Code"	Added.
122	Adjustment of the Scanner Delays	Added: extended automatic adjustment.
142	Clock Overruns	Added.
144	chapter 7.4 "Laser Control"	Revised: The terms "laser active" and "laser standby" were introduced in this chapter for clarification (these are <i>not</i> new functions). The terms are now also used elsewhere in this manual. Corrected: standby signals in YAG modes. Added: chapter 7.4.7 "Laser Mode 6" , page 155.
242	Regular (Periodic) External List Starts	Added.
253	chapter 10.1.3 "Version Information"	Added.
253	chapter 10.1.4 "Optional Functions"	Added.
270	auto_cal command	Corrected (return values): axis 1/2 -> X/Y axis.
283	control_command command	Description revised for clarification. Corrected: The calibration angle (<code>Data = 0523H</code>) is returned in millidegrees (not in mrad). Corrected: bit assignment for <code>Data = 0526H, 0527H, 0529H</code> and <code>053FH</code> . Corrected: The scan system's status return behavior can only be temporarily changed (<code>Data = 0A00H</code> , see also page 182). New: SetControlDefinitionMode command (<code>CodeH = 0EH</code>).
302	get_error command	New: Bit #9 (Verify error).
307	get_head_status command	Changed: now this command (as get_value or set_trigger already did before) returns reserved bits as 1.
309	get_hi_data command	Changed: Prerequisite canceled.
309	get_hi_pos command	Changed: Prerequisite canceled.

Page	Name of chapter / command	Change
314	<code>get_marking_info</code> command	Added: command also returns information about improper encoder signals.
316	<code>get_master_slave</code> command	Added.
317	<code>get_mcbsp</code> command	Corrected: Result value is a <i>signed</i> 32-bit value.
317	<code>get_mcbsp_list</code> command	Added.
318	<code>get_overrun</code> command	Added.
320	<code>get_standby</code> command	Added.
321	<code>get_startstop_info</code> command	Added: beginning with firmware version 508, the firmware returns the pulse length error bit (bit #17 or 25), when no scan system is currently connected.
328	<code>get_text_table_pointer</code> command	Corrected: allowed index range.
332	<code>get_value</code> command	Added: current laser status returns even outside list execution.
338	<code>goto_xy</code> command	Added: new comment (external stop signal).
340	<code>home_position</code> command	Corrected: A home jump is also executed, if list execution is stopped by <code>stop_execution</code> or by an external stop signal.
340	<code>home_position_xyz</code> command	Added.
373	<code>load_auto_laser_control</code> command	Changed: return code 11.
375	<code>load_correction_file</code> command	Added: new error codes.
379	<code>load_disk</code> command	Added: initialization by Name = 0.
387	<code>load_position_control</code> command	Changed: return code 11.
389	<code>load_program_file</code> command	Added: new error code.
395	<code>load_varpolydelay</code> command	Changed: return code 11.
396	<code>load_z_table</code> command	New: Error code 64 as return Value.
402	<code>mark_date</code> command (<code>mark_date_abs</code> command, too)	Added: month and day-of-the-week can also be marked as normal numerals (parameter Part = 6, 7).
470	<code>set_control_mode</code> command	Added: Bit#10 for regular (periodic) external list starts.
482	<code>set_ext_start_delay</code> command	Added: RTC4 compatibility mode for parameter Delay.
483	<code>set_ext_start_delay_list</code> command	Changed: Now a normal list command Added: RTC4 compatibility mode for parameter Delay.
487	<code>set_fly_rot</code> command (analogously for <code>set_fly_rot_pos</code> , <code>set_fly_x</code> , <code>set_fly_y</code> , <code>set_fly_x_pos</code> and <code>set_fly_y_pos</code>)	Changed: If unallowed parameter values are supplied, only deactivates a Processing-on-the-fly correction previously activated by <code>set_fly_rot</code> but not any other Processing-on-the-fly correction.
508	<code>set_laser_mode</code> command	Added: Laser mode 6.
531	<code>set_offset_xyz</code> command	Added.
532	<code>set_offset_xyz_list</code> command	
562	<code>set_trigger</code> command	Added: effective output value by parameter Signal1/Signal2 = 20-23.
571	<code>set_verify</code> command	Added: extended download verification.
583	<code>simulate_ext_start</code> command	Changed: Now a normal list command Added: RTC4 compatibility mode for parameter Delay.



Page	Name of chapter / command	Change
584	simulate_ext_start_ctrl command	Added.
596	stop_execution command	Corrected: A home jump is executed.
602	switch_iport command	Added.
603	sync_slaves command	Added.
631	verify_checksum command	Added.
647	chapter 11 "Demo Programs"	Added.
648	chapter 12 "Troubleshooting"	Added.



Changes from revision 1.0 to revision 1.1

Page	Name of chapter / command	Change
31	chapter 2.7 "Notes for RTC4 Users"	Extended.
64	chapter 5.8 "Exchanging RTC5 Boards"	Added.
105	"Ellipse Commands"	Added.
139	"Inverse Tables"	Added.
174	chapter 8.1.3 "Position Monitoring"	Added.
196	"Checking the Z-Axis Calibration"	Revised.
209	chapter 8.7.5 "Deactivating Processing-on-the-fly Corrections"	Added.
251	"Undelayed and Delayed Short List Commands (as of Version OUT 515)"	Added.
267	<code>arc_abs_3d</code> command	Added.
269	<code>arc_rel_3d</code> command	Added.
302	<code>get_error</code> command	Added: Bit #10 (DSP version error).
319	<code>get_RTC_version</code> command	Added: Bits #16...#23 (DSP version number).
329	<code>get_transform</code> command	Added.
332	<code>get_value</code> command	Added: new comments, code example.
334	<code>get_values</code> command	Added.
355	<code>laser_on_pulses_list</code> command	Added.
405	<code>mark_ellipse_abs</code> command	Added.
406	<code>mark_ellipse_rel</code> command	Added.
467	<code>set_auto_laser_params</code> command	Added.
467	<code>set_auto_laser_params_list</code> command	Added.
472	<code>set_default_pixel</code> command	Added.
472	<code>set_default_pixel_list</code> command	Added.
477	<code>set_ellipse</code> command	Added.
504	<code>set_laser_control</code> command	Added: Bit#5 for <code>laser_on_pulses_list</code> .
550	<code>set_sky_writing_para</code> command	Added.
552	<code>set_sky_writing_para_list</code> command	Added.
597	<code>stop_trigger</code> command	Added.



Page	Name of chapter / command	Change
603	sync_slaves command	Changed: Beginning with DLL 518, OUT 517, RBF 514, the following applies: <ul style="list-style-type: none">• Master/Slave synchronization is only executable by the sync_slaves command. Synchronization cannot be achieved by an appropriately sequenced initialization of the boards.• Synchronization is only applied to downstream slave boards allocated to the user program.• sync_slaves now automatically executes a simulate_ext_stop command.
617 - 626	commands timed_para_jump_abs ... timed_para_mark_rel_3d	Eight timed_para* commands added.
627	transform command	Added.
630	upload_transform command	Added.
641	write_io_port_mask command	Added.
641	write_io_port_mask_list command	Added.

Changes from revision 1.1 to revision 1.2

Page	Name of chapter / command	Change
183	chapter 8.2 "Coordinate Transformations"	Added: behavior for <code>at_once > 1</code> (see below).
224	chapter 8.11 "Automatic Self-Calibration"	Extended and revised: including information on <code>auto_cal</code> (<code>Command = 4</code>) and on iDRIVE systems.
250	Normal, Short and Variable List Commands and List Multi-Commands	Changed: Beginning with OUT 518, the following applies: <ul style="list-style-type: none"> • Now up to 12 short list commands per 10 µs clock period are possible. • For exceedances of the maximum number, a 10 µs clock period is now automatically inserted instead of a <code>list_nop</code> (for polylines, the laser remains on).
251	Undelayed and Delayed Short List Commands (as of Version OUT 515)	Changed: Beginning with OUT 518, the following applies: For several short list commands in a row, a "delayed short list command" only executes undelayed if a further "delayed short list command" follows, but no longer if an "undelayed short list command" follows.
266 ...	arc_abs command and the corresponding _rel, _3d, para_ and timed_commands	Marking of zero-length arcs no longer changes the laser control signals (see page 115).
270	auto_cal command	Changed: as of DLL 520, OUT 519, the following applies: <ul style="list-style-type: none"> • The ASC hardware can be checked by <code>Command = 4</code> (new). The hardware check is likewise performed with <code>Command = 0</code> and in some circumstances also with <code>Command = 1</code> and <code>3</code> (see command description). • The command now also supports Type 2 sensor systems. • For <code>Command = 0</code> and <code>Command = 2</code>, too, the RTC5 returns the galvanometer scanners to the original position that preceded the command call (i.e. no longer to their mid-positions).
283	control_command command	Changed: After switching to a different data source (by $Code_H = 05_H$, $0E_H$ or 21_H or $Data = 1700_H$), a waiting time of 60 µs is automatically inserted.
299	get_auto_cal command	Added.
338 ... 352	goto_xy , goto_xyz , jump_abs , jump_rel commands	Changed: behavior for coordinate transformations with <code>at_once = 2</code> (see page 185).
398 ...	mark_abs command and the corresponding _rel, _3d, para_ and timed_commands	Zero-length marking no longer changes the laser control signals (see page 115).
405, 406	mark_ellipse_abs command, mark_ellipse_rel command	Marking of zero-length arcs no longer changes the laser control signals (see page 115).
440	read_encoder command	Added.



Page	Name of chapter / command	Change
462 ... 543	Commands for coordinate transformations: <code>set_angle</code> , <code>set_angle_list</code> , <code>set_offset</code> , <code>set_offset_list</code> , <code>set_offset_xyz</code> , <code>set_offset_xyz_list</code> , <code>set_matrix</code> , <code>set_matrix_list</code> , <code>set_scale</code> , <code>set_scale_list</code>	Changed: behavior for <code>at_once > 1</code> (collection as with <code>at_once = 0</code> , and execution of a consolidated jump in conjunction with <code>jump_abs</code> , <code>jump_rel</code> , <code>goto_xy</code> or <code>goto_xyz</code> – see also page 185).
597	<code>store_encoder</code> command	Added.

Changes from revision 1.2 to revision 1.3

Page	Name of chapter / command	Change
54	"Use as McBSP Interface"	Changes and additions (as of version OUT 526): <ul style="list-style-type: none"> • Stagnation of data transmission no longer occurs. • Data transmission for online positioning.
94	"Matching of Short-Command Counts (as of Version DLL 523, OUT 524)"	Added.
108	chapter 7.1.3 "Marking Points"	Added.
116	Notes in section "Mark Delay"	As of RTC5OUT.out version 526: changed behavior of timed mark or arc commands with zero spacial length (see also notes in chapter 8.10 "Timed Vector and Arc Commands", page 223).
121	Automatic Delay Adjustments	Change (as of version DLL 522, OUT 522, RBF 517): <ul style="list-style-type: none"> • in section (1): If LaserOff overlaps LaserOn, then the LaserOff delay still gets automatically extended, but only so that the laser control signals remain on for 0.5 µs (instead of 10 µs). • in section (2): If a LaserOff delay expires and simultaneously a LaserOn delay should again be set, then an additional scanner delay (of 10 µs) is no longer inserted.
138	Notes in section Assigning Loaded Correction Tables	Changed (as of version DLL 521, OUT 521): load_correction_file automatically calls select_cor_table after loading a correction table.
140	Correction File Header	Changed: Parameter 15 (get_head_para , get_table_para) is 2.0 (instead of 1.0) for 'field size clipped'.
187	chapter 8.3 "Online Positioning"	Added.
248	chapter 9.3.4	Added (as of version OUT 526): online positioning.
264, 265	apply_mcbsp command and apply_mcbsp_list command	Added.
317, 317	get_mcbsp command and get_mcbsp_list command	Change (as of version OUT 526): see "Version info" in the command description.
375	load_correction_file command	Changed (as of version DLL 521, OUT 521): load_correction_file automatically calls select_cor_table after loading a correction table.
422	move_to command	Added.
426	para_laser_on_pulses_list command	Added.
443	read_mcbsp command	Added.
476	set_dsp_mode command	Added.
488... 493	set_fly_rot_pos , set_fly_x_pos and set_fly_y_pos	Added (as of version OUT 526): cannot be used simultaneously with online positioning.



Page	Name of chapter / command	Change
523... 525	<code>set_mcbsp_rot</code> , <code>set_mcbsp_rot_list</code> , <code>set_mcbsp_x</code> , <code>set_mcbsp_x_list</code> , <code>set_mcbsp_y</code> , <code>set_mcbsp_y_list</code>	Added.
603	<code>sync_slaves</code> command	Updated with version DLL 523, OUT 524: also triggers matching of short command counts when necessary (see page 94).
605	<code>time_fix_f_off</code> command	Added.



Changes from revision 1.3 to revision 1.4

Page	Name of chapter / command	Change
108	chapter 7.1.3 "Marking Points"	Added (as of version OUT 527): Marking points by mark or arc commands of zero length.
176	chapter 8.1.5 "Jump Mode"	Added.
213	chapter 8.7.9 "Monitoring Processing-on-the-fly Corrections"	Revisions and additions to the section " Customer-Defined Monitoring Area and Conditional Command Execution (as of Version DLL 525, OUT 527) ".
221	chapter 8.9 "Microvector Commands"	Added.
250	chapter "Normal, Short and Variable List Commands and List Multi-Commands"	Added (as of version DLL 525, OUT 527): Postponing short list commands by list_continue (alternative to list_nop).
278	clear_fly_overflow command	Added.
311	get_jump_table command	Added.
314	get_marking_info command	Change (as of version DLL 525, OUT 527): bits#4...7.
342 ...	if_fly_x_overflow command, if_fly_y_overflow command, if_not_fly_x_overflow command, if_not_fly_y_overflow command	Added.
346 ...	if_not_pin_cond command, if_pin_cond command	New note about reliable usage page 244 .
363	list_continue command	Added.
383	load_jump_table_offset command	Added.
398 ...	mark_abs command and the corresponding _rel, _3d and para_commands (but not the corresponding timed commands)	Change (as of version OUT 527): A mark of zero length only does not change the laser control signals if it occurs within a polyline (see page 115).
417	measurement_status command	Added: Pos = $2^{32}-1$.
418	micro_vector_abs command, micro_vector_rel command	Added.
462 ... 543	Commands for coordinate transformations: set_angle , set_angle_list , set_offset , set_offset_list , set_offset_xyz , set_offset_xyz_list , set_matrix , set_matrix_list , set_scale , set_scale_list	Changed: behavior for at_once = 3 (here the laser control signals remain unaffected).
485	set_fly_limits command	Added.
498	set_jump_mode command	Added.
501	set_jump_mode_list command	Added.
503	set_jump_table command	Added.
569	set_vector_control command	Change (as of version DLL 525, OUT 527): Variation of the focus shift with Ctrl = 7.



Changes from revision 1.4 to revision 1.5

Page	Name of chapter / command	Change
26 and 61	chapter 2.2.2 "Software" and chapter 5.3 "Installing the Drivers"	Warning added: Standby or sleep modes of the operating system must be deactivated.
58	chapter 4.4.7 "STEPPER MOTOR Socket Connector (Stepper Motor Control)"	Revised.
93	chapter 6.6.3 "Master/Slave Operation"	Corrected: An RTC5 Board is the master board if no further RTC5 Board is connected to its SLAVE (!) connector.
127	chapter 7.2.4 "Sky Writing"	Revised and extended (Sky Writing mode 2).
146	chapter "Laser-Signal Auto-Suppression Triggered by Scan-System-Errors"	Added.
234	chapter 9.1.5 "Stepper Motor Control"	Added.
314	<code>get_marking_info</code> command	Added: now also returns the error bits of laser-signal auto-suppression (bits#10...15 and 26...31).
316	<code>get_master_slave</code> command	Corrected: master/slave recognition by bit#0 and 1.
324	<code>get stepper_status</code> command	Added.
338, 339	<code>goto_xy</code> command, <code>goto_xyz</code> command	Changed (as of version DLL 527, OUT 529): These commands return only after motion has completed.
354	<code>laser_on_list</code> command	Corrected: The laser control signals only turn off with a subsequent list command.
375	<code>load_correction_file</code> command	Changed (as of version DLL 527, OUT 529): The command only returns after the jump to the corrected galvanometer position has completed.
382	<code>load_jump_table</code> command	Corrected: has no parameter <code>Offset</code> .
383	<code>load_jump_table_offset</code> command	Added: Function as described in version 1.4 for the <code>load_jump_table</code> command (with parameter <code>Offset</code>).
548, 549	<code>set_sky_writing_mode</code> command, <code>set_sky_writing_mode_list</code> command	Added.
550, 552	<code>set_sky_writing_para</code> command, <code>set_sky_writing_para_list</code> command	Upgraded: Sky Writing mode 2.



Page	Name of chapter / command	Change
586 ... 595	Stepper motor control commands: <code>stepper_abs, stepper_abs_list,</code> <code>stepper_abs_no,</code> <code>stepper_abs_no_list,</code> <code>stepper_control,</code> <code>stepper_control_list,</code> <code>stepper_enable,</code> <code>stepper_enable_list, stepper_init,</code> <code>stepper_rel, stepper_rel_list,</code> <code>stepper_rel_no,</code> <code>stepper_rel_no_list, stepper_wait</code>	Added.
603	<code>sync_slaves</code> command	Corrected: An RTC5 Board is the master board if no further RTC5 Board is connected to its SLAVE (!) connector.
651	chapter 14 "Technical Specifications of the RTC5 PCI Board"	Added: Specifications for "SPI / I2C" connector and "STEPPER MOTOR" connector.
654	chapter 14.1 "Compliance with EC Guidelines for Electromagnetic Compatibility (EMC)"	Added.
654	chapter 14.2 "Compliance with FCC Rules"	Added.
656 and 670	"Software" and chapter 15.3.4 "Installing the Drivers and the RTC5 Software"	Warning added: Standby or sleep modes of the operating system must be deactivated.
672	chapter 16 "Appendix B: The RTC5 PCI Express Board"	Added.
439	<code>read_analog_in</code> command	Added.



Changes from revision 1.5 to revision 1.6

Page	Name of chapter / command	Change
22	chapter 1.3.1 "Delivered Software" and further chapters	Added: The RTC5 software package now includes drivers for 64-bit Windows operating systems as well as Win32- and Win64-based DLLs, but no longer the MS Visual Studio runtime libraries. Note the corresponding comments in "Installing the RTC5 Software", page 62, "User Programs and Demo Software", page 64, "Initialization and Program Start-Up", page 66 and "Data Types", page 260.
26	chapter 2.2.2 "Software"	Added: Requirements for new and older RTC5 software packages (new RTC5 DLLs need at least Windows XP SP2).
48	"I/O Circuits" for LASER connector	Added.
102	chapter 6.9.1 "Free Variables"	Added.
128	"Mode 3" for Sky Writing	Added (as of version DLL 531, OUT 532).
157	chapter 7.4.8 "Pulse Picking Laser Mode"	Added.
187	chapter 8.3 "Online Positioning"	Change (with version DLL 531, OUT 532): If both memory locations contain identical data types (with identical bit#31 codings) at the time of the request by apply_mcbsp (or apply_mcbsp_list), then the most recently copied value is always used (older DLL/OUT versions always use the value in memory location 2).
199	chapter 8.7 "Processing-on-the-fly (Optional)"	Correction by McBSP interface with additional McBSP input (see new sections page 203 and page 205 and chapter 8.7.5 "Deactivating Processing-on-the-fly Corrections", page 209).
237	chapter 9.1.7 "McBSP/SPI Interface"	Added: set_mcbsp_out_ptr .
241	New note about "External List Start"	External starts are suppressed after pause_list , stop_list or set_wait .
304	get_free_variable command	Added.
326	get_sync_status command	Added.
332	get_value command	Added (with version DLL 531, OUT 532): Signal = 25...42.
355	laser_on_pulses_list command	Changed (as of version DLL 530, OUT 531, RBF 522): changed functionality for $2^{31} \leq \text{Period} \leq (2^{32}-1)$.
443	read_mcbsp command	Added (with version DLL 531, OUT 532): No = 3 and set_mcbsp_in input.



470	set_control_mode command, set_control_mode_list command	Added: bit#1 (as of version DLL 528, OUT 530).
495	set_free_variable command, set_free_variable_list command	Added.
498	set_jump_mode command	Changed (as of version DLL 530, OUT 531, RBF 522): changed error code for return values >1 (byte#1 and byte#2).
517	set_mcbsp_freq command	Added.
518	set_mcbsp_in command, set_mcbsp_in_list command	Added.
521	set_mcbsp_out command	Added (with version DLL 531, OUT 532): Signal1, Signal2 = 25...42.
522	set_mcbsp_out_ptr command	Added.
538	set_pulse_picking command, set_pulse_picking_list command	Added.
547	set_sky_writing_limit command, set_sky_writing_limit_list command	Added (as of version DLL 531, OUT 532).
548	set_sky_writing_mode command, set_sky_writing_mode_list command	Added (as of version DLL 531, OUT 532): mode 3.
562	set_trigger command	Added (with version DLL 531, OUT 532): Signal1, Signal2 = 25...42.
678	chapter 17 "Appendix C: The RTC5 PCIe/104 Board"	Added.



Changes from revision 1.6 to revision 1.7

Page	Name of chapter / command	Change
26	chapter 2.2.2 "Software"	Added: Notes on new drivers.
59	chapter 4.4.8 "Analog Inputs (Optional Accessory)"	Added.
61	chapter 5.3 "Installing the Drivers"	Added: Installation under Windows 8 and new chapter 5.3.1 "Software Package Upgrades", page 62.
146	chapter 7.4.2 "Configuring the Laser Connector (LASER)"	Added.
157	chapter 7.4.8 "Pulse Picking Laser Mode"	Added: constant pulse length mode.
169	chapter 7.4.10 "Output Synchronization"	Added.
187	chapter 8.3 "Online Positioning"	Added: matrix operation (<code>set_mcbsp_matrix</code> , <code>set_mcbsp_matrix_list</code>).
210	chapter 8.7.6 "Virtual Image Field"	Added: Examples of using the virtual image field with Processing-on-the-fly applications.
215	chapter 8.7.10 "Tracking Error Compensation of Encoder Values for Processing-on-the-fly Applications"	Added.
270	<code>auto_cal</code> command	Change with version DLL 535: return value 8.
280	<code>config_laser_signals</code> command	Added (with version DLL 533, OUT 534, RBF 524).
302	<code>get_error</code> command	Added with version DLL 535: bits #11, 12, 16.
308	<code>get_hex_version</code> command	Change with version DLL 535.
319	<code>get_RTC_version</code> command	Change with version DLL 535.
354	<code>laser_on_list</code> command	Corrected: changed behavior as of DLL 530, OUT 531, RBF 522 (as with <code>laser_on_pulses_list</code>).
489	<code>set_fly_tracking_error</code> command	Added with version DLL 533, OUT 534, RBF 524.
504	<code>set_laser_control</code> command	Added: bit#6 for output synchronization and bit#7 for constant pulse length mode (with version DLL 530, OUT 531, RBF 522 and DLL 533, OUT 534, RBF 524).
517	<code>set_mcbsp_freq</code> command	Corrected: behavior for out-of-range parameter values.
520	<code>set_mcbsp_matrix</code> command, <code>set_mcbsp_matrix_list</code> command	Added with version DLL 533, OUT 534, RBF 524.
538	<code>set_pulse_picking_length</code> command	Added (with version DLL 533, OUT 534, RBF 524).
571	<code>set_verify</code> command	Added with version DLL 535: automatic error recognition of a failed memory request.
596	<code>stop_execution</code> command	Change with version OUT 535.

Changes from revision 1.7 to revision 1.8

Page	Name of chapter / command	Change
91	chapter 6.5.5 "Loops"	Chapter added (version DLL 536, OUT 536).
144	chapter 7.4.1 "Enabling, Activating and Switching Laser Control Signals"	Corrected: The laser output ports (LASERON, LASER1 and LASER2) also enter the (high impedance) tristate mode if the laser control signals are subsequently disabled by a command (see also <code>disable_laser</code> , <code>pause_list</code> and <code>set_laser_control</code> (bit#2 = 1)).
169	chapter 7.4.10 "Output Synchronization"	Notes changed (version RBF 527).
179	"Notes on Loading Determined Jump Delay Values" (chapter 8.1.5 "Jump Mode")	Corrected: the maximum value of the <i>Length</i> parameter for jump delay tables is 1048576.0 (instead of 524288.0).
192	chapter 8.5 "Using Several Different Correction Tables"	Changed (version DLL 533, OUT 534): up to 4 correction files can be loaded.
198	chapter 8.6.5 "Enhanced 3D Correction"	Chapter added (version DLL 536, OUT 536).
199	chapter 8.7 "Processing-on-the-fly (Optional)"	Changed (version DLL 536, OUT 536): <ul style="list-style-type: none"> New: section <code>Overview</code>. New: chapter 8.7.4 "Compensation of 2D Motions" (incl. the sections 2D Encoder Compensation for XY Stages and Coordinate Transformations in the Virtual Image Field). Revised: chapter 8.7.6 "Virtual Image Field". Extended: chapter 8.7.7 "Synchronization of Processing-on-the-fly Applications". New: chapter 8.7.8 "Encoder Resets".
252	"List Multi-Commands"	Added.
262	<code>activate_fly_2d</code> command, <code>activate_fly_xy</code> command	Added (version DLL 536, OUT 536).
304	<code>get_fly_2d_offset</code> command	Added (version DLL 536, OUT 536).
314	<code>get_marking_info</code> command	Changed (version DLL 536, OUT 536): bit #9.
327	<code>get_table_para</code> command	Changed (with version DLL 533, OUT 534): value range of TableNo.
332	<code>get_value</code> command, <code>get_values</code> command	Changed (with version DLL 536, OUT 536): additional selectable data signals (Signal = 43 and 44).
336	<code>get_waveform</code> command	Changed (with version DLL 536, OUT 536): Channel = 3 and 4.
344	<code>if_not_activated</code> command	Added (version DLL 536, OUT 536).
347	<code>init_fly_2d</code> command	Added (version DLL 536, OUT 536).
370	<code>list_repeat</code> command, <code>list_until</code> command	Added (with version DLL 536, OUT 536).
375	<code>load_correction_file</code> command	Changed (with version DLL 533, OUT 534): value range of No (up to 4 correction files can be loaded).
381	<code>load_fly_2d_table</code> command	Added (version DLL 536, OUT 536).



392	load_stretch_table command	Added (with version DLL 536, OUT 536).
418	micro_vector_abs command, micro_vector_rel command	Changed (version DLL 536, OUT 536): LasOn, LasOff < 2^{15} .
419	micro_vector_abs_3d command, micro_vector_rel_3d command	Added (version DLL 536, OUT 536).
431	park_position command, park_return command	Added (version DLL 536, OUT 536).
457	select_cor_table command, select_cor_table_list command	Changed (with version DLL 533, OUT 534): value range of parameters.
476	set_delay_mode_list command	Added (with version DLL 536, OUT 536).
478	set_encoder_speed command, set_encoder_speed_ctrl command	Changed (with version DLL 536, OUT 536): EncoderNo = 2 and 3.
484	set_fly_2d command	Added (version DLL 536, OUT 536).
504	set_laser_control command	Changed (with version DLL 536, OUT 536): bit#28.
515	set_matrix command	Changed (version DLL 536, OUT 536): HeadNo = 4.
530	set_offset command, set_offset_xyz command	Changed (version DLL 536, OUT 536): HeadNo = 4.
536	set_pixel_line_3d command	Added (with version DLL 536, OUT 536).
554	set_softstart_level_list command	Added (with version DLL 536, OUT 536).
556	set_softstart_mode_list command	Added (with version DLL 536, OUT 536).
562	set_trigger command	Changed (with version DLL 536, OUT 536): Signal1, Signal2 = 43 and 44.
568	set_trigger4 command	Added (with version DLL 536, OUT 536).
619	timed_para_jump_abs_3d , timed_para_jump_rel_3d , timed_para_mark_abs_3d , timed_para_mark_rel_3d commands	Corrected: These commands are list <i>multi</i> -commands (instead of normal list commands).
632	wait_for_encoder_in_range command	Added (version DLL 536, OUT 536).



Changes from revision 1.8 to revision 1.9

Page	Name of chapter / command	Change
170	chapter 7.5.2 "Marking Serial Numbers"	Expanded (with version DLL 537, OUT 537): Serial number marking now with selectable serial-number-sets.
189	chapter 8.4 "Wobbel Mode"	Expanded: specify direction of motion and "freely definable wobbel shapes".
190	chapter 8.4.1 "Wobbel Shapes – Important Notes on Choosing Appropriate Parameter Values"	Added.
203	chapter "Correction by McBSP/SPI Interface with Enhanced McBSP/SPI Input"	Processing-on-the-fly correction in Z direction and laser power variation is now possible.
278	clear_fly_overflow command	Changed with version DLL 531, OUT 532: bits #4, 5.
280	config_laser_signals_list command	Added (with version DLL 537, OUT 537).
298	fly_return_z command	Added (with version DLL 531, OUT 532).
313	get_list_serial command	Added (with version DLL 537, OUT 537).
343	if_fly_z_overflow command	Added (with version DLL 531, OUT 532).
346	if_not_fly_z_overflow command	Added (with version DLL 531, OUT 532).
389	load_program_file command	Expanded (with version DLL 537, OUT 537): PCI error now has its own error code 16.
436	range_checking command	Added (with version DLL 537, OUT 537).
444	read_multi_mcbsp command	Added (with version DLL 537, OUT 537).
461, 461	select_serial_set command, select_serial_set_list command	Added (with version DLL 537, OUT 537).
546	set_serial_step_list command	Added (with version DLL 537, OUT 537).
486	set_fly_limits_z command	Added (with version DLL 531, OUT 532).
526	set_multi_mcbsp_in command	Added (with version DLL 537, OUT 537).
528	set_multi_mcbsp_in_list command	Added (with version DLL 537, OUT 537).
550	set_sky_writing_para command	Changed parameter <code>Timelag</code> (with version DLL 537, OUT 537).
575	set_wobbel_control command	Added (with version DLL 537, OUT 537).
576	set_wobbel_direction command	Added (with version DLL 537, OUT 537).
577	set_wobbel_mode command	Changed with version DLL 537, OUT 537: <code>Modes > 0</code> .
579	set_wobbel_offset command	Added (with version DLL 537, OUT 537).
580	set_wobbel_vector command	Added (with version DLL 537, OUT 537).

Changes from revision 1.9 to revision 1.10

Page	Name of chapter / command	Change
30	chapter 2.5.6 "RTC5 varioSCAN FLEX Extension Board"	Added.
30	chapter 2.6 "Supplementary Software"	Expanded.
54	chapter 4.4.6 "SPI / I ² C Socket Connector"	Expanded (with version DLL 538, OUT 538): The "SPI / I ² C" socket connector now can also be initialized with SPI functionality (Serial Peripheral Interface): "SPI interface". "McBSP interface" is an established term since long times in this manual is extended with "/SPI" and used as such.
84	section Repeatedly Executed Calls	Added.
128	section Mode 3 (Sky Writing)	Expanded.
129	figure 44	Corrected.
142	section Precalculation and Diagnosis	Added.
229	chapter 8.12 "Camming"	Added.
249	chapter 9.4 "Periodical I/O Signals"	Added.
270	auto_cal command	Text added: error code 9, 90.
276	camming command	Added (with version DLL 512, OUT 511).
304	get_free_variable command	Changed (with version DLL 539, OUT 539): value range of parameter No increased.
305	get_galvo_controls command	Added (with version DLL 539, OUT 539).
307	get_head_status command	Expanded.
415	mcbsp_init_spi command	Added (with version DLL 538, OUT 538).
434	periodic_toggle command	Added (with version DLL 538, OUT 538).
435	periodic_toggle_list command	Added (with version DLL 538, OUT 538).
438	read_abc_from_file command	Added (with version DLL 538, OUT 538).
495	set_free_variable command	Changed (with version DLL 539, OUT 539): value range of parameter No increased.
562	set_trigger command	Changed (with version DLL 538, OUT 538): Signal1, Signal2 = 45 and 46.
562	set_trigger command	Changed (with version DLL 539, OUT 539): Signal1, Signal2 = 47...51.
580	set_wobbel_vector command	Corrected: formula for variation of the current laser power P within the wobbel shape section.
600	sub_call_abs_repeat command	Added (with version DLL 538, OUT 538).
601	sub_call_repeat command	Added (with version DLL 538, OUT 538).
635	write_abc_to_file command	Added (with version DLL 538, OUT 538).
639	write_hi_pos command	Added (with version DLL 538, OUT 538).



Page	Name of chapter / command	Change
651	chapter 14 "Technical Specifications of the RTC5 PCI Board"	section "SPI / I2C" Connector: SPI configuration values added.
677	chapter 16.5 "Compliance with EC Guidelines for Electromagnetic Compatibility (EMC)"	Added.



Changes from revision 1.10 to revision 1.11

Page	Name of chapter / command	Change
22	chapter 1.3.1 "Delivered Software" and others	Expanded (Windows 10 is supported).
81	section "Non-Indexed Subroutines"	Expanded.
95	section "Example Code"	Added (identification of master/slave boards).
165	section Speed-Dependent Laser Control	Expanded (note on <code>set_auto_laser_control</code> Mode = 6).
287	<code>control_command</code> command	Changed: formula for coolant-flow rate at $05_{H}3B_{H}$.
287	<code>control_command</code> command	Changed: formula for protective-window scattered light value at $05_{H}3C_{H}$.
360	<code>list_call_abs_repeat</code> command	Added (with version DLL 540, OUT 540).
362	<code>list_call_repeat</code> command	Added (with version DLL 540, OUT 540).
422	<code>move_to</code> command	Command description has been moved to the manual "Installation and Operation RTC Step Motor Extension for the RTC4 and RTC5 PC interface boards" (available in English only).
464	<code>set_auto_laser_control</code> command	Changed (with version DLL 540, OUT 540): Mode = 6.
590	<code>stepper_enable</code> command	Corrected: the command does not have a result.
684	chapter 17.4 "Technical Specifications of RTC5 PCIe/104 Board"	Expanded (note on ANALOG OUT1 and ANALOG OUT2 at the RTC5 PCIe/104 Board).



Changes from revision 1.11 to revision 1.12

Page	Name of chapter / command	Change
22	chapter 1.3.1 "Delivered Software"	Expanded (the RTC5 software package now also contains files for CMake).
62	chapter 5.3.2 "Exchange of RTC Boards and Update of RTC Board Drivers"	Added. For further information, refer to the (updated as of version DLL 542) file ReadMe_ScanlabClassChecker.pdf.
208	section "Coordinate Transformations in the Virtual Image Field"	Expanded (coordinate transformations in the virtual image field are now also available with the commands <code>set_fly_x</code> and/or <code>set_fly_y</code> ; as of version DLL 541, OUT 541).
215	chapter 8.7.11 "Processing-on-the-fly Correction for the Z-Axis — "FlyZ Correction" (as of Version DLL 530, OUT 531)"	Added.
231	chapter 8.13 "Time Measurements"	Added.
287	<code>control_command</code> command	Changed: value range decreased (from 2250 to 1992) for temperature of mirror 2 at $05_{\text{H}}36_{\text{H}}$.
287	<code>control_command</code> command	Changed: value range decreased (from 2250 to 1992) for temperature of mirror 1 at $05_{\text{H}}37_{\text{H}}$.
311	<code>get_lap_time</code> command	Added (with version DLL 541, OUT 541).
369	<code>list_next</code> command	Added (with version DLL 541, OUT 541).
494	<code>set_fly_z</code> command	Added (with version DLL 530, OUT 531).
539	<code>set_pulse_picking_list</code> command	Corrected: this command is an undelayed short list command.
568	<code>set_trigger4</code> command	Corrected: this command is a delayed short list command.
589	<code>stepper_disable_switch</code> command	Added (with version DLL 542, OUT 542).



Changes from revision 1.12 to revision 1.13

Page	Name of chapter / command	Change
52	chapter 4.4.3 "EXTENSION 2 Socket Connector", figure 17	Corrected: signal at pin (26) is LASERON (not: NOT CONNECTED).
59	chapter 4.4.8 "Analog Inputs (Optional Accessory)"	Expanded: The analog input values (ANALOG IN0 and ANALOG IN1, see <code>read_analog_in</code>) can be recorded with trigger signal 54 (see <code>set_trigger</code> or <code>set_trigger4</code> ; as of version DLL 543, OUT 543, RBF 524).
189	chapter 8.4 "Wobbel Mode"	Expanded: The present wobbel amplitude (from <code>set_wobbel</code> or <code>set_wobbel_mode</code>) can be recorded with trigger signal 53 (see <code>set_trigger</code> or <code>set_trigger4</code> ; as of version DLL 543, OUT 543, RBF 524).
231	chapter 8.13 "Time Measurements"	Expanded: Time stamp counter. Can be recorded with trigger signal 52 (see <code>set_trigger</code> or <code>set_trigger4</code> ; as of version DLL 543, OUT 543, RBF 524).
244	chapter 9.3.2 "Conditional Command Execution"	Expanded: control command <code>set_pause_list_cond</code> (as of version DLL 543, OUT 543).
434	<code>periodic_toggle</code> command	Changed: endless toggling with Count = $2^{32}-1$ (as of version DLL 543, OUT 543).
435	<code>periodic_toggle_list</code> command	Changed: endless toggling with Count = $2^{32}-1$ (as of version DLL 543, OUT 543).
470	<code>set_control_mode</code> command	Expanded: bit #4 (as of version DLL 543, OUT 543).
472	<code>set_control_mode_list</code> command	Expanded: bit #4 (as of version DLL 543, OUT 543).
533	<code>set_pause_list_cond</code> command	Added (as of version DLL 543, OUT 543).
562	<code>set_trigger</code> command	Changed: Signal1, Signal2 = 52...54 (as of version DLL 543, OUT 543, RBF 524).
568	<code>set_trigger4</code> command	Changed: Signal1, Signal2 = 52...54 (as of version DLL 543, OUT 543, RBF 524).
652	Section "EXTENSION 2 Connector"	Corrected: missing signal LASERON added.